



# The L<sup>A</sup>T<sub>E</sub>X Companion

## Second Edition

TOOLS AND TECHNIQUES FOR COMPUTER TYPESetting



**Frank Mittelbach and Michel Goossens**  
with Johannes Braams, David Carlisle, and Chris Rowley

## The L<sup>A</sup>T<sub>E</sub>X Companion

*Second Edition*

**Frank Mittelbach and Michel Goossens**

with Johannes Braams, David Carlisle, and Chris Rowley

*The L<sup>A</sup>T<sub>E</sub>X Companion* has long been the essential resource for anyone using L<sup>A</sup>T<sub>E</sub>X to create high-quality printed documents. This completely updated edition brings you all the latest information about L<sup>A</sup>T<sub>E</sub>X and the vast range of add-on packages now available—over 200 are covered! Full of new tips and tricks for using L<sup>A</sup>T<sub>E</sub>X in both traditional and modern typesetting, this book will also show you how to customize layout features to your own needs—from phrases and paragraphs to headings, lists, and pages.

*Inside, you will find:*

- Expert advice on using L<sup>A</sup>T<sub>E</sub>X's basic formatting tools to create all types of publications—from memos to encyclopedias
- In-depth coverage of important extension packages for tabular and technical typesetting, floats and captions, multicolumn layouts—including reference guides and discussions of the underlying typographic and T<sub>E</sub>Xnical concepts
- Detailed techniques for generating and typesetting contents lists, bibliographies, indexes, etc.
- Tips and tricks for L<sup>A</sup>T<sub>E</sub>X programmers and systems support



The accompanying CD-ROM contains a complete plug-and-play L<sup>A</sup>T<sub>E</sub>X installation, including all the packages and examples featured in the book.

*New to this edition:*

- Nearly 1,000 fully tested examples that illustrate the text and solve typographical and technical problems—all ready to run!
- An additional chapter on citations and bibliographies
- Expanded material on the setup and use of fonts to access a huge collection of glyphs, and to typeset text from a wide range of languages and cultures
- Major new packages for graphics, “verbatim” listings, floats, and page layout
- Full coverage of the latest packages for all types of documents—mathematical, multilingual, and many more
- Detailed help on all error messages, including those troublesome low-level T<sub>E</sub>X errors

Like its predecessor, *The L<sup>A</sup>T<sub>E</sub>X Companion, Second Edition*, is an indispensable reference for anyone wishing to use L<sup>A</sup>T<sub>E</sub>X productively.

All of the authors have over 10 years of varied experience working with L<sup>A</sup>T<sub>E</sub>X-related software systems. All but one are active members of the L<sup>A</sup>T<sub>E</sub>X3 Project Team, developing and maintaining the core L<sup>A</sup>T<sub>E</sub>X system.

The authors are donating a portion of the book's royalties to the L<sup>A</sup>T<sub>E</sub>X3 project fund, which supports the maintenance and future development of L<sup>A</sup>T<sub>E</sub>X.

[www.awprofessional.com](http://www.awprofessional.com)

Cover design by Melanie Buck

Cover illustration by © David Arky/CORBIS

Text printed on recycled paper

Addison-Wesley

Pearson Education

5 5999  
  
 9 780201 362992

ISBN 0-201-36299-6

\$59.99 US  
 \$86.99 CANADA

# The L<sup>A</sup>T<sub>E</sub>X Companion

## Second Edition

## Addison-Wesley Series on Tools and Techniques for Computer Typesetting

This 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 of both users and system developers. Initial titles comprise handy references for *L<sup>A</sup>T<sub>E</sub>X* 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](mailto:mittelbach@aw.com).  
Send all other comments to the publisher: [awprofessional@aw.com](mailto:awprofessional@aw.com).

### Series Editor

Frank Mittelbach  
*Manager L<sup>A</sup>T<sub>E</sub>X3 Project, Germany*

### Editorial Board

Jacques André <i>Irisa/Inria-Rennes, France</i>	Tim Bray <i>Textuality Services, Canada</i>	Chris Rowley <i>Open University, UK</i>
Barbara Beeton <i>Editor, TUGboat, USA</i>	Peter Flynn <i>University College, Cork, Ireland</i>	Richard Rubinstein <i>Human Factors International, USA</i>
David Brailsford <i>University of Nottingham, UK</i>	Leslie Lamport <i>Creator of L<sup>A</sup>T<sub>E</sub>X, USA</i>	Paul Stiff <i>University of Reading, UK</i>

### Series Titles

- Guide to L<sup>A</sup>T<sub>E</sub>X, Fourth Edition*, by Helmut Kopka and Patrick W. Daly  
*The L<sup>A</sup>T<sub>E</sub>X Companion, Second Edition*, by Frank Mittelbach and Michel Goossens with Johannes Braams, David Carlisle, and Chris Rowley  
*The L<sup>A</sup>T<sub>E</sub>X Graphics Companion*, by Michel Goossens, Sebastian Rahtz, and Frank Mittelbach  
*The L<sup>A</sup>T<sub>E</sub>X Web Companion*, by Michel Goossens and Sebastian Rahtz

Also from Addison-Wesley:

- L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System, Second Edition*, by Leslie Lamport  
*The Unicode Standard, Version 4.0*, by the Unicode Consortium

# The L<sup>A</sup>T<sub>E</sub>X Companion

## Second Edition

Frank Mittelbach  
*L<sup>A</sup>T<sub>E</sub>X3 Project, Mainz, Germany*

Michel Goossens  
*CERN, Geneva, Switzerland*

with Johannes Braams, David Carlisle,  
and Chris Rowley

and contributions by  
Christine Detig and Joachim Schrod

◆ Addison-Wesley

Boston • San Francisco • New York • Toronto • Montreal  
London • Munich • Paris • Madrid  
Capetown • Sydney • Tokyo • Singapore • Mexico City

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 was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers discounts on this book when ordered in quantity for bulk purchases and special sales. For more information, please contact:

U.S. Corporate and Government Sales  
(800) 382-3419  
[corpsales@pearsontechgroup.com](mailto:corpsales@pearsontechgroup.com)

For sales outside of the U.S., please contact:

International Sales  
(317) 581-3793  
[international@pearsontechgroup.com](mailto:international@pearsontechgroup.com)

Visit Addison-Wesley on the Web: [www.awprofessional.com](http://www.awprofessional.com)

*Library of Congress Cataloging-in-Publication Data*

Mittelbach, Frank.

The LaTeX Companion.- 2nd ed. / Frank Mittelbach and Michel Goossens, with Johannes Braams, David Carlisle, and Chris Rowley.

p. cm.

Goossens' name appears first on the earlier edition.

Includes bibliographical references and index.

ISBN 0-201-36299-6 (pbk. : alk. paper)

1. LaTeX (Computer file) 2. Computerized typesetting. I. Goossens, Michel. II. Rowley, Chris, 1948- III. Title.

Z253.4.L38G66 2004

686.2'2544536-dc22

2003070810

Copyright © 2004 by Pearson Education, 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 consent of the publisher.  
Printed in the United States of America. Published simultaneously in Canada.

For information on obtaining permission for use of material from this work, please submit a written request to:

Pearson Education, Inc.  
Rights and Contracts Department  
75 Arlington Street, Suite 300  
Boston, MA 02116  
Fax: (617) 848-7047

Text printed on recycled and acid-free paper.

ISBN 0201362996

3 4 5 6 7 8 CRW 07 06 05



We dedicate this book to the memory of Michael Downes (1958–2003),  
a great friend and wonderful colleague on the L<sup>A</sup>T<sub>E</sub>X Team.  
His thoughtful contributions to our work and our lives are diverse  
and profound. Moreover, he brightens the lives of countless grateful  
(I)A{T}EX users through the wisdom built into his support for all  
aspects of mathematical typesetting—very many *masterpieces of the  
publishing art* will stand for ever as superb memorials to his quiet  
but deep insights.

# Contents

<b>List of Figures</b>	<b>xix</b>
<b>List of Tables</b>	<b>xxi</b>
<b>Preface</b>	<b>xxv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 A brief history . . . . .	1
1.2 Today's system . . . . .	6
1.3 Working with this book . . . . .	10
1.3.1 What's here . . . . .	10
1.3.2 Typographic conventions . . . . .	11
1.3.3 Using the examples . . . . .	14
<b>2 The Structure of a L<sup>A</sup>T<sub>E</sub>X Document</b>	<b>15</b>
2.1 The structure of a source file . . . . .	15
2.1.1 Processing of options and packages . . . . .	17
2.1.2 Splitting the source file into parts . . . . .	18
2.1.3 Combining several files . . . . .	20
2.1.4 <code>optional</code> —Providing variants in the document source . . .	21
2.2 Sectioning commands . . . . .	22
2.2.1 Numbering headings . . . . .	24
2.2.2 Formatting headings . . . . .	27
2.2.3 Changing fixed heading texts . . . . .	34
2.2.4 <code>fncychap</code> —Predefined chapter heading layouts . . . . .	34
2.2.5 <code>quotchap</code> —Mottos on chapters . . . . .	35
2.2.6 <code>titlesec</code> —A different approach to headings . . . . .	36

2.3	Table of contents structures . . . . .	45
2.3.1	Entering information into the contents files . . . . .	46
2.3.2	Typesetting a contents list . . . . .	49
2.3.3	Combining contents lists . . . . .	52
2.3.4	Providing additional contents files . . . . .	54
2.3.5	<i>shorttoc</i> —Summary table of contents . . . . .	55
2.3.6	<i>minitoc</i> —Multiple tables of contents . . . . .	56
2.3.7	<i>titletoc</i> —A different approach to contents lists . . . . .	58
2.4	Managing references . . . . .	66
2.4.1	<i>showkeys</i> —Displaying the reference keys . . . . .	68
2.4.2	<i>varioref</i> —More flexible cross-references . . . . .	68
2.4.3	<i>prettyref</i> —Adding frills to references . . . . .	75
2.4.4	<i>titleref</i> —Non-numerical references . . . . .	76
2.4.5	<i>hyperref</i> —Active references . . . . .	78
2.4.6	<i>xr</i> —References to external documents . . . . .	78
3	<b>Basic Formatting Tools</b>	79
3.1	Phrases and paragraphs . . . . .	80
3.1.1	<i>xspace</i> —Gentle spacing after a macro . . . . .	80
3.1.2	<i>ellipsis, lips</i> —Marks of omission . . . . .	81
3.1.3	<i>amsmath</i> —Nonbreaking dashes . . . . .	83
3.1.4	<i>relsize</i> —Relative changes to the font size . . . . .	83
3.1.5	<i>textcase</i> —Change case of text intelligently . . . . .	85
3.1.6	<i>ulem</i> —Emphasize via underline . . . . .	87
3.1.7	<i>soul</i> —Letterspacing or stealing sheep . . . . .	88
3.1.8	<i>url</i> —Typesetting URLs, path names, and the like . . . . .	93
3.1.9	<i>euro</i> —Converting and typesetting currencies . . . . .	96
3.1.10	<i>lettrine</i> —Dropping your capital . . . . .	99
3.1.11	Paragraph justification in L <sup>A</sup> T <sub>E</sub> X . . . . .	102
3.1.12	<i>ragged2e</i> —Enhancing justification . . . . .	105
3.1.13	<i>setspace</i> —Changing interline spacing . . . . .	106
3.1.14	<i>picinpar</i> —Making rectangular holes . . . . .	108
3.2	Footnotes, endnotes, and marginals . . . . .	109
3.2.1	Using standard footnotes . . . . .	110
3.2.2	Customizing standard footnotes . . . . .	112
3.2.3	<i>ftnright</i> —Right footnotes in a two-column environment . . . . .	114
3.2.4	<i>footmisc</i> —Various footnotes styles . . . . .	114
3.2.5	<i>perpage</i> —Resetting counters on a “per-page” basis . . . . .	120
3.2.6	<i>manyfoot</i> —Independent footnotes . . . . .	122
3.2.7	<i>endnotes</i> —An alternative to footnotes . . . . .	125
3.2.8	Marginal notes . . . . .	126
3.3	List structures . . . . .	128
3.3.1	Modifying the standard lists . . . . .	128
3.3.2	<i>paralist</i> —Extended list environments . . . . .	132

3.3.3	<i>amsthm</i> —Providing headed lists . . . . .	138
3.3.4	Making your own lists . . . . .	144
3.4	Simulating typed text . . . . .	151
3.4.1	Simple verbatim extensions . . . . .	152
3.4.2	<i>upquote</i> —Computer program style quoting . . . . .	153
3.4.3	<i>fancyvrb</i> —Highly customizable verbatim environments . .	155
3.4.4	<i>listings</i> —Pretty-printing program code . . . . .	168
3.5	Lines and columns . . . . .	175
3.5.1	<i>lineno</i> —Numbering lines of text . . . . .	176
3.5.2	<i>parallel</i> —Two text streams aligned . . . . .	181
3.5.3	<i>multicol</i> —A flexible way to handle multiple columns . .	184
3.5.4	<i>changebar</i> —Adding revision bars to documents . . . . .	189
<b>4</b>	<b>The Layout of the Page</b>	<b>193</b>
4.1	Geometrical dimensions of the layout . . . . .	193
4.2	Changing the layout . . . . .	197
4.2.1	<i>layouts</i> —Displaying your layout . . . . .	199
4.2.2	A collection of page layout packages . . . . .	202
4.2.3	<i>typearea</i> —A traditional approach . . . . .	203
4.2.4	<i>geometry</i> —Layout specification with auto-completion . .	206
4.2.5	<i>lscape</i> —Typesetting individual pages in landscape mode .	211
4.2.6	<i>crop</i> —Producing trimming marks . . . . .	212
4.3	Dynamic page data: page numbers and marks . . . . .	215
4.3.1	<i>L<sup>A</sup>T<sub>E</sub>X</i> page numbers . . . . .	215
4.3.2	<i>lastpage</i> —A way to reference it . . . . .	216
4.3.3	<i>chappg</i> —Page numbers by chapters . . . . .	216
4.3.4	<i>L<sup>A</sup>T<sub>E</sub>X</i> mark commands . . . . .	217
4.3.5	<i>extramarks</i> —Providing new marks . . . . .	220
4.4	Page styles . . . . .	221
4.4.1	The low-level page style interface . . . . .	223
4.4.2	<i>fancyhdr</i> —Customizing page styles . . . . .	224
4.4.3	<i>truncate</i> —Truncate text to a given length . . . . .	232
4.5	Visual formatting . . . . .	234
4.5.1	<i>nextpage</i> —Extensions to <i>\clearpage</i> . . . . .	235
4.6	Doing layout with class . . . . .	236
4.6.1	<i>KOMA-Script</i> —A drop-in replacement for <i>article</i> et al. . .	236
4.6.2	<i>memoir</i> —Producing complex publications . . . . .	237
<b>5</b>	<b>Tabular Material</b>	<b>239</b>
5.1	Standard <i>L<sup>A</sup>T<sub>E</sub>X</i> environments . . . . .	240
5.1.1	Using the <i>tabbing</i> environment . . . . .	241
5.1.2	Using the <i>tabular</i> environment . . . . .	242
5.2	<i>array</i> —Extending the <i>tabular</i> environments . . . . .	243
5.2.1	Examples of preamble commands . . . . .	244

5.2.2	Defining new column specifiers . . . . .	248
5.3	Calculating column widths . . . . .	249
5.3.1	Explicit calculation of column widths . . . . .	250
5.3.2	<i>tabularx</i> —Automatic calculation of column widths . . . . .	251
5.3.3	<i>tabulary</i> —Column widths based on content . . . . .	253
5.3.4	Differences between <i>tabular*</i> , <i>tabularx</i> , and <i>tabulary</i> . . . . .	255
5.4	Multipage tabular material . . . . .	255
5.4.1	<i>supertabular</i> —Making multipage tabulars . . . . .	256
5.4.2	<i>longtable</i> —Alternative multipage tabulars . . . . .	259
5.5	Color in tables . . . . .	264
5.6	Customizing table rules and spacing . . . . .	265
5.6.1	Colored table rules . . . . .	265
5.6.2	Variable-width rules . . . . .	266
5.6.3	<i>hhline</i> —Combining horizontal and vertical lines . . . . .	266
5.6.4	<i>arydshln</i> —Dashed rules . . . . .	267
5.6.5	<i>tbls</i> —Controlling row spacing . . . . .	269
5.6.6	<i>booktabs</i> —Formal ruled tables . . . . .	269
5.7	Further extensions . . . . .	272
5.7.1	<i>multirow</i> —Vertical alignment in tables . . . . .	273
5.7.2	<i>dcolumn</i> —Decimal column alignments . . . . .	274
5.8	Footnotes in tabular material . . . . .	277
5.8.1	Using <i>minipage</i> footnotes with tables . . . . .	277
5.8.2	<i>threeparttable</i> —Setting table and notes together . . . . .	278
5.9	Applications . . . . .	279
5.9.1	Managing tables with wide entries . . . . .	279
5.9.2	Tables inside tables . . . . .	280
<b>6</b>	<b>Mastering Floats</b>	<b>283</b>
6.1	Understanding float parameters . . . . .	284
6.2	Float placement control . . . . .	286
6.2.1	<i>placeins</i> —Preventing floats from crossing a barrier . . . . .	288
6.2.2	<i>afterpage</i> —Taking control at the page boundary . . . . .	289
6.2.3	<i>endfloat</i> —Placing figures and tables at the end . . . . .	289
6.3	Extensions to L <sup>A</sup> T <sub>E</sub> X's float concept . . . . .	291
6.3.1	<i>float</i> —Creating new float types . . . . .	291
6.3.2	<i>caption</i> —For nonfloating figures and tables . . . . .	295
6.3.3	<i>rotating</i> —Rotating floats . . . . .	296
6.3.4	<i>rotfloat</i> —Combining <i>float</i> and <i>rotating</i> . . . . .	298
6.4	Inline floats . . . . .	298
6.4.1	<i>wrapfig</i> —Wrapping text around a figure . . . . .	299
6.4.2	<i>picins</i> —Placing pictures inside the text . . . . .	302
6.5	Controlling the float caption . . . . .	306
6.5.1	<i>caption</i> —Customizing your captions . . . . .	308
6.5.2	<i>subfig</i> —Substructuring floats . . . . .	315

6.5.3	<code>subfloat</code> —Sub-numbering floats . . . . .	321
6.5.4	<code>sidecap</code> —Place captions sideways . . . . .	323
6.5.5	<code>fltpage</code> —Captions on a separate page . . . . .	325
<b>7</b>	<b>Fonts and Encodings</b>	<b>327</b>
7.1	Introduction . . . . .	327
7.1.1	The history of L <sup>A</sup> T <sub>E</sub> X's font selection scheme (NFSS) . . . . .	327
7.1.2	Input and output encodings . . . . .	329
7.2	Understanding font characteristics . . . . .	331
7.2.1	Monospaced and proportional fonts . . . . .	331
7.2.2	Serifed and sans serif fonts . . . . .	332
7.2.3	Font families and their attributes . . . . .	333
7.2.4	Font encodings . . . . .	336
7.3	Using fonts in text . . . . .	337
7.3.1	Standard L <sup>A</sup> T <sub>E</sub> X font commands . . . . .	338
7.3.2	Combining standard font commands . . . . .	343
7.3.3	Font commands versus declarations . . . . .	344
7.3.4	Accessing all characters of a font . . . . .	345
7.3.5	Changing the default text fonts . . . . .	346
7.3.6	L <sup>A</sup> T <sub>E</sub> X 2.09 font commands . . . . .	347
7.4	Using fonts in math . . . . .	347
7.4.1	Special math alphabet identifiers . . . . .	348
7.4.2	Text font commands in math . . . . .	351
7.4.3	Mathematical formula versions . . . . .	352
7.5	Standard L <sup>A</sup> T <sub>E</sub> X font support . . . . .	353
7.5.1	Computer Modern—The L <sup>A</sup> T <sub>E</sub> X standard fonts . . . . .	353
7.5.2	<code>inputenc</code> —Selecting the input encoding . . . . .	357
7.5.3	<code>fontenc</code> —Selecting font encodings . . . . .	361
7.5.4	<code>textcomp</code> —Providing additional text symbols . . . . .	362
7.5.5	<code>exscale</code> —Scaling large operators . . . . .	368
7.5.6	<code>tracefnt</code> —Tracing the font selection . . . . .	368
7.5.7	<code>nfssfont.tex</code> —Displaying font tables and samples . . . . .	369
7.6	<code>PSNFSS</code> —PostScript fonts with L <sup>A</sup> T <sub>E</sub> X . . . . .	370
7.6.1	Font samples for fonts supported by PSNFSS . . . . .	373
7.6.2	<code>mathptmx</code> —Times Roman in math and text . . . . .	376
7.6.3	<code>mathpazo</code> —Palatino in math and text . . . . .	377
7.6.4	<code>pifont</code> —Accessing Pi and Symbol fonts . . . . .	378
7.7	A collection of font packages . . . . .	381
7.7.1	<code>eco</code> —Old-style numerals with Computer Modern . . . . .	381
7.7.2	<code>ccfonts</code> , <code>concmath</code> —The Concrete fonts . . . . .	383
7.7.3	<code>cmbright</code> —The Computer Modern Bright fonts . . . . .	385
7.7.4	<code>luximono</code> —A general-purpose typewriter font . . . . .	386
7.7.5	<code>txfonts</code> —Alternative support for Times Roman . . . . .	388
7.7.6	<code>pxfonts</code> —Alternative support for Palatino . . . . .	390

7.7.7	The Fourier-GUTenberg fonts . . . . .	391
7.7.8	The URW Antiqua and Grotesk fonts . . . . .	393
7.7.9	yfonts—Typesetting with Old German fonts . . . . .	394
7.7.10	euler, eulervm—Accessing the Euler fonts . . . . .	396
7.8	The L <sup>A</sup> T <sub>E</sub> X world of symbols . . . . .	399
7.8.1	dingbat—A selection of hands . . . . .	400
7.8.2	wasysym—Waldi's symbol font . . . . .	401
7.8.3	marvosym—Interface to the MarVoSym font . . . . .	401
7.8.4	bding—A METAFONT alternative to Zapf Dingbats . . . . .	403
7.8.5	ifsym—Clocks, clouds, mountains, and other symbols . . . . .	403
7.8.6	tipa—International Phonetic Alphabet symbols . . . . .	405
7.8.7	Typesetting the euro symbol (€) . . . . .	407
7.9	The low-level interface . . . . .	412
7.9.1	Setting individual font attributes . . . . .	413
7.9.2	Setting several font attributes . . . . .	417
7.9.3	Automatic substitution of fonts . . . . .	418
7.9.4	Using low-level commands in the document . . . . .	418
7.10	Setting up new fonts . . . . .	419
7.10.1	Overview . . . . .	419
7.10.2	Naming those thousands of fonts . . . . .	420
7.10.3	Declaring new font families and font shape groups . . . . .	421
7.10.4	Modifying font families and font shape groups . . . . .	429
7.10.5	Declaring new font encoding schemes . . . . .	430
7.10.6	Internal file organization . . . . .	431
7.10.7	Declaring new fonts for use in math . . . . .	432
7.10.8	Example: Defining your own .fd files . . . . .	437
7.10.9	The order of declaration . . . . .	439
7.11	L <sup>A</sup> T <sub>E</sub> X's encoding models . . . . .	440
7.11.1	Character data within the L <sup>A</sup> T <sub>E</sub> X system . . . . .	440
7.11.2	L <sup>A</sup> T <sub>E</sub> X's internal character representation (LICR) . . . . .	442
7.11.3	Input encodings . . . . .	443
7.11.4	Output encodings . . . . .	447
7.12	Compatibility packages for very old documents . . . . .	463
7.12.1	oldlfont, rawfonts, newlfont—Processing old documents . . . . .	463
7.12.2	latexsym—Providing symbols from L <sup>A</sup> T <sub>E</sub> X 2.09 lasy fonts . . . . .	464
<b>8</b>	<b>Higher Mathematics</b> . . . . .	<b>465</b>
8.1	Introduction to <i>AMSLATEX</i> . . . . .	466
8.2	Display and alignment structures for equations . . . . .	468
8.2.1	Comparison with standard L <sup>A</sup> T <sub>E</sub> X . . . . .	470
8.2.2	A single equation on one line . . . . .	471
8.2.3	A single equation on several lines: no alignment . . . . .	471
8.2.4	A single equation on several lines: with alignment . . . . .	473
8.2.5	Equation groups without alignment . . . . .	474

8.2.6	Equation groups with simple alignment . . . . .	475
8.2.7	Multiple alignments: <code>align</code> and <code>flalign</code> . . . . .	475
8.2.8	Display environments as mini-pages . . . . .	477
8.2.9	Interrupting displays: <code>\intertext</code> . . . . .	479
8.2.10	Vertical space and page breaks in and around displays . . . . .	479
8.2.11	Equation numbering and tags . . . . .	482
8.2.12	Fine-tuning tag placement . . . . .	483
8.2.13	Subordinate numbering sequences . . . . .	484
8.2.14	Resetting the equation counter . . . . .	485
8.3	Matrix-like environments . . . . .	485
8.3.1	The <code>cases</code> environment . . . . .	486
8.3.2	The <code>matrix</code> environments . . . . .	486
8.3.3	Stacking in subscripts and superscripts . . . . .	487
8.3.4	Commutative diagrams . . . . .	488
8.3.5	<code>delarray</code> —Delimiters surrounding an array . . . . .	489
8.4	Compound structures and decorations . . . . .	490
8.4.1	Decorated arrows . . . . .	490
8.4.2	Continued fractions . . . . .	490
8.4.3	Boxed formulas . . . . .	491
8.4.4	Limiting positions . . . . .	491
8.4.5	Multiple integral signs . . . . .	492
8.4.6	Modular relations . . . . .	492
8.4.7	Fractions and generalizations . . . . .	493
8.4.8	Dottier accents . . . . .	494
8.4.9	<code>amsxtra</code> —Accents as superscripts . . . . .	495
8.4.10	Extra decorations . . . . .	495
8.5	Variable symbol commands . . . . .	495
8.5.1	Ellipsis . . . . .	496
8.5.2	Horizontal extensions . . . . .	497
8.5.3	Vertical extensions . . . . .	498
8.6	Words in mathematics . . . . .	499
8.6.1	The <code>\text</code> command . . . . .	499
8.6.2	Operator and function names . . . . .	499
8.7	Fine-tuning the mathematical layout . . . . .	502
8.7.1	Controlling the automatic sizing and spacing . . . . .	502
8.7.2	Sub-formulas . . . . .	503
8.7.3	Big-g delimiters . . . . .	504
8.7.4	Radical movements . . . . .	504
8.7.5	<code>Ghostbusters™</code> . . . . .	505
8.7.6	Horizontal spaces . . . . .	507
8.8	Fonts in formulas . . . . .	508
8.8.1	Additional math font commands . . . . .	509
8.8.2	<code>bm</code> —Making bold . . . . .	510
8.8.3	A collection of math font set-ups . . . . .	513

8.9	Symbols in formulas . . . . .	524
8.9.1	Mathematical symbol classes . . . . .	524
8.9.2	Letters, numerals, and other Ordinary symbols . . . . .	526
8.9.3	Mathematical accents . . . . .	529
8.9.4	Binary operator symbols . . . . .	529
8.9.5	Relation symbols . . . . .	531
8.9.6	Punctuation . . . . .	535
8.9.7	Operator symbols . . . . .	536
8.9.8	Opening and Closing symbols . . . . .	537
<b>9</b>	<b>L<sup>A</sup>T<sub>E</sub>X in a Multilingual Environment</b>	<b>539</b>
9.1	T <sub>E</sub> X and non-English languages . . . . .	539
9.1.1	Language-related aspects of typesetting . . . . .	541
9.1.2	Culture-related aspects of typesetting . . . . .	542
9.1.3	Babel—L <sup>A</sup> T <sub>E</sub> X speaks multiple languages . . . . .	542
9.2	The <code>babel</code> user interface . . . . .	543
9.2.1	Setting or getting the current language . . . . .	544
9.2.2	Handling shorthands . . . . .	547
9.2.3	Language attributes . . . . .	549
9.3	User commands provided by language options . . . . .	550
9.3.1	Translations . . . . .	550
9.3.2	Available shorthands . . . . .	550
9.3.3	Language-specific commands . . . . .	558
9.3.4	Layout considerations . . . . .	564
9.3.5	Languages and font encoding . . . . .	566
9.4	Support for non-Latin alphabets . . . . .	569
9.4.1	The Cyrillic alphabet . . . . .	569
9.4.2	The Greek alphabet . . . . .	574
9.4.3	The Hebrew alphabet . . . . .	576
9.5	Tailoring <code>babel</code> . . . . .	579
9.5.1	Hyphenating in several languages . . . . .	580
9.5.2	The package file . . . . .	581
9.5.3	The structure of the <code>babel</code> language definition file . . . . .	582
9.6	Other approaches . . . . .	591
9.6.1	More complex languages . . . . .	591
9.6.2	Omega . . . . .	592
<b>10</b>	<b>Graphics Generation and Manipulation</b>	<b>593</b>
10.1	Producing portable graphics and ornaments . . . . .	595
10.1.1	<code>boxedminipage</code> —Boxes with frames . . . . .	595
10.1.2	<code>shadow</code> —Boxes with shadows . . . . .	595
10.1.3	<code>fancybox</code> —Ornamental boxes . . . . .	596
10.1.4	<code>epic</code> —An enhanced <code>picture</code> environment . . . . .	600
10.1.5	<code>eepic</code> —Extending the <code>epic</code> package . . . . .	607
10.1.6	Special-purpose languages . . . . .	611

10.2	$\text{\LaTeX}$ 's device-dependent graphics support . . . . .	613
10.2.1	Options for <code>graphics</code> and <code>graphicx</code> . . . . .	614
10.2.2	The <code>\includegraphics</code> syntax in the <code>graphics</code> package .	616
10.2.3	The <code>\includegraphics</code> syntax in the <code>graphicx</code> package .	618
10.2.4	Setting default key values for the <code>graphicx</code> package .	623
10.2.5	Declarations guiding the inclusion of images . . . . .	624
10.2.6	A caveat: Encapsulation is important . . . . .	627
10.3	Manipulating graphical objects in $\text{\LaTeX}$ . . . . .	628
10.3.1	Scaling a $\text{\LaTeX}$ box . . . . .	628
10.3.2	Resizing to a given size . . . . .	629
10.3.3	Rotating a $\text{\LaTeX}$ box . . . . .	630
10.3.4	<code>rotating</code> —Revisited . . . . .	633
10.4	Display languages: PostScript, PDF, and SVG . . . . .	634
10.4.1	The PostScript language . . . . .	635
10.4.2	<code>dvips</code> PostScript driver . . . . .	637
10.4.3	<code>pspicture</code> —An enhanced <code>picture</code> environment for <code>dvips</code> .	638
10.4.4	The Portable Document Format . . . . .	642
10.4.5	Scalable Vector Graphics . . . . .	644
<b>11</b>	<b>Index Generation</b>	<b>647</b>
11.1	Syntax of the index entries . . . . .	648
11.1.1	Simple index entries . . . . .	650
11.1.2	Generating subentries . . . . .	650
11.1.3	Page ranges and cross-references . . . . .	651
11.1.4	Controlling the presentation form . . . . .	651
11.1.5	Printing special characters . . . . .	652
11.1.6	Creating a glossary . . . . .	653
11.1.7	Defining your own index commands . . . . .	653
11.1.8	Special considerations . . . . .	654
11.2	<code>makeindex</code> —A program to format and sort indexes .	654
11.2.1	Generating the formatted index . . . . .	655
11.2.2	Detailed options of the <i>MakeIndex</i> program . . . . .	655
11.2.3	Error messages . . . . .	658
11.2.4	Customizing the index with <i>MakeIndex</i> . . . . .	659
11.2.5	<i>MakeIndex</i> pitfalls . . . . .	665
11.3	<code>xindy</code> —An alternative to <i>MakeIndex</i> . . . . .	666
11.3.1	Generating the formatted index with <code>xindy</code> . . . . .	668
11.3.2	International indexing with <code>xindy</code> . . . . .	669
11.3.3	Modules for common tasks . . . . .	671
11.3.4	Style files for individual solutions . . . . .	673
11.4	Enhancing the index with $\text{\LaTeX}$ features . . . . .	679
11.4.1	Modifying the layout . . . . .	679
11.4.2	<code>showidx</code> , <code>repeatindex</code> , <code>tocbibind</code> , <code>indxcite</code> —Little helpers .	680
11.4.3	<code>index</code> —Producing multiple indexes . . . . .	681

<b>12 Managing Citations</b>	<b>683</b>
12.1 Introduction . . . . .	683
12.1.1 Bibliographical reference schemes . . . . .	684
12.1.2 Markup structure for citations and bibliography . . . . .	686
12.1.3 Using $\text{\LaTeX}$ to produce the bibliography input . . . . .	687
12.2 The number-only system . . . . .	691
12.2.1 Standard $\text{\LaTeX}$ —Reference by number . . . . .	691
12.2.2 <code>cite</code> —Enhanced references by number . . . . .	693
12.2.3 <code>notoccite</code> —Solving a problem with unsorted citations . . . . .	697
12.3 The author-date system . . . . .	698
12.3.1 Early attempts . . . . .	699
12.3.2 <code>natbib</code> —Customizable author-date references . . . . .	700
12.3.3 <code>bibentry</code> —Full bibliographic entries in running text . . . . .	710
12.4 The author-number system . . . . .	712
12.4.1 <code>natbib</code> —Revisited . . . . .	712
12.5 The short-title system . . . . .	715
12.5.1 <code>jurabib</code> —Customizable short-title references . . . . .	715
12.5.2 <code>camel</code> —Dedicated law support . . . . .	743
12.6 Multiple bibliographies in one document . . . . .	745
12.6.1 <code>chapterbib</code> —Bibliographies per included file . . . . .	747
12.6.2 <code>bibunits</code> —Bibliographies for arbitrary units . . . . .	749
12.6.3 <code>bibtopic</code> —Combining references by topic . . . . .	753
12.6.4 <code>multibib</code> —Separate global bibliographies . . . . .	755
<b>13 Bibliography Generation</b>	<b>757</b>
13.1 The $\text{\BibTeX}$ program and some variants . . . . .	758
13.1.1 <code>bibtex8</code> —An 8-bit reimplementation of $\text{\BibTeX}$ . . . . .	759
13.1.2 Recent developments . . . . .	759
13.2 The $\text{\BibTeX}$ database format . . . . .	761
13.2.1 Entry types and fields . . . . .	762
13.2.2 The text part of a field explained . . . . .	764
13.2.3 Abbreviations in $\text{\BibTeX}$ . . . . .	769
13.2.4 The $\text{\BibTeX}$ preamble . . . . .	771
13.2.5 Cross-referencing entries . . . . .	772
13.3 On-line bibliographies . . . . .	773
13.4 Bibliography database management tools . . . . .	774
13.4.1 <code>biblist</code> —Printing $\text{\BibTeX}$ database files . . . . .	774
13.4.2 <code>bibtools</code> —A collection of command-line tools . . . . .	775
13.4.3 <code>bibclean</code> , etc.—A second set of command-line tools . . . . .	777
13.4.4 <code>bibtool</code> —A multipurpose command-line tool . . . . .	778
13.4.5 <code>pybliographer</code> —An extensible bibliography manager . . . . .	784
13.4.6 <code>JBibtexManager</code> —A $\text{\BibTeX}$ database manager in Java . . . . .	787
13.4.7 <code>BibTexMng</code> —A $\text{\BibTeX}$ database manager for Windows . . . . .	789

13.5	Formatting the bibliography with <code>B<small>IB</small>T<small>EX</small></code> styles . . . . .	790
13.5.1	A collection of <code>B<small>IB</small>T<small>EX</small></code> style files . . . . .	791
13.5.2	<code>custom-bib</code> —Generate <code>B<small>IB</small>T<small>EX</small></code> styles with ease . . . . .	798
13.6	The <code>B<small>IB</small>T<small>EX</small></code> style language . . . . .	805
13.6.1	The <code>B<small>IB</small>T<small>EX</small></code> style file commands and built-in functions . . . . .	805
13.6.2	The documentation style <code>btxbst.doc</code> . . . . .	806
13.6.3	Introducing small changes in a style file . . . . .	809
<b>14</b>	<b>L<small>A</small>T<small>E</small>X Package Documentation Tools</b>	<b>813</b>
14.1	<code>doc</code> —Documenting L <small>A</small> T <small>E</small> X and other code . . . . .	813
14.1.1	General conventions for the source file . . . . .	814
14.1.2	Describing new macros and environments . . . . .	815
14.1.3	Cross-referencing all macros used . . . . .	817
14.1.4	The documentation driver . . . . .	818
14.1.5	Conditional code in the source . . . . .	819
14.2	<code>docstrip.tex</code> —Producing ready-to-run code . . . . .	824
14.2.1	Invocation of the <code>DOCSTRIP</code> utility . . . . .	825
14.2.2	<code>DOCSTRIP</code> script commands . . . . .	826
14.2.3	Installation support and configuration . . . . .	830
14.2.4	Using <code>DOCSTRIP</code> with other languages . . . . .	833
14.3	<code>ltxdoc</code> —A simple L <small>A</small> T <small>E</small> X documentation class . . . . .	834
14.3.1	Extensions provided by <code>ltxdoc</code> . . . . .	834
14.3.2	Customizing the output of documents that use <code>ltxdoc</code> . . . . .	835
14.4	Making use of version control tools . . . . .	836
14.4.1	<code>rcs</code> —Accessing individual keywords . . . . .	837
14.4.2	<code>rcsinfo</code> —Parsing the <code>\$Id\$</code> keyword . . . . .	838
<b>A</b>	<b>A L<small>A</small>T<small>E</small>X Overview for Preamble, Package, and Class Writers</b>	<b>841</b>
A.1	Linking markup and formatting . . . . .	841
A.1.1	Command and environment names . . . . .	842
A.1.2	Defining new commands . . . . .	843
A.1.3	Defining new environments . . . . .	847
A.1.4	Defining and changing counters . . . . .	851
A.1.5	Defining and changing space parameters . . . . .	854
A.2	Page markup—Boxes and rules . . . . .	860
A.2.1	LR boxes . . . . .	860
A.2.2	Paragraph boxes . . . . .	862
A.2.3	Rule boxes . . . . .	866
A.2.4	Manipulating boxed material . . . . .	868
A.2.5	Box commands and color . . . . .	870
A.3	Control structure extensions . . . . .	871
A.3.1	<code>calc</code> —Arithmetic calculations . . . . .	871
A.3.2	<code>ifthen</code> —Advanced control structures . . . . .	872

A.4	Package and class file structure . . . . .	877
A.4.1	The identification part . . . . .	877
A.4.2	The initial code part . . . . .	880
A.4.3	The declaration of options . . . . .	880
A.4.4	The execution of options . . . . .	881
A.4.5	The package loading part . . . . .	882
A.4.6	The main code part . . . . .	883
A.4.7	Special commands for package and class files . . . . .	883
A.4.8	Special commands for class files . . . . .	886
A.4.9	A minimal class file . . . . .	888
<b>B</b>	<b>Tracing and Resolving Problems</b>	<b>889</b>
B.1	Error messages . . . . .	890
B.1.1	Dying with memory exceeded . . . . .	915
B.2	Warnings and informational messages . . . . .	920
B.3	T <sub>E</sub> X and L <sup>A</sup> T <sub>E</sub> X commands for tracing . . . . .	931
B.3.1	Displaying command definitions and register values . . . . .	932
B.3.2	Diagnosing page-breaking problems . . . . .	935
B.3.3	Diagnosing and solving paragraph-breaking problems . . . . .	939
B.3.4	Other low-level tracing tools . . . . .	943
B.3.5	trace—Selectively tracing command execution . . . . .	945
<b>C</b>	<b>L<sup>A</sup>T<sub>E</sub>X Software and User Group Information</b>	<b>947</b>
C.1	Getting help . . . . .	947
C.2	How to get those T <sub>E</sub> X files? . . . . .	948
C.3	Using CTAN . . . . .	950
C.3.1	Finding files on the archive . . . . .	950
C.3.2	Using the T <sub>E</sub> X file catalogue . . . . .	950
C.3.3	Getting multiple files . . . . .	952
C.4	Finding the documentation on your T <sub>E</sub> X system . . . . .	954
C.4.1	texdoc—Command-line interface for a search by name . . . . .	954
C.4.2	texdoctk—Panel interface for a search by subject . . . . .	955
C.5	T <sub>E</sub> X user groups . . . . .	956
<b>D</b>	<b>TLC2 T<sub>E</sub>X CD</b>	<b>959</b>
	<b>Bibliography</b>	<b>963</b>
	<b>Index of Commands and Concepts</b>	<b>983</b>
	<b>People</b>	<b>1080</b>
	<b>Biographies</b>	<b>1083</b>
	<b>Production Notes</b>	<b>1089</b>

# List of Figures

1.1	Data flow in the $\text{\LaTeX}$ system . . . . .	9
2.1	The layout for a display heading . . . . .	28
2.2	The layout for a run-in heading . . . . .	29
2.3	Parameters defining the layout of a contents file . . . . .	51
3.1	Schematic layout of footnotes . . . . .	113
3.2	The placement of text and footnotes with the <code>ftnright</code> package . . . . .	115
3.3	Parameters used by the <code>list</code> environment . . . . .	145
4.1	Page layout parameters and visualization . . . . .	194
4.2	Schematic overview of how $\text{\LaTeX}$ 's marker mechanism works . . . . .	219
6.1	Spacing layout of the <code>subfig</code> package . . . . .	317
7.1	Major font characteristics . . . . .	332
7.2	Comparison of serifed and sans serif letters . . . . .	332
7.3	Comparison between upright and italic shapes . . . . .	333
7.4	Comparison between caps and small caps . . . . .	334
7.5	Outline and shaded shapes . . . . .	335
7.6	Scaled and designed fonts (Computer Modern) . . . . .	336
8.1	Sample page typeset with Computer Modern fonts . . . . .	513
8.2	Sample page typeset with Concrete fonts . . . . .	514
8.3	Sample page typeset with Concrete and Euler fonts . . . . .	514

8.4	Sample page typeset with Fourier fonts . . . . .	515
8.5	Sample page typeset with Times and Symbol . . . . .	516
8.6	Sample page typeset with Times and TX fonts . . . . .	516
8.7	Sample page typeset with Times and TM Math fonts . . . . .	517
8.8	Sample page typeset with Palatino and Math Pazo . . . . .	518
8.9	Sample page typeset with Palatino and PX fonts . . . . .	518
8.10	Sample page typeset with Palatino and PA Math fonts . . . . .	519
8.11	Sample page typeset with Baskerville fonts . . . . .	520
8.12	Sample page typeset with Charter fonts . . . . .	520
8.13	Sample page typeset with Lucida Bright . . . . .	521
8.14	Sample page typeset with CM Bright fonts . . . . .	522
8.15	Sample page typeset with Helvetica Math fonts . . . . .	522
8.16	Sample page typeset with Info Math fonts . . . . .	523
9.1	A Hebrew document . . . . .	577
10.1	The contents of the file <code>w.eps</code> . . . . .	616
10.2	A $\text{\LaTeX}$ box and possible origin reference points . . . . .	632
10.3	SVG generated from a <code>dvi</code> file . . . . .	646
11.1	The sequential flow of index processing . . . . .	648
11.2	Stepwise development of index processing . . . . .	649
11.3	Example of <code>\index</code> commands and the <code>showidx</code> package . . . . .	656
11.4	Printing the index and the output of the <code>showidx</code> option . . . . .	656
11.5	Example of the use of special characters with <code>MakeIndex</code> . . . . .	663
11.6	Example of customizing the output format of an index . . . . .	663
11.7	Adding leaders to an index . . . . .	664
11.8	<code>xindy</code> process model . . . . .	674
12.1	Data flow when running $\text{\BibTeX}$ and $\text{\LaTeX}$ . . . . .	688
12.2	Sample $\text{\BibTeX}$ database <code>tex.bib</code> . . . . .	690
12.3	Sample $\text{\BibTeX}$ database <code>jura.bib</code> . . . . .	717
13.1	Output of the program <code>printbib</code> . . . . .	776
13.2	Output of the program <code>bib2html</code> . . . . .	777
13.3	The <code>pybliographic</code> work space . . . . .	785
13.4	Native editing in <code>pybliographic</code> . . . . .	786
13.5	The <code>JBibTeXManager</code> work space . . . . .	788
13.6	The <code>BibTeXMng</code> work space . . . . .	790
A.1	An example of a class file extending <code>article</code> . . . . .	886
C.1	The $\text{\TeX}$ Users Group web home page . . . . .	949
C.2	Using the CTAN web interface . . . . .	951
C.3	Graham Williams' $\text{\TeX}$ catalogue on the web . . . . .	952
C.4	Finding documentation with the <code>texdoctk</code> program . . . . .	955

# List of Tables

1.1	Major file types used by $\text{\TeX}$ and $\text{\LaTeX}$ . . . . .	8
2.1	$\text{\LaTeX}$ 's standard sectioning commands . . . . .	23
2.2	Language-dependent strings for headings . . . . .	34
2.3	A summary of the <code>minitoc</code> parameters . . . . .	57
3.1	ISO currency codes of the <i>euro</i> and the 12 <i>euro-zone</i> countries . . . . .	97
3.2	Parameters used by <code>ragged2e</code> . . . . .	106
3.3	Effective <code>\baselinestretch</code> values for different font sizes . . . . .	108
3.4	Footnote symbol lists predefined by <code>footmisc</code> . . . . .	117
3.5	Commands controlling an <code>itemize</code> list environment . . . . .	128
3.6	Commands controlling an <code>enumerate</code> list environment . . . . .	130
3.7	Languages supported by <code>listings</code> (Winter 2003) . . . . .	169
3.8	Length parameters used by <code>multicols</code> . . . . .	185
3.9	Counters used by <code>multicols</code> . . . . .	186
4.1	Standard paper size options in $\text{\LaTeX}$ . . . . .	195
4.2	Default values for the page layout parameters ( <code>letterpaper</code> ) . . . . .	196
4.3	Page style defining commands in $\text{\LaTeX}$ . . . . .	223
5.1	The preamble options in the standard $\text{\LaTeX}$ <code>tabular</code> environment . .	243
5.2	Additional preamble options in the <code>array</code> package . . . . .	244
5.3	The preamble options in the <code>tabulary</code> package . . . . .	254
7.1	Standard size-changing commands . . . . .	342
7.2	Standard font-changing commands and declarations . . . . .	344

7.3	Font attribute defaults . . . . .	346
7.4	Predefined math alphabet identifiers in L <sup>A</sup> T <sub>E</sub> X . . . . .	349
7.5	Classification of the Computer Modern font families . . . . .	354
7.6	Commands made available with <code>textcomp</code> . . . . .	363
7.6	Commands made available with <code>textcomp</code> (cont.) . . . . .	364
7.7	Fonts used by PSNFSS packages . . . . .	371
7.8	Classification of font families in the PSNFSS distribution . . . . .	372
7.9	Glyphs in the PostScript font Zapf Dingbats . . . . .	379
7.10	Glyphs in the PostScript font Symbol . . . . .	382
7.11	Classification of the Concrete font families . . . . .	384
7.12	Classification of the Computer Modern Bright font families . . . . .	385
7.13	Classification of the LuxiMono font family . . . . .	387
7.14	Classification of the TX font families . . . . .	388
7.15	Classification of the PX font families . . . . .	391
7.16	Classification of the Fourier-GUTenberg font families . . . . .	392
7.17	Classification of the URW Antiqua and Grotesk fonts . . . . .	393
7.18	Classification of the Euler math font families . . . . .	397
7.19	Glyphs in the <code>wasy</code> fonts . . . . .	400
7.20	Glyphs in the MarVoSym font . . . . .	402
7.21	Glyphs in the METAFONT font bbdng . . . . .	404
7.22	TIPA shortcut characters . . . . .	406
7.23	Classification of the EuroSym font family . . . . .	409
7.24	Classification of the Adobe euro font families . . . . .	411
7.25	Weight and width classification of fonts . . . . .	414
7.26	Shape classification of fonts . . . . .	415
7.27	Standard font encodings used with L <sup>A</sup> T <sub>E</sub> X . . . . .	416
7.28	Karl Berry's font file name classification scheme . . . . .	420
7.29	Glyph chart for <code>msbm10</code> produced by the <code>nfssfont.tex</code> program . . . . .	434
7.30	Math symbol type classification . . . . .	435
7.31	LICR objects represented with single characters . . . . .	441
7.32	Glyph chart for a T1-encoded font ( <code>ecrm1000</code> ) . . . . .	449
7.33	Standard LICR objects . . . . .	455
8.1	Display environments in the <code>amsmath</code> package . . . . .	469
8.2	Default rule thickness in different math styles . . . . .	494
8.3	Vertically extensible symbols . . . . .	498
8.4	Predefined operators and functions . . . . .	500
8.5	Mathematical styles in sub-formulas . . . . .	502
8.6	Mathematical spacing commands . . . . .	508
8.7	Space between symbols . . . . .	525
8.8	Latin letters and Arabic numerals . . . . .	526
8.9	Symbols of class <code>\mathord</code> (Greek) . . . . .	527
8.10	Symbols of class <code>\mathord</code> (letter-shaped) . . . . .	527
8.11	Symbols of class <code>\mathord</code> (miscellaneous) . . . . .	528

8.12	Mathematical accents, giving sub-formulas of class <code>\mathord</code> . . . . .	529
8.13	Symbols of class <code>\mathbin</code> (miscellaneous) . . . . .	530
8.14	Symbols of class <code>\mathbin</code> (boxes) . . . . .	530
8.15	Symbols of class <code>\mathbin</code> (circles) . . . . .	531
8.16	Symbols of class <code>\mathrel</code> (equality and order) . . . . .	532
8.17	Symbols of class <code>\mathrel</code> (equality and order—negated) . . . . .	532
8.18	Symbols of class <code>\mathrel</code> (sets and inclusion) . . . . .	533
8.19	Symbols of class <code>\mathrel</code> (sets and inclusion—negated) . . . . .	533
8.20	Symbols of class <code>\mathrel</code> (arrows) . . . . .	534
8.21	Symbols of class <code>\mathrel</code> (arrows—negated) . . . . .	534
8.22	Symbols of class <code>\mathrel</code> (negation and arrow extensions) . . . . .	535
8.23	Symbols of class <code>\mathrel</code> (miscellaneous) . . . . .	535
8.24	Symbols of class <code>\mathpunct</code> , <code>\mathord</code> , <code>\mathinner</code> (punctuation) .	536
8.25	Symbols of class <code>\mathop</code> . . . . .	536
8.26	Symbol pairs of class <code>\mathopen</code> and <code>\mathclose</code> (extensible) . . . . .	537
8.27	Symbol pairs of class <code>\mathopen</code> and <code>\mathclose</code> (non-extensible) .	537
9.1	Language options supported by the <code>babel</code> system . . . . .	543
9.2	Language-dependent strings in <code>babel</code> (English defaults) . . . . .	547
9.3	Language-dependent strings in <code>babel</code> (French, Greek, Polish, Russian)	551
9.4	Different methods for representing numbers by letters . . . . .	560
9.5	Alternative mathematical operators for Eastern European languages .	564
9.6	Glyph chart for a T2A-encoded font ( <code>larm1000</code> ) . . . . .	572
9.7	Glyph chart for an LGR-encoded font ( <code>grmn1000</code> ) . . . . .	575
9.8	Greek transliteration with Latin letters for the LGR encoding . . . . .	576
9.9	LGR ligatures producing single-accented glyphs . . . . .	576
9.10	Available composite spiritus and accent combinations . . . . .	576
9.11	Glyph chart for an LHE-encoded font ( <code>shold10</code> ) . . . . .	578
9.12	Hebrew font-changing commands . . . . .	579
10.1	Overview of color and graphics capabilities of device drivers . . . . .	615
10.2	Arguments of <code>\DeclareGraphicsRule</code> . . . . .	626
10.3	Major options of the <code>dvips</code> program . . . . .	638
11.1	Input style parameters for <code>MakeIndex</code> . . . . .	660
11.2	Output style parameters for <code>MakeIndex</code> . . . . .	661
11.3	Languages supported by <code>texindy</code> . . . . .	670
11.4	<code>xindy</code> standard modules . . . . .	672
12.1	Gender specification in <code>jurabib</code> . . . . .	735
12.2	Comparison of packages for multiple bibliographies . . . . .	746
13.1	<code>BIBTEX</code> 's entry types as defined in most styles . . . . .	763
13.2	<code>BIBTEX</code> 's standard entry fields . . . . .	765

13.3	Predefined journal strings in <code>B<small>IB</small>T<small>EX</small></code> styles . . . . .	771
13.4	Selected <code>B<small>IB</small>T<small>EX</small></code> style files . . . . .	791
13.5	Requirements for formatting names . . . . .	798
13.6	Language support in <code>custom-bib</code> (summer 2003) . . . . .	800
13.7	<code>B<small>IB</small>T<small>EX</small></code> style file commands . . . . .	807
13.8	<code>B<small>IB</small>T<small>EX</small></code> style file built-in functions . . . . .	808
14.1	Overview of <code>doc</code> package commands . . . . .	820
A.1	<code>L<small>AT</small>E<small>X</small></code> 's units of length . . . . .	855
A.2	Predefined horizontal spaces . . . . .	856
A.3	Predefined vertical spaces . . . . .	857
A.4	Default values for <code>T<small>EX</small></code> 's rule primitives . . . . .	868
A.5	<code>L<small>AT</small>E<small>X</small></code> 's internal \boolean switches . . . . .	875
A.6	Commands for package and class files . . . . .	879

# Preface

A full decade has passed since the publication of the first edition of *The L<sup>A</sup>T<sub>E</sub>X Companion*—a decade during which some people prophesied the demise of T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X and predicted that other software would take over the world. There have been a great many changes indeed, but neither prediction has come to pass: T<sub>E</sub>X has not vanished and the interest in L<sup>A</sup>T<sub>E</sub>X has not declined, although the approach to both has gradually changed over time.

When we wrote the *Companion* in 1993 [55], we intended to describe what is usefully available in the L<sup>A</sup>T<sub>E</sub>X world (though ultimately we ended up describing what was available at CERN in those days). As an unintentional side effect, the first edition *defined* for most readers what should be available in a then-modern L<sup>A</sup>T<sub>E</sub>X distribution. Fortunately, most of the choices we made at that time proved to be reasonable, and the majority (albeit not all) of the packages described in the first edition are still in common use today. Thus, even though “the book shows its age, it still remains a solid reference in most parts”, as one reviewer put it recently.

Nevertheless, much has changed and a lot of new and exciting functionality has been added to L<sup>A</sup>T<sub>E</sub>X during the last decade. As a result, while revising the book we ended up rewriting 90% of the original content and adding about 600 additional pages describing impressive new developments.

What you are holding now is essentially a new book—a book that we hope preserves the positive aspects of the first edition even as it greatly enhances them, while at the same time avoiding the mistakes we made back then, both in content and presentation (though doubtless we made some others). For this book we used the CTAN archives as a basis and also went through the `comp.text.tex` news group archives to identify the most pressing questions and queries.

In addition to highlighting a good selection of the contributed packages available on the CTAN archives, the book describes many aspects of the basic L<sup>A</sup>T<sub>E</sub>X system that are not fully covered in the *L<sup>A</sup>T<sub>E</sub>X Manual*, Leslie Lamport's *L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System* [104]. Note, however, that our book is not a replacement for the *L<sup>A</sup>T<sub>E</sub>X Manual* but rather a companion to it: a reader of our book is assumed to have read at least the first part of that book (or a comparable introductory work, such as the *Guide to L<sup>A</sup>T<sub>E</sub>X* [101]) and to have some practical experience with producing L<sup>A</sup>T<sub>E</sub>X documents.

The second edition has seen a major change in the authorship; Frank took over as principal author (so he is to blame for all the faults in this book) and several members of the L<sup>A</sup>T<sub>E</sub>X3 project team joined in the book's preparation, enriching it with their knowledge and experience in individual subject areas.

*Thanks to a great  
guy!*

The preparation of the book was overshadowed by the sudden death of our good friend, colleague, and prospective co-author Michael Downes, whose great contributions to L<sup>A</sup>T<sub>E</sub>X, and *AMS-L<sup>A</sup>T<sub>E</sub>X* in particular, are well known to many people. We dedicate this book to him and his memory.

\* \* \*

We first of all wish to thank Peter Gordon, our editor at Addison-Wesley, who not only made this book possible, but through his constant encouragement also kept us on the right track (just a few years late). When we finally went into production, Elizabeth Ryan was unfailingly patient with our idiosyncrasies and steered us safely to completion.

We are especially indebted to Barbara Beeton, David Rhead, Lars Hellström, and Walter Schmidt for their careful reading of individual parts of the manuscript. Their numerous comments, suggestions, corrections, and hints have substantially improved the quality of the text.

Our very special thanks go to our contributing authors Christine Detig and Joachim Schrod for their invaluable help with Chapter 11 on index preparation.

*Haunted package  
authors*

Those who keep their ears to the ground for activities in the L<sup>A</sup>T<sub>E</sub>X world may have noticed an increased number of new releases of several well-established packages in 2002 and 2003. Some of these releases were triggered by our questions and comments to the package authors as we were preparing the manuscript for this second edition. Almost all package authors responded favorably to our requests for updates, changes, and clarifications, and all spent a considerable amount of time helping us with our task. We would particularly like to thank Jens Berger (*jurabib*), Axel Sommerfeldt (*caption*), Steven Cochran (*subfig*), Melchior Franz (*soul*, *euro*), and Carsten Heinz (*listings*) who had to deal with the bulk of the nearly 6000 e-mail messages that have been exchanged with various package authors.

Hearty thanks for similar reasons go to Alexander Rozhenko (*manyfoot*), Bernd Schndl (*paralist*), David Kastrup (*perpage*), Donald Arseneau (*cite*, *relsize*, *threeparttable*, *url*), Fabrice Popineau (*TeX Live CD*), Frank Bennett, Jr. (*camel*), Gerd Neugebauer (*bibtool*), Harald Harders (*subfloat*), Hideo Umeki

(geometry), Hubert Gäßlein (sidecap, pict2e), Javier Bezos (titlesec, titletoc), Jean-Pierre Drucbert (minitoc), Jeffrey Goldberg (endfloat, lastpage), John Lavagnino (endnotes), Markus Kohm (typearea), Martin Schröder (ragged2e), Matthias Eckermann (parallel), Michael Covington (upquote), Michel Bovani (fourier), Patrick Daly (custom-bib, natbib), Peter Heslin (ellipsis), Peter Wilson (layouts), Piet van Oostrum (extramarks, fancyhdr), Rei Fukui (tipa), Robin Fairbairns (footmisc), Rolf Niepraschk (sidecap, pict2e), Stephan Böttcher (lineno), Thomas Esser (teTeX distribution), Thomas Henlich (marvosym), Thorsten Hansen (bibunits, multibib), and Walter Schmidt (fix-cm, PSNFSS). Our apologies if we missed someone.

We gratefully recognize all of our many colleagues in the (L)A<sup>T</sup>E<sub>X</sub> world who developed the packages—not only those described here, but also the hundreds of others—that aim to help users meet the typesetting requirements for their documents. Without the continuous efforts of these enthusiasts, L<sup>A</sup>T<sub>E</sub>X would not be the magnificent and flexible tool it is today.

We would also like to thank Blenda Horn from Y&Y and Michael Vulis from MicroPress for supplying the fonts used to typeset the pages of this book.

The picture of Chris Rowley, taken after a good lunch at the Hong Kong International Airport, appears courtesy of Wai Wong. The picture of Michael Downes, taken at the T<sub>E</sub>X 2000 conference, Oxford, appears courtesy of Alan Wetmore.

\* \* \*

Any mistake found and reported is a gain for all readers of our book. We would therefore like to thank those readers who reported any of the mistakes which had been overlooked so far. The latest version of the errata file can be found on the L<sup>A</sup>T<sub>E</sub>X project site at <http://www.latex-project.org/guides/tlc2.err> where you will also find an on-line version of the index and other extracts from the book.

*To Err is Human*

\* \* \*

We would like to thank our families and friends for the support given during the preparation of this book—though this may sound like an alibi sentence to many, it never felt truer than with this book.

Chris would like to thank the Open University, United Kingdom, for supporting his work on L<sup>A</sup>T<sub>E</sub>X and the School of Computer Science and Engineering, University of New South Wales, for providing a most pleasant environment in which to complete his work on this book.

Frank Mittelbach  
Michel Goossens  
Johannes Braams  
David Carlisle  
Chris Rowley

*August 2004*



## C H A P T E R 1

# Introduction

$\text{\LaTeX}$  is not just a system for typesetting mathematics. Its applications span the one-page memorandum, business and personal letters, newsletters, articles, and books covering the whole range of the sciences and humanities, ... right up to full-scale expository texts and reference works on all topics. Nowadays, versions of  $\text{\LaTeX}$  exist for practically every type of computer and operating system. This book provides a wealth of information about its many present-day uses but first provides some background information.

The first section of this chapter looks back at the origins and subsequent development of  $\text{\LaTeX}$ .<sup>1</sup> The second section gives an overview of the file types used by a typical current  $\text{\LaTeX}$  system and the rôle played by each. Finally, the chapter offers some guidance on how to use the book.

### 1.1 A brief history

In May 1977, Donald Knuth of Stanford University [94] started work on the text-processing system that is now known as “ $\text{\TeX}$  and METAFONT” [82–86]. In the foreword of *The  $\text{\TeX}book$*  [82], Knuth writes: “ $\text{\TeX}$  [is] a new typesetting system intended for the creation of beautiful books—and especially for books that contain a lot of mathematics. By preparing a manuscript in  $\text{\TeX}$  format, you will be telling a computer exactly how the manuscript is to be transformed into pages whose typographic quality is comparable to that of the world’s finest printers.”

*In the Beginning ...*

<sup>1</sup>A more personal account can be found in *The  $\text{\LaTeX}$  legacy: 2.09 and all that* [148].

In 1979, Gordon Bell wrote in a foreword to an earlier book, *T<sub>E</sub>X and METAFONT, New Directions in Typesetting* [80]: “Don Knuth’s Tau Epsilon Chi (T<sub>E</sub>X) is potentially the most significant invention in typesetting in this century. It introduces a standard language in computer typography and in terms of importance could rank near the introduction of the Gutenberg press.”

In the early 1990s, Donald Knuth officially announced that T<sub>E</sub>X would not undergo any further development [96] in the interest of stability. Perhaps unsurprisingly, the 1990s saw a flowering of experimental projects that extended T<sub>E</sub>X in various directions; many of these are coming to fruition in the early 21st century, making it an exciting time to be involved in automated typography.

The development of T<sub>E</sub>X from its birth as one of Don’s “personal productivity tools” (created simply to ensure the rapid completion and typographic quality of his then-current work on *The Art of Computer Programming*) [88] was largely influenced and nourished by the American Mathematical Society on behalf of U.S. research mathematicians.

While Don was developing T<sub>E</sub>X, in the early 1980s, Leslie Lamport started work on the document preparation system now called L<sub>A</sub>T<sub>E</sub>X, which used T<sub>E</sub>X’s typesetting engine and macro system to implement a declarative document description language based on that of a system called Scribe by Brian Reid [142]. The appeal of such a system is that a few high-level L<sub>A</sub>T<sub>E</sub>X declarations, or commands, allow the user to easily compose a large range of documents without having to worry much about their typographical appearance. In principle at least, the details of the layout can be left for the document designer to specify elsewhere.

The second edition of *L<sub>A</sub>T<sub>E</sub>X: A Document Preparation System* [104] begins as follows: “L<sub>A</sub>T<sub>E</sub>X is a system for typesetting documents. Its first widely available version, mysteriously numbered 2.09, appeared in 1985.” This release of a stable and well-documented L<sub>A</sub>T<sub>E</sub>X led directly to the rapid spread of T<sub>E</sub>X-based document processing beyond the community of North American mathematicians.

L<sub>A</sub>T<sub>E</sub>X was the first widely used language for describing the logical structure of a large range of documents and hence introducing the philosophy of logical design, as used in Scribe. The central tenet of “logical design” is that the author should be concerned only with the logical content of his or her work and not its visual appearance. Back then, L<sub>A</sub>T<sub>E</sub>X was described variously as “T<sub>E</sub>X for the masses” and “Scribe liberated from inflexible formatting control”. Its use spread very rapidly during the next decade. By 1994 Leslie could write, “L<sub>A</sub>T<sub>E</sub>X is now extremely popular in the scientific and academic communities, and it is used extensively in industry.” But that level of ubiquity looks quite small when compared with the present day when it has become, for many professionals on every continent, a workhorse whose presence is as unremarkable and essential as the workstation on which it is used.

The worldwide availability of L<sub>A</sub>T<sub>E</sub>X quickly increased international interest in T<sub>E</sub>X and in its use for typesetting a range of languages. L<sub>A</sub>T<sub>E</sub>X 2.09 was (deliberately) not globalized but it was globalizable; moreover, it came with documentation worth translating because of its clear structure and straightforward style. Two

*... and Lamport  
saw that it was  
Good.*

*Going global*

pivotal conferences (Exeter UK, 1988, and Karlsruhe Germany, 1989) established clearly the widespread adoption of  $\text{\LaTeX}$  in Europe and led directly to International  $\text{\LaTeX}$  [151] and to work led by Johannes Braams [25] on more general support for using a wide variety of languages and switching between them (see Chapter 9).

Note that in the context of typography, the word *language* does not refer exclusively to the variety of natural languages and dialects across the universe; it also has a wider meaning. For typography, “language” covers a lot more than just the choice of “characters that make up words”, as many important distinctions derive from other cultural differences that affect traditions of written communication. Thus, important typographic differences are not necessarily in line with national groupings but rather arise from different types of documents and distinct publishing communities.

Another important contribution to the reach of  $\text{\LaTeX}$  was the pioneering work of Frank Mittelbach and Rainer Schöpf on a complete replacement for  $\text{\LaTeX}$ ’s interface to font resources, the New Font Selection Scheme (NFSS) (see Chapter 7). They were also heavily involved in the production of the  $\mathcal{AM}\text{-}\text{\LaTeX}$  system that added advanced mathematical typesetting capabilities to  $\text{\LaTeX}$  (see Chapter 8).

As a reward for all their efforts, which included a steady stream of bug reports (and fixes) for Leslie, by 1991 Frank and Rainer had “been allowed” to take over the technical support and maintenance of  $\text{\LaTeX}$ . One of their first acts was to consolidate International  $\text{\LaTeX}$  as part of the kernel<sup>1</sup> of the system, “according to the standard developed in Europe”. Very soon Version 2.09 was formally frozen and, although the change-log entries continue for a few months into 1992, plans for its demise as a supported system were already far advanced as something new was badly needed. The worldwide success of  $\text{\LaTeX}$  had by the early 1990s led in a sense to too much development activity: under the hood of Leslie’s “family sedan” many  $\text{\TeX}$ nicians had been laboring to add such goodies as super-charged, turbo-injection, multi-valved engines and much “look-no-thought” automation. Thus, the announcement in 1994 of the new standard  $\text{\LaTeX}$ , christened  $\text{\LaTeX} 2\epsilon$ , explains its existence in the following way:

*The Next Generation*

*Too much of a  
Good Thing™*

“Over the years many extensions have been developed for  $\text{\LaTeX}$ . This is, of course, a sure sign of its continuing popularity but it has had one unfortunate result: incompatible  $\text{\LaTeX}$  formats came into use at different sites. Thus, to process documents from various places, a site maintainer was forced to keep  $\text{\LaTeX}$  (with and without NFSS), SLI $\text{\TeX}$ ,  $\mathcal{AM}\text{-}\text{\LaTeX}$ , and so on. In addition, when looking at a source file it was not always clear for which format the document was written.

To put an end to this unsatisfactory situation a new release of  $\text{\LaTeX}$  was produced. It brings all such extensions back under a single format and thus prevents the proliferation of mutually incompatible dialects of  $\text{\LaTeX}$  2.09.”

<sup>1</sup> *Kernel* here means the core, or center, of the system.

The development of this “New Standard L<sup>A</sup>T<sub>E</sub>X” and its maintenance system *Standard L<sup>A</sup>T<sub>E</sub>X* was started in 1993 by the L<sup>A</sup>T<sub>E</sub>X3 Project Team [126], which soon comprised Frank Mittelbach, Rainer Schöpf, Chris Rowley, Johannes Braams, Michael Downes, David Carlisle, Alan Jeffrey, and Denys Duchier, with some encouragement and gentle bullying from Leslie. Although the major changes to the basic L<sup>A</sup>T<sub>E</sub>X system (the kernel) and the standard document classes (styles in 2.09) were completed by 1994, substantial extra support for colored typography, generic graphics, and fine positioning control were added later, largely by David Carlisle. Access to fonts for the new system incorporated work by Mark Purtill on extensions of NFSS to better support variable font encodings and scalable fonts [30–32].

*The 21st century* Although the original goal for this new version was consolidation of the wide range of models carrying the L<sup>A</sup>T<sub>E</sub>X marquee, what emerged was a substantially more powerful system with both a robust mechanism (via L<sup>A</sup>T<sub>E</sub>X packages) for extension and, importantly, a solid technical support and maintenance system. This provides robustness via standardization and maintainability of both the code base and the support systems. This system remains the current standard L<sup>A</sup>T<sub>E</sub>X system that is described in this book. It has fulfilled most of the goals for “a new L<sup>A</sup>T<sub>E</sub>X for the 21st Century”, as they were envisaged back in 1989 [129, 131].

The specific claims of the current system are “... better support for fonts, graphics and color; actively maintained by the L<sup>A</sup>T<sub>E</sub>X3 Project Team”. The details of how these goals were achieved, and the resulting subsystems that enabled the claims to be substantially attained, form a revealing study in distributed software support: The core work was done in at least five countries and, as is illustrated by the bugs database [106], the total number of active contributors to the technical support effort remains high.

*The package system* Although the L<sup>A</sup>T<sub>E</sub>X kernel suffered a little from feature creep in the late 1990s, the package system together with the clear development guidelines and the legal framework of the L<sup>A</sup>T<sub>E</sub>X Project Public License (LPPL) [111] have enabled L<sup>A</sup>T<sub>E</sub>X to remain almost completely stable while supporting a wide range of extensions. These have largely been provided by a similarly wide range of people who have, as the project team are happy to acknowledge and the on-line catalogue [169] bears witness, enhanced the available functionality in a vast panoply of areas.

*Development work* All major developments of the base system have been listed in the regular issues of *L<sup>A</sup>T<sub>E</sub>X News* [107]. At the turn of the century, development work by the L<sup>A</sup>T<sub>E</sub>X3 Project Team focused on the following areas: supporting multi-language documents [120]; a “Designer Interface for L<sup>A</sup>T<sub>E</sub>X” [123]; major enhancements to the output routine [121]; improved handling of inter-paragraph formatting; and the complex front-matter requirements of journal articles. Prototype code has been made available; see [124].

*No new features ...* One thing the project team steadfastly refused to do was to unnecessarily “enhance” the kernel by providing additional features as part of it, thereby avoiding the trap into which L<sup>A</sup>T<sub>E</sub>X 2.09 fell in the early 1990s: the disintegration into incompatible dialects where documents written at one site could not be successfully processed at another site. In this discussion it should not be forgotten that L<sup>A</sup>T<sub>E</sub>X

serves not only to produce high-quality documents, but also to enable collaboration and exchange by providing a lingua franca for various research communities.

With  $\text{\LaTeX} 2\epsilon$ , documents written in 1996<sup>1</sup> can still be run with today's  $\text{\LaTeX}$ . New documents run on older kernel releases if the additional packages used are brought up-to-date—a task that, in contrast to updating the  $\text{\LaTeX}$  kernel software, is easily manageable even for users working in a multiuser environment (e.g., in a university or company setting).

But a stable kernel is not identical to a standstill in software development; of equally crucial importance to the continuing relevance and popularity of  $\text{\LaTeX}$  is the diverse collection of contributed packages building on this stable base. The success of the package system for non-kernel extensions is demonstrated by the enthusiasm of these contributors—many thanks to all of them! As can be easily appreciated by visiting the highly accessible and stable Comprehensive  $\text{\TeX}$  Archive Network (see Appendix C) or by reading this book (where more than 250 of these “Good Guys”<sup>2</sup> are listed on page 1080), this has supported the existence of an enormous treasure trove of  $\text{\LaTeX}$  packages and related software.

*... but no standstill*

The provision of services, tools, and systems-level support for such a highly distributed maintenance and development system was itself a major intellectual challenge, because many standard working methods and software tools for these tasks assume that your colleagues are in the next room, not the next continent (and in the early days of the development, e-mail and FTP were the only reliable means of communication). The technical inventiveness and the personalities of everyone involved were both essential to creating this example of the friendly face of open software maintenance, but Alan Jeffrey and Rainer Schöpf deserve special mention for “fixing everything”.

*The back office*

A vital part of this system that is barely visible to most people is the regression testing system with its vast suite of test files [119]. It was devised and set up by Frank and Rainer with Daniel Flipo; it has proved its worth countless times in the never-ending battle of the bugs.

Some members of the project team have built on the team's experience to extend their individual research work in document science beyond the current  $\text{\LaTeX}$  structures and paradigms. Some examples of their work up to 2003 can be found in the following references: [33–36, 117, 127, 138, 147, 149].

*Research*

Meanwhile, the standard  $\text{\LaTeX}$  system will have two major advantages over anything else that will emerge in the next 10 years to support fully automated document processing. First, it will efficiently provide high-quality formatting of a large range of elements in very complex documents of arbitrary size. Second, it will be robust in both use and maintenance and hence will have the potential to remain in widespread use for at least a further 15 years.<sup>3</sup>

*Until 2020?*

<sup>1</sup>The time between 1994 and 1996 was a consolidation time for  $\text{\LaTeX} 2\epsilon$ , with major fixes and enhancements being made until the system was thoroughly stable.

<sup>2</sup>Unfortunately, this is nearly the literal truth: You need a keen eye to spot the nine ladies listed.

<sup>3</sup>One of the authors has publicly staked a modest amount of beer on  $\text{\TeX}$  remaining in general use (at least by mathematicians) until at least 2010.

*...and into the future* An important spin-off from the research work was the provision of some interfaces and extensions that are immediately usable with standard L<sup>A</sup>T<sub>E</sub>X. As more such functionality is added, it will become necessary to assess the likelihood that merely extending L<sup>A</sup>T<sub>E</sub>X in this way will provide a more powerful, yet still robust and maintainable, system. This is not the place to speculate further about the future of L<sup>A</sup>T<sub>E</sub>X but we can be sure that it will continue to develop and to expand its areas of influence whether in traditional publishing or in electronic systems for education and commerce.

## 1.2 Today's system

This section presents an overview of the vast array of files used by a typical L<sup>A</sup>T<sub>E</sub>X system with its many components. This overview will also involve some descriptions of how the various program components interact. Most users will never need to know anything of this software environment that supports their work, but this section will be a useful general reference and an aid to understanding some of the more technical parts of this book.

Although modern L<sup>A</sup>T<sub>E</sub>X systems are most often embedded in a project-oriented, menu-driven interface, behind the scenes little has changed from the file-based description given here. The stability of L<sup>A</sup>T<sub>E</sub>X over time also means that an article by Joachim Schrod on *The Components of T<sub>E</sub>X* [153] remains the best source for a more comprehensive explanation of a T<sub>E</sub>X-based typesetting system. The following description assumes familiarity with a standard computer file system in which a “file extension” is used to denote the “type of a file”.

In processing a document, the L<sup>A</sup>T<sub>E</sub>X program reads and writes several files, some of which are further processed by other applications. These are listed in Table 1.1, and Figure 1.1 shows schematically the flow of information behind the scenes (on pages 8 and 9).

The most obviously important files in any L<sup>A</sup>T<sub>E</sub>X-based documentation project are the *input source files*. Typically, there will be a master file that uses other subsidiary files (see Section 2.1). These files most often have the extension .tex (code documentation for L<sup>A</sup>T<sub>E</sub>X typically carries the extension .dtx; see Chapter 14); they are commonly known as “plain text files” since they can be prepared with a basic text editor. Often, external graphical images are included in the typeset document utilizing the graphics interface described in Section 10.2.

L<sup>A</sup>T<sub>E</sub>X also needs several files containing structure and layout definitions: *class* files with the extension .cls; *option* files with the extension .clo; *package* files with the extension .sty (see Appendix A). Many of these are provided by the basic system set-up, but others may be supplied by individual users. L<sup>A</sup>T<sub>E</sub>X is distributed with five standard document classes: article, report, book, slides, and letter. These document classes can be customized by the contents of other files specified either by class options or by loading additional packages as described in Section 2.1. In addition, many L<sup>A</sup>T<sub>E</sub>X documents will implicitly input *language definition files* of

the babel system with the extension `.ldf` (see Chapter 9) and *encoding definition files* of the `inputenc/fontenc` packages with the extension `.def` (see Chapter 7).

The information that  $\text{\LaTeX}$  needs about the glyphs to be typeset is found in  $\text{\TeX}$  *font metric* files (extension `.tfm`). This does not include information about the shapes of glyphs, only about their dimensions. Information about which font files are needed by  $\text{\LaTeX}$  is stored in *font definition* files (extension `.fd`). Both types are loaded automatically when necessary. See Chapter 7 for further information about font resources.

Font resources

A few other files need to be available to  $\text{\TeX}$ , but you are even less likely to come across them directly. An example includes the  $\text{\LaTeX}$  format file `latex.fmt` that contains the core  $\text{\LaTeX}$  instructions, precompiled for processing by the  $\text{\TeX}$  formatter. There are some situations in which this format needs to be recompiled—for example, when changing the set of hyphenation rules available to  $\text{\LaTeX}$  (configured in `language.dat`; see Section 9.5.1) and, of course, when a new  $\text{\LaTeX}$  kernel is made available. The details regarding how such formats are generated differ from one  $\text{\TeX}$  implementation to the next, so they are not described in this book.

The  $\text{\LaTeX}$  format

The output from  $\text{\LaTeX}$  itself is a collection of *internal* files (see below), plus one very important file that contains all the information produced by  $\text{\TeX}$  about the typeset form of the document.

$\text{\TeX}$ 's own particular representation of the formatted document is that of a *device-independent* file (extension `.dvi`).  $\text{\TeX}$  positions glyphs and rules with a precision far better than  $0.01\mu\text{m}$  ( $1/4,000,000$  inch). Therefore, the output generated by  $\text{\TeX}$  can be effectively considered to be independent of the abilities of any physical rendering device—hence the name. Some variants of the  $\text{\TeX}$  program, such as `pdfTeX` [159, 161] and `VTeX` [168], can produce device-independent file formats including the Portable Document Format (PDF) (extension `.pdf`), which is the native file format of Adobe Acrobat.

Formatted output

The `.dvi` file format specifies only the names/locations of fonts and their glyphs—it does not contain any rendering information for those glyphs. The `.pdf` file format can contain such rendering information.

Some of the *internal* files contain code needed to pass information from one  $\text{\LaTeX}$  run to the next, such as for cross-references (the `auxiliary` file, extension `.aux`; see Section 2.3) and for typesetting particular elements of the document such as the table of contents (extension `.toc`) and the lists of figures (extension `.lof`) and of tables (extension `.lot`). Others are specific to particular packages (such as `minitoc`, Section 2.3.6, or `endnotes`, Section 3.2.7) or to other parts of the system (see below).

Cross-references

Finally,  $\text{\TeX}$  generates a transcript file of its activities with the extension `.log`. This file contains a lot of information, such as the names of the files read, the numbers of the pages processed, warning and error messages, and other pertinent data that is especially useful when debugging errors (see Appendix B).

Errors, warnings, and information

A file with the extension `.idx` contains individual unsorted items to be indexed. These items need to be sorted, collated, and unified by a program like `makeindex` or `xindy` (see Chapter 11). The sorted version is typically placed into

Indexing

	<i>File Type</i>	<i>Common File Extension(s)</i>
<i>Document Input</i>	text	.tex .dtx .ltx
	bibliography	.bb1
	index / glossary	.ind / .gnd
<i>Graphics</i>	internal	.tex
	external	.ps .eps .tif .png .jpg .gif .pdf
<i>Other Input</i>	layout and structure	.clo .cls .sty
	encoding definitions	.def
	language definitions	.ldf
	font access definitions	.fd
	configuration data	.cfg
<i>Internal Communication (Input and Output)</i>	auxiliary	.aux
	table of contents	.toc
	list of figures / tables	.lof / .lot
<i>Low-level T<sub>E</sub>X Input</i>	format	.fmt
	font metrics	.tfm
<i>Output</i>	formatted result	.dvi .pdf
	transcript	.log
<i>Bibliography (B<sub>B</sub>T<sub>E</sub>X)</i>	input / output	.aux / .bb1
	database / style / transcript	.bib / .bst / .blg
<i>Index (MakeIndex)</i>	input / output	.idx / .ind
	style / transcript	.ist / .ilg

Table 1.1: Overview of the file types used by T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X

a file (extension .ind) that is itself input to L<sup>A</sup>T<sub>E</sub>X. For makeindex, the *index style information* file has an extension of .ist and its transcript file has an extension .ilg; in contrast xindy appears not to use any predefined file types.

Information about bibliographic citations (see Chapter 12) in a document is normally output by L<sup>A</sup>T<sub>E</sub>X to the *auxiliary* file. This information is used first to extract the necessary information from a bibliographic database and then to sort it; the sorted version is put into a *bibliography* file (extension .bb1) that is itself input to L<sup>A</sup>T<sub>E</sub>X. If the system uses B<sub>B</sub>T<sub>E</sub>X (see Chapter 13) for this task, then the *bibliographic database* files will have an extension of .bib, and information about the process will be in a *bibliography style* file (extension .bst). Its transcript file has the extension .blg.

Because of the limitations of T<sub>E</sub>X, especially its failure to handle graphics, it is often necessary to complete the formatting of some elements of the typeset document after T<sub>E</sub>X has positioned everything and written this information to the .dvi file. This is normally done by attaching extra information and handling instructions at the correct “geometrical position in the typeset document”, using

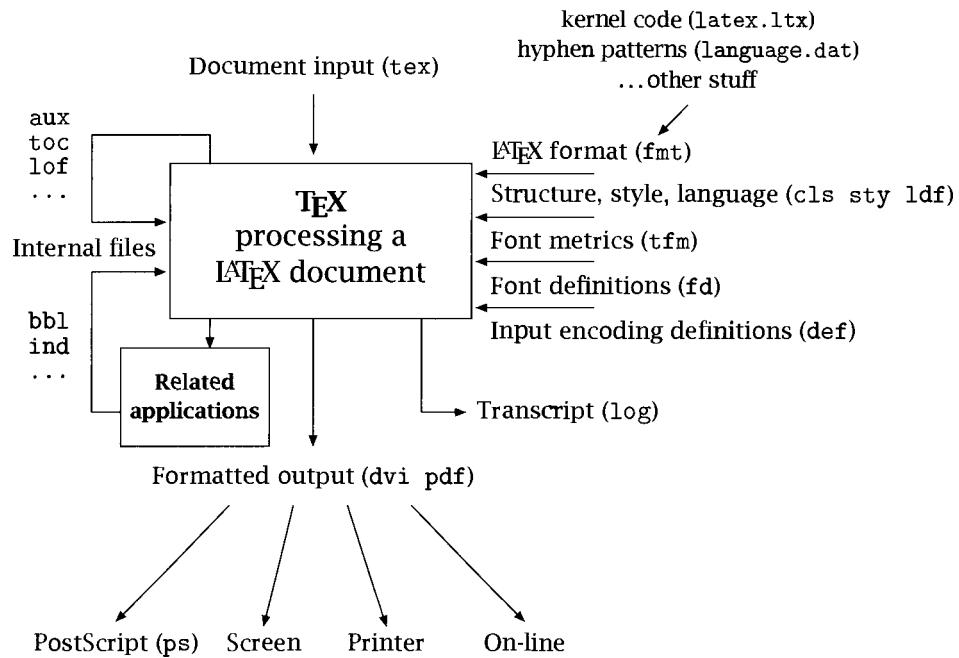


Figure 1.1: Data flow in the LATEX system

`\special`'s \special primitive that simply puts this information at the correct place in the .dvi file (see Chapter 10). This information may be simply the name of a graphics file to be input; or it may be instructions in a graphics language. Currently the most common such secondary formatter is a PostScript interpreter. To use this method, all information output by TeX to the .dvi file, including that in the \specials, must be transformed into PostScript code; applications to do this form part of all LATEX systems.

*PostScript*

Once the document has been successfully processed by TeX (and possibly transformed into PostScript), you will probably want to take a look at the formatted text. This is commonly done on screen, but detailed inspection of printed output should always be performed via printing on paper at the highest available resolution. The applications available for viewing documents on screen still (as of late 2003) vary quite a lot from system to system. Some require a .dvi file, while others use a .ps file. A current favorite approach is to use a .pdf file, especially when electronic distribution of the formatted document is required. Occasionally you will find that some applications will produce much better quality screen output than others; this is due to limitations of the different technologies and the availability of suitable font resources.

*Seeing is believing*

## 1.3 Working with this book

This final section of Chapter 1 gives an overview of the structure of this book, the typographic conventions used, and ways to use the examples given throughout the book.

### 1.3.1 What's here

Following is a summary of the subject areas covered by each chapter and appendix. In principle, the remaining chapters can be read independently since, when necessary, pointers are given to where necessary supplementary information can be found in other parts of the book.

- Chapter 1** gives a short introduction to the  $\text{\LaTeX}$  system and this book.
- Chapter 2** discusses document structure markup, including sectioning commands and cross-references.
- Chapter 3** describes  $\text{\LaTeX}$ 's basic typesetting commands.
- Chapter 4** explains how to influence the visual layout of the pages in various ways.
- Chapter 5** shows how to lay out material in columns and rows, on single and multiple pages.
- Chapter 6** discusses floating material and caption formatting.
- Chapter 7** discusses in detail  $\text{\LaTeX}$ 's Font Selection Scheme and shows how to access new fonts.
- Chapter 8** reviews mathematical typesetting, particularly the packages supported by the American Mathematical Society.
- Chapter 9** describes support for using  $\text{\LaTeX}$  with multiple languages, particularly the `babel` system.
- Chapter 10** covers the simpler extensions of  $\text{\LaTeX}$  for graphics, including the use of PostScript.
- Chapter 11** discusses the preparation and typesetting of an index; the programs `makeindex` and `xindy` are described.
- Chapter 12** describes  $\text{\LaTeX}$ 's support for the different bibliographical reference schemes in common use.
- Chapter 13** explains how to use bibliographical databases in conjunction with  $\text{\LaTeX}$  and how to generate typeset bibliographies according to publishers' expectations.

**Chapter 14** shows how to document  $\text{\LaTeX}$  files and how to use such files provided by others.

**Appendix A** reviews how to handle and manipulate the basic  $\text{\LaTeX}$  programming structures and how to produce class and package files.

**Appendix B** discusses how to trace and resolve problems.

**Appendix C** explains how to obtain the packages and systems described in this book and the support systems available.

**Appendix D** briefly introduces the TLC2  $\text{\TeX}$  CD-ROM (at the back of the book).

Some of the material covered in the book may be considered “low-level”  $\text{\TeX}$  that has no place in a book about  $\text{\LaTeX}$ . However, to the authors’ knowledge, much of this information has never been described in the “ $\text{\LaTeX}$ ” context though it is important. Moreover, we do not think that it would be helpful simply to direct readers to books like *The  $\text{\TeX}book$* , because most of the advice given in books about Plain  $\text{\TeX}$  is either not applicable to  $\text{\LaTeX}$  or, worse, produces subtle errors if used with  $\text{\LaTeX}$ . In some sections we have, therefore, tried to make the treatment as self-contained as possible by providing all the information about the underlying  $\text{\TeX}$  engine that is relevant and useful within the  $\text{\LaTeX}$  context.

### 1.3.2 Typographic conventions

It is essential that the presentation of the material conveys immediately its function in the framework of the text. Therefore, we present below the typographic conventions used in this book.

Throughout the text,  $\text{\LaTeX}$  command and environment names are set in monospaced type (e.g., `\caption`, `enumerate`, `\begin{tabular}`), while names of package and class files are in sans serif type (e.g., `article`). Commands to be typed by the user on a computer terminal are shown in monospaced type and are underlined (e.g., `This is user input`).

The syntax of the more complex  $\text{\LaTeX}$  commands is presented inside a rectangular box. Command arguments are shown in italic type:

```
\titlespacing*{cmd}{left-sep}{before-sep}{after-sep}[right-sep]
```

In  $\text{\LaTeX}$ , optional arguments are denoted with square brackets and the star indicates a variant form (i.e., is also optional), so the above box means that the `\titlespacing` command can come in four different incarnations:

```
\titlespacing{cmd}{left-sep}{before-sep}{after-sep}
\titlespacing{cmd}{left-sep}{before-sep}{after-sep}[right-sep]
\titlespacing*{cmd}{left-sep}{before-sep}{after-sep}
\titlespacing*{cmd}{left-sep}{before-sep}{after-sep}[right-sep]
```

For some commands, not all combinations of optional arguments and/or star forms are valid. In that case the valid alternatives are explicitly shown together, as, for example, in the case of L<sup>A</sup>T<sub>E</sub>X's sectioning commands:

```
\section*{title}      \section[toc-entry]{title}
```

Here the optional *toc-entry* argument can be present only in the unstarred form; thus, we get the following valid possibilities:

```
\section*{title}
\section{title}
\section[toc-entry]{title}
```

*Code examples ...*

Lines containing examples with L<sup>A</sup>T<sub>E</sub>X commands are indented and are typeset in a monospaced type at a size somewhat smaller than that of the main text:

```
\addtocontents{lof}{\protect\addvspace{10pt}}
\addtocontents{lot}{\protect\addvspace{10pt}}
```

*... with output ...*

However, in the majority of cases we provide complete examples together with the output they produce side by side:

The right column shows the input text `\usepackage{ragged2e}`  
to be treated by L<sup>A</sup>T<sub>E</sub>X with preamble material shown in blue. In the left column one sees the result after typesetting.

The right column shows the input text to be treated by `\LaTeX{}` with preamble material shown in blue. In the left column one sees the result after typesetting.

1-3-1

Note that all preamble commands are always shown in blue in the example source.  
*with several pages* In case several pages need to be shown to prove a particular point, (partial) “page spreads” are displayed and usually framed to indicate that we are showing material from several pages.

1 A TEST
<b>1 A test</b>
Some text for our page which might get reused over and over again.
Page 6 of 7

1 A TEST
Some text for our page which might get reused over and over again.
Page 7 of 7

```
\usepackage{fancyhdr, lastpage}
\pagestyle{fancy}
\fancyhf{} % --- clear all fields
\fancyhead[R,O,LE]{\leftmark}
\fancyfoot[C]{Page \thepage\ 
of \pageref{LastPage}}
% \sample defined as before
\section{A test}
\sample \par \sample
```

1-3-2

A number of points should be noted here:

- We usually arrange the examples to show pages 6 and 7 so that a double spread is displayed.

- We often use the command `\sample` to hold a short piece of text to keep the example code short (the definition for this command is either given as part of the example or, as indicated here, repeated from an earlier example—which in this example is simply a lie as `\sample` is not defined).
- The output may or may not show a header and footer. In the above case it shows both.

For large examples, where the input and output cannot be shown conveniently alongside each other, the following layout is used:

```
\usepackage{ragged2e}
This is a wide line, whose input commands and output result cannot
be shown nicely in two columns.
```

Depending on the example content, some additional explanation might appear between input and output (as in this case).

1-3-3 This is a wide line, whose input commands and output result cannot be shown nicely in two columns.

Chapter 8 shows yet another example format, where the margins of the example are explicitly indicated with thin blue vertical rules. This is done to better show the precise placement of displayed formulas and their tags in relation to the text margins.

*... or with lines  
indicating the  
margins*

1-3-4 (1) 
$$(a + b)^2 = a^2 + 2ab + b^2$$

```
\usepackage[leqno]{amsmath}
\begin{equation} (a+b)^2 = a^2+2ab+b^2 \end{equation}
```

All of these examples are “complete” if you mentally add a `\documentclass` line (with the `article` class<sup>1</sup> as an argument) and surround the body of the example with a `document` environment. In fact, this is how all of the (nearly 1000) examples in this book were produced. When processing the book, special `LATEX` commands take the source lines for an example and write them to an external file, thereby automatically adding the `\documentclass` and the `document` environment lines. This turns each example into a small but complete `LATEX` document. These documents are then externally processed (using a mechanism that runs each example as often as necessary, including the generation of a bibliography through `BIBTEX`). The result is converted into small EPS graphics, which are then loaded in the appropriate place the next time `LATEX` is run on the whole book. More details on the actual implementation of this scheme can be found in Section 3.4.3 on page 162.

Throughout the book, blue notes are sprinkled in the margin to help you easily find certain information that would otherwise be hard to locate. In a few cases these notes exhibit a warning sign, indicating that you should probably read this information even if you are otherwise only skimming through the particular section.

 Watch  
out for these

<sup>1</sup>Except for examples involving the `\chapter` command, which need the `report` or `book` class.

### 1.3.3 Using the examples

Our aim when producing this book was to make it as useful as possible for our readers. For this reason the book contains nearly 1000 complete, self-contained examples of all aspects of typesetting covered in the book.

These examples are made available in source format on CTAN in `info/examples/tlc2` and are also provided on the accompanying CD-ROM in `Books/tlc2/examples`. The examples are numbered per section, and each number is shown in a small box in the inner margin (e.g., 1-3-4 for the example on the preceding page). These numbers are also used for the external file names by appending `.ltx` (single-page examples) or `.ltx2` (double-page examples).

To reuse any of the examples it is usually sufficient to copy the preamble code (typeset in blue) into the preamble of your document and, if necessary, adjust the document text as shown. In some cases it might be more convenient to place the preamble code into your own package (or class file), thus allowing you to load this package in multiple documents using `\usepackage`. If you want to do the latter, there are two points to observe:

- Any use of `\usepackage` in the preamble code should be replaced by `\RequirePackage`, which is the equivalent command for use in package and class files (see Section A.4.5).
- Any occurrence of `\makeatletter` and `\makeatother` *must* be removed from the preamble code. This is very important because the `\makeatother` would stop correct reading of such a file.

So let us assume you wish to reuse the code from the following example:

## 1 Equations...

$$(a+b)^2 = a^2 + 2ab + b^2 \quad (1.1)$$

$$(a-b)^2 = a^2 - 2ab + b^2 \quad (1.2)$$

## 2 ...per section

$$(a+b)(a-b) = a^2 - b^2 \quad (2.1)$$

```
\makeatletter % '@' now normal "letter"
\@addtoreset{equation}{section}
\makeatother % '@' is restored as "non-letter"
\renewcommand\theequation{\oldstylenums{\thesection}%
.\oldstylenums{\arabic{equation}}}

\section{Equations\ldots}
\begin{equation} (a+b)^2 = a^2 + 2ab + b^2 \end{equation}
\begin{equation} (a-b)^2 = a^2 - 2ab + b^2 \end{equation}

\section{\ldots per section}
\begin{equation} (a+b)(a-b) = a^2 - b^2 \end{equation}
```

1-3-5

You have two alternatives: You can copy the preamble code (i.e., code colored blue) into your own document preamble or you can place that code—but without the `\makeatletter` and `\makeatother`—in a package file (e.g., `reseteqn.sty`) and afterwards load this “package” in the preamble of your own documents with `\usepackage{reseteqn}`.

## CHAPTER 2

# The Structure of a L<sup>A</sup>T<sub>E</sub>X Document

One of the ideas behind L<sup>A</sup>T<sub>E</sub>X is the separation between layout and structure (as far as possible), which allows the user to concentrate on content rather than having to worry about layout issues [104]. This chapter explains how this general principle is implemented in L<sup>A</sup>T<sub>E</sub>X.

The first section of this chapter shows how document class files, packages, options, and preamble commands can affect the structure and layout of a document. The logical subdivisions of a document are discussed in general, before explaining in more detail how sectioning commands and their arguments define a hierarchical structure, how they generate numbers for titles, and how they produce running heads and feet. Different ways of typesetting section titles are presented with the help of examples. It is also shown how the information that is written to the table of contents can be controlled and how the look of this table, as well as that of the lists of tables and figures, can be customized. The final section introduces L<sup>A</sup>T<sub>E</sub>X commands for managing cross-references and their scoping rules.

### 2.1 The structure of a source file

You can use L<sup>A</sup>T<sub>E</sub>X for several purposes, such as writing an article or a letter, or producing overhead slides. Clearly, documents for different purposes may need different logical structures, i.e., different commands and environments. We say that a document belongs to a *class* of documents having the same general structure (but not necessarily the same typographical appearance). You specify the class to which your document belongs by starting your L<sup>A</sup>T<sub>E</sub>X file with a \documentclass

command, where the mandatory parameter specifies the name of the *document class*. The document class defines the available logical commands and environments (for example, `\chapter` in the report class) as well as a default formatting for those elements. An optional argument allows you to modify the formatting of those elements by supplying a list of *class options*. For example, `11pt` is an option recognized by most document classes that instructs L<sup>A</sup>T<sub>E</sub>X to choose eleven point as the basic document type size.

Many L<sup>A</sup>T<sub>E</sub>X commands described in this book are not specific to a single class but can be used with several classes. A collection of such commands is called a package and you inform L<sup>A</sup>T<sub>E</sub>X about your use of certain packages in the document by placing one or more `\usepackage` commands after `\documentclass`.

Just like the `\documentclass` declaration, `\usepackage` has a mandatory argument consisting of the name of the package and an optional argument that can contain a list of *package options* that modify the behavior of the package.

The document classes and the packages reside in external files with the extensions `.cls` and `.sty`, respectively. Code for class options is sometimes stored in files (in this case with the extension `.clo`) but is normally directly specified in the class or package file (see Appendix A for information on declaring options in classes and packages). However, in case of options, the file name can differ from the option name. For example, the option `11pt` might be related to `art11.clo` when used in the article class and to `bk11.clo` inside the book class.

*The document preamble* Commands placed between `\documentclass` and `\begin{document}` are in the so-called *document preamble*. All style parameters must be defined in this preamble, either in package or class files or directly in the document *before* the `\begin{document}` command, which sets the values for some of the global parameters. A typical document preamble could look similar to the following:

```
\documentclass[twocolumn,a4paper]{article}
\usepackage{multicol}
\usepackage[german,french]{babel}
\addtolength{\textheight}{3\baselineskip}
\begin{document}
```

This document preamble defines that the class of the document is `article` and that the layout is influenced by the formatting request `twocolumn` (typeset in two columns) and the option `a4paper` (print on A4 paper). The first `\usepackage` declaration informs L<sup>A</sup>T<sub>E</sub>X that this document contains commands and structures provided by the package `multicol`. In addition, the `babel` package with the options `german` (support for German language) and `french` (support for French language) is loaded. Finally, the default height of the text body was enlarged by three lines for this document.

Generally, nonstandard L<sup>A</sup>T<sub>E</sub>X package files contain modifications, extensions, or improvements<sup>1</sup> with respect to standard L<sup>A</sup>T<sub>E</sub>X, while commands in the pream-

<sup>1</sup>Many of these packages have become de facto standards and are described in this book. This

ble define changes for the current document. Thus, to modify the layout of a document, you have several possibilities:

- Change the standard settings for parameters in a class file by options defined for that class.
- Add one or more packages to your document and make use of them.
- Change the standard settings for parameters in a package file by options defined for that package.
- Write your own local packages containing special parameter settings and load them with `\usepackage` after the package or class they are supposed to modify (as explained in the next section).
- Make final adjustments inside the preamble.

If you want to get deeper into L<sup>A</sup>T<sub>E</sub>X's internals, you can, of course, define your own general-purpose packages that can be manipulated with options. You will find additional information on this topic in Appendix A.

### 2.1.1 Processing of options and packages

Today's L<sup>A</sup>T<sub>E</sub>X makes a clear distinction between declared options (of a class or package) and general-purpose package files. The latter have to be specified using the `\usepackage` command. Think of options as properties of the whole document (when used in `\documentclass`) or as properties of individual packages (if specified in `\usepackage`).

You can specify options in a `\usepackage` command only if these options are declared by the package. Otherwise, you will receive an error message, informing you that your specified option is unknown to the package in question. Options to the `\documentclass` are handled slightly differently. If a specified option is not declared by the class, it will be assumed to be a "global option".

All options to `\documentclass` (both declared and global ones) are automatically passed as class options to all `\usepackage` declarations. Thus, if a package file loaded with a `\usepackage` declaration recognizes (i.e., declares) some of the class options, it can take appropriate actions. If not, the class options will be ignored while processing that package. Because all options have to be defined inside the class or package file, their actions are under the control of the class or package (an action can be anything from setting internal switches to reading an external file). For this reason their order in the optional argument of `\documentclass` or `\usepackage` is (usually) irrelevant.

does not mean, however, that packages that are not described here are necessarily less important or useful, of inferior quality, or should not be used. We merely concentrated on a few of the more established ones; for others, we chose to explain what functionality is possible in a given area.

If you want to use several packages, all taking the same options (for example, none), it is possible to load them all with a single \usepackage command by specifying the package names as a comma-separated list in the mandatory argument. For example,

```
\usepackage[german]{babel}    \usepackage[german]{varioref}
\usepackage{multicol}         \usepackage{epic}
```

is equivalent to

```
\usepackage[german]{babel,varioref} \usepackage{multicol,epic}
```

Specifying german as a global option to the class can further shorten the \usepackage declaration as german will be passed to all loaded packages and thus will be processed by those packages that declare it.

```
\documentclass[german]{book}
\usepackage{babel,varioref,multicol,epic}
```

Of course, this assumes that neither multicol nor epic changes its behavior when german is passed as a class option.

Finally, when the \begin{document} is reached, all global options are checked to see whether each has been used by at least one package; if not, a warning message is displayed. It is usually a spelling mistake if your option name is never used; another possibility is the removal of a \usepackage command loading a package that used this option previously.

If you want to make some modifications to a document class or a package (for example, changing parameter values or redefining some commands), you should put the relevant code into a separate file with the extension .sty. Then load this file with a \usepackage command after the package whose behavior you wish to modify (or the document class, if your modifications concern class issues).

Alternatively, you can insert the modifications directly into the preamble of your document. In that case, you may have to bracket them with \makeatletter and \makeatother if they contain internal L<sup>A</sup>T<sub>E</sub>X commands (i.e., those with an @ sign in their names). For more details see the discussion on page 843 concerning internal commands in the preamble.

### 2.1.2 Splitting the source file into parts

*Partial processing* L<sup>A</sup>T<sub>E</sub>X source documents can be conveniently split into several parts by using \include commands. Moreover, documents can be reformatted piecewise by specifying as arguments of an \includeonly command only those files L<sup>A</sup>T<sub>E</sub>X has to reprocess. For the other files that are specified in \include statements, the counter information (page, chapter, table, figure, equation, ...) will be read from the corresponding .aux files as long as they have been generated during a previous run.

In the following example, the user wants to reprocess only files `chap1.tex` and `appen1.tex`:

```
\documentclass{book}          % the document class "book"
\includeonly{chap1,appen1} % only include chap1 and appen1
\begin{document}
\include{chap1}            % input chap1.tex
\include{chap2}            % input chap2.tex
\include{chap3}            % input chap3.tex
\include{appn1}             % input appn1.tex
\include{appn2}             % input appn2.tex
\end{document}
```

Be aware that L<sup>A</sup>T<sub>E</sub>X only issues a warning message like "No file `xxx.tex`" when it cannot find a file specified in an `\include` statement, not an error message, and continues processing.

If the information in the `.aux` files is up-to-date, it is possible to process only part of a document and have all counters, cross-references, and pages be corrected in the reformatted part. However, if one of the counters (including the page number for cross-references) changes in the reprocessed part, then the complete document might have to be rerun to get the index, table of contents, and bibliographic references consistently correct.

Note that each document part loaded via `\include` starts on a new page and finishes by calling `\clearpage`; thus, floats contained therein will not move outside the pages produced by this part. So natural candidates for `\include` are whole chapters of a book but not necessarily small fractions of text.

While it is certainly an advantage to split a larger document into smaller parts and to work on more manageable files with a text editor, partial reformatting should be used only with great care and when still in the developing stage for one or more chapters. When a final and completely correct copy is needed, the only safe procedure is to reprocess the complete document. If a document is too large to process in a single run, it can be subdivided into parts that can be run separately. However, in this case, the pieces must be processed *in the correct sequence* (if necessary several times), to ensure that the cross-references and page numbers are correct.

If you intend to work with `\include` commands, consider using the small package `askinlude` created by Pablo Straub. It interactively asks you which files to include. You can then specify the files as a comma-separated list (i.e., what you would put into the `\includeonly` argument). If the Enter button is pressed in response, then the files from the previous run are included automatically (except on the first run, where this response means to include all files). If the answer is a `*`, then all files are included; a `-` means no files should be included. This way you do not have to modify your master source to process different parts of your document (a very useful feature during the production of this book).

*Excluding instead of including*

An extension to the `\include` mechanism is provided by the package `excludeonly` created by Dan Luecking and Donald Arseneau. It offers the command `\excludeonly`, which takes a comma-separated list of `\include` file names and prevents their inclusion. If both `\includeonly` and `\excludeonly` are used, then only the files permitted by both declarations are used. For example,

```
\includeonly{chap1, chap2, chap3, appen1}
\excludeonly{chap2, chap3, appen2}
```

results in only `chap1` and `appen1` being included. This behavior actually contradicts the package name, which indicates that “only” the given list is excluded. You can achieve this effect by calling the package with the option `only`, in which case an `\includeonly` declaration is ignored.

This package redefines the internal `\@include` command, so it will not work with packages or classes that redefine this command as well. Known conflicts are with the document classes `paper` and `thesis` by Wenzel Matiaske.

### 2.1.3 Combining several files

When sending a L<sup>A</sup>T<sub>E</sub>X document to another person you may have to send local or uncommon package files (e.g., your private modifications to some packages) along with the source. In such cases it is often helpful if you can put all the information required to process the document into a single file.

For this purpose, L<sup>A</sup>T<sub>E</sub>X provides the environment `filecontents`. This environment takes one argument, the name of a file;<sup>1</sup> its body consists of the contents of this file. It is only allowed to appear before a `\documentclass` declaration. The `\begin` and `\end` tags should be placed on lines of their own in the source. In particular, there should be no material following them, or you will get L<sup>A</sup>T<sub>E</sub>X errors.

If L<sup>A</sup>T<sub>E</sub>X encounters such an environment, it will try to find the mentioned file name. If it cannot, it will write the body of the environment verbatim into a file in the current directory and inform you about this action. Conversely, if a file with the given name was found by L<sup>A</sup>T<sub>E</sub>X, it will inform you that it has ignored this instance of the `filecontents` environment because the file is already present on the file system.

The generated file will get a few comment lines (using % as a comment character) added to the top to announce that this file was written by a `filecontents` environment:

```
%% LaTeX2e file 'foo.txt'
%% generated by the 'filecontents' environment
%% from source 'test' on 2003/04/16.
```

<sup>1</sup>If no extension is specified, the actual external file name will be the one L<sup>A</sup>T<sub>E</sub>X would read if you used this name as an argument to `\input`, i.e., typically adding the extension `.tex`.

If this is not appropriate—for example, if the file is not a  $\text{\LaTeX}$  file—use the `filecontents*` environment instead, which does not produce such extra lines.

To get a list of (nearly) all files used in your document (so that you know what you might have to pack together), specify the command `\listfiles` in the preamble.

#### 2.1.4 optional—Providing variants in the document source

Sometimes it is useful to keep several versions of a document together in a single source, especially if most of the text is shared between versions. This functionality is provided by the `optional` package created by Donald Arseneau.

The variant text parts are specially marked in the source using the command `\opt`, and during formatting some of them are selected. The command takes two arguments: a label (or a comma-separated list of labels) that describes to which variant the optional text belongs, and the text to be conditionally printed. Because the text is given in an argument, it cannot contain `\verb` commands and must have balanced braces. This approach works well enough for shorter texts. With longer parts to be optionally printed, however, it is usually best to store them in an external file and conditionally load this file using the `\opt` command, as was done in the example below.

There are a number of ways to select which variants are to be printed. The following example shows the non-interactive way, where the variants to be printed are specified by selecting them as options on the `\usepackage` declaration.

```
\usepackage[code]{optional}
\opt{doc}{Typeset this if option doc was declared.}
\opt{code}{Typeset this if option code was declared.}
\opt{doc,code}{Typeset this for either doc or code.}
Typeset this always. \opt{}{and this never!}
\opt{doc}{\input{examples}}
```

Typeset this if option code was declared. Typeset this for either doc or code. Typeset this always.

Alternatively, you can prompt the user each time for a list of options by including the declaration `\AskOptions` in the preamble, though that can become tedious if used too often. To help the person select the right options interactively you can define the command `\ExplainOptions`—if defined, its replacement text will be displayed on the terminal prior to asking for a list of options.

If your  $\text{\LaTeX}$  implementation supports passing  $\text{\LaTeX}$  code instead of a file name to the program, there is a third way to select the variants. If you invoke  $\text{\LaTeX}$  with the line

```
latex "\newcommand\UseOption{doc,code}\input{file}"
```

then the variants with the labels `doc` and `code` will be used (in addition to those specified on the `\usepackage`, if any). The example command line above would be suitable for a UN\*X system; on other platforms, you might need different quotes.

The optional package selects the variants to process during the L<sup>A</sup>T<sub>E</sub>X formatting. Depending on the application, it might be better to use a different approach involving a preprocessor that extracts individual variants from the master source. For example, the docstrip program can be successfully used for this purpose; in contrast to other preprocessors, it has the advantage that it will be usable at every site that has an installed L<sup>A</sup>T<sub>E</sub>X system (see Section 14.2 for details).

## 2.2 Sectioning commands

The standard L<sup>A</sup>T<sub>E</sub>X document classes (i.e., article, report, and book) contain commands and environments to define the different hierarchical structural units of a document (e.g., chapters, sections, appendices). Each such command defines a nesting level inside a hierarchy and each structural unit belongs to some level.

A typical document (such as an article) consists of a title, some sections with probably a multilevel nested substructure, and a list of references. To describe such a structure the title-generating command `\maketitle`, sectioning commands such as `\section` and `\subsection`, and the `thebibliography` environment are used. The commands should be correctly nested. For example, a `\subsection` command should be issued only after a previous `\section`.

Longer works (such as reports, manuals, and books) start with more complex title information, are subdivided into chapters (and parts), provide cross-reference information (table of contents, list of figures, list of tables, and indexes), and probably have appendices. In such a document you can easily distinguish the *front matter*, *body*, and *back matter*. In L<sup>A</sup>T<sub>E</sub>X's book class these three parts can be explicitly marked up using the commands `\frontmatter`, `\mainmatter`, and `\backmatter`. In other classes you often find only the command `\appendix`, which is used to separate the body matter from the back matter.

In the front matter the so-called *starred form* of the `\section` or `\chapter` sectioning command is normally used. This form suppresses the numbering of a heading. Sectional units with fixed names, such as "Introduction", "Index", and "Preface", are usually not numbered. In the standard classes, the commands `\tableofcontents`, `\listoftables`, and `\listoffigures`, and the `theindex` and `thebibliography` environments internally invoke the command (`\section` or `\chapter`) using their starred form.

Standard L<sup>A</sup>T<sub>E</sub>X provides the set of sectioning commands shown in Table 2.1. The `\chapter` command defines level zero of the hierarchical structure of a document, `\section` defines level one, and so on, whereas the optional `\part` command defines the level minus one (or zero in classes that do not define `\chapter`). Not all of these commands are defined in all document classes. The `article` class does not have `\chapter` and the `letter` class does not support sectioning commands at all. It is also possible for a package to define other sectioning commands, allowing either additional levels or variants for already supported levels.

\part (in book and report)	level -1	\part (in article)	level 0
\chapter (only book and report)	level 0	\section	level 1
\subsection	level 2	\subsubsection	level 3
\paragraph	level 4	\ subparagraph	level 5

Table 2.1: L<sup>A</sup>T<sub>E</sub>X's standard sectioning commands

Generally, the sectioning commands automatically perform one or more of the following typesetting actions:

- Produce the heading number reflecting the hierarchical level.
- Store the heading as an entry for a table of contents (into the `.toc` file).
- Save the contents of the heading to be (perhaps) used in a running head and/or foot.
- Format the heading.

All sectioning commands have a common syntax as exemplified here by the `\section` command:

```
\section*{title}      \section[toc-entry]{title}
```

The starred form (e.g., `\section*{...}`) suppresses the numbering for a title and does not produce an entry in the table of contents or the running head. In the second form the optional argument *toc-entry* is used when the text string for the table of contents and the running head and/or foot is different from the printed title. If this variant is used, numbering depends on the current value of the counter `secnumdepth` (discussed in the next section).

If you try to advise T<sub>E</sub>X on how to split the heading over a few lines using the “~” symbol or the `\backslash` command, then side effects may result when formatting the table of contents or generating the running head. In this case the simplest solution is to repeat the heading text without the specific markup in the optional parameter of the sectioning command.

*Problems with explicit formatting*

The remainder of this section discusses how the appearance of headings can be modified. It explains how to define a command like `\section` that has the above syntax, produces a table of contents entry if desired, but has a thick rule above its heading text or uses a normal-sized *italic* font rather than a large **bold** one.

First, some examples show how to change the numbering of headings. Next, examples demonstrate how to enter information about headings into the table of contents. Finally, changes to the general layout of headings are discussed, showing what L<sup>A</sup>T<sub>E</sub>X offers to define them.

### 2.2.1 Numbering headings

To support numbering, L<sup>A</sup>T<sub>E</sub>X uses a counter for each sectional unit and composes the heading number from these counters.

Perhaps the change desired most often concerning the numbering of titles is to alter the nesting level up to which a number should be produced. This is controlled by a counter named `secnumdepth`, which holds the highest level with numbered headings. For example, some documents have none of their headings numbered. Instead of always using the starred form of the sectioning commands, it is more convenient to set the counter `secnumdepth` to -2 in the document preamble. The advantages of this method are that an entry in the table of contents can still be produced, and that arguments from the sectioning commands can produce information in running headings. As discussed above, these features are suppressed in the starred form.

*Numbering no  
headings*

To number all headings down to `\subparagraph` or whatever the deepest sectioning level for the given class is called, setting the counter to a high enough value (e.g., a declaration such as `\setcounter{secnumdepth}{10}`) should normally be sufficient).

*Numbering all  
headings*

Finally, the `\addtocounter` command provides an easy way of numbering more or fewer heading levels without worrying about the level numbers of the corresponding sectioning commands. For example, if you need one more level with numbers, you can place `\addtocounter{secnumdepth}{1}` in the preamble of your document without having to look up the right value.

Every sectioning command has an associated counter, which by convention has the same name as the sectioning command (e.g., the command `\subsection` has a corresponding counter `subsection`). This counter holds the current (formatted) number for the given sectioning command. Thus, in the report class, the commands `\chapter`, `\section`, `\subsection`, and so on represent the hierarchical structure of the document and a counter like `subsection` keeps track of the number of `\subsections` used inside the current `\section`. Normally, when a counter at a given hierarchical level is stepped, then all lower-level counters (i.e., those with higher-level numbers) are reset. For example, the report class file contains the following declarations:

```
\newcounter{part} % (-1) parts
\newcounter{chapter} % (0) chapters
\newcounter{section}[chapter] % (1) sections
\newcounter{subsection}[section] % (2) subsections
\newcounter{subsubsection}[subsection] % (3) subsubsections
\newcounter{paragraph}[subsubsection] % (4) paragraphs
\newcounter{ subparagraph}[paragraph] % (5) subparagraphs
```

These commands declare the various counters. The level one (`section`) counter is reset when the level zero (`chapter`) counter is stepped. Similarly, the level two (`subsection`) counter is reset whenever the level one (`section`) counter is

stepped. The same mechanism is used down to the `\subparagraph` command. Note that in the standard classes the `part` counter is decoupled from the other counters and has no influence on the lower-level sectioning commands. As a consequence, `\chapters` in the `book` or `report` class or `\sections` in `article` will be numbered consecutively even if a `\part` command intervenes. Changing this is simple—you just replace the corresponding declaration of the chapter counter with:

```
\newcounter{chapter}[part]
```

The behavior of an already existing counter can be changed with the command `\@addtoreset` (see Appendix A.1.4), for example,

```
\@addtoreset{chapter}{part}
```

Recall that the latter instruction, because of the presence of the `@` character, can be issued only inside a package file or in the document preamble between `\makeatletter` and `\makeatother` commands, as explained on page 843.

Every counter in L<sup>A</sup>T<sub>E</sub>X, including the sectioning counters, has an associated command constructed by prefixing the counter name with `\the`, which generates a typeset representation of the counter in question. In case of the sectioning commands this representation form is used to produce the full number associated with the commands, as in the following definitions:

```
\renewcommand{\thechapter}{\arabic{chapter}}
\renewcommand{\thesection}{\thechapter.\arabic{section}}
\renewcommand{\thesubsection}{\thesection.\arabic{subsection}}
```

In this example, `\thesubsection` produces an Arabic number representation of the subsection counter prefixed by the command `\thesection` and a dot. This kind of recursive definition facilitates modifications to the counter representations because changes do not need to be made in more than one place. If, for example, you want to number sections using capital letters, you can redefine the command `\thesection`:

## A Different-looking section

### A.1 Different-looking subsection

Due to the default definitions not only the numbers on sections change, but lower-level sectioning commands also show this representation of the section number.

2-2-1

```
\renewcommand{\thesection}{\Alph{section}}
```

```
\section{Different-looking section}
\subsection{Different-looking subsection}
Due to the default definitions not only the numbers on sections change, but lower-level sectioning commands also show this representation of the section number.
```

Thus, by changing the counter representation commands, it is possible to change the number displayed by a sectioning command. However, the representa-

tion of the number cannot be changed arbitrarily by this method. Suppose you want to produce a subsection heading with the number surrounded by a box. Given the above examples one straightforward approach would be to redefine `\thesubsection`, e.g.,

```
\renewcommand\thesubsection{\fbox{\thesection.\arabic{subsection}}}
```

But this is not correct, as one sees when trying to reference such a section.

```
\renewcommand\thesubsection
  {\fbox{\thesection.\arabic{subsection}}}
\setcounter{section}{3}
\subsection{A mistake}\label{wrong}
Referencing a subsection in this format
produces a funny result as we can see
looking at subsection~\ref{wrong}.
We get a boxed reference.
```

2-2-2

### 3.1 A mistake

Referencing a subsection in this format produces a funny result as we can see looking at subsection 3.1. We get a boxed reference.

In other words, the counter representation commands are also used by L<sup>A</sup>T<sub>E</sub>X's cross-referencing mechanism (the `\label`, `\ref` commands; see Section 2.4). Therefore, we can only make small changes to the counter representation commands so that their use in the `\ref` command still makes sense. To produce the box around the heading number without spoiling the output of a `\ref`, we have to redefine L<sup>A</sup>T<sub>E</sub>X's internal command `\@secntformat`, which is responsible for typesetting the counter part of a section title. The default definition of `\@secntformat` typesets the `\the` representation of the section counter (in the example above, it uses the `\thesection` command), followed by a fixed horizontal space of 1em. Thus, to correct the problem, the previous example should be rewritten as follows:

```
\makeatletter
\renewcommand\@secntformat[1]{\fbox
  {\csname the#1\endcsname}\hspace{0.5em}}
\makeatother
\section{This is correct}\label{sec:OK}
Referencing a section using this
definition generates the correct result
for the section reference~\ref{sec:OK}.
```

2-2-3

### 1 This is correct

Referencing a section using this definition generates the correct result for the section reference 1.

The framed box around the number in the section heading is now defined only in the `\@secntformat` command, and hence the reference labels come out correctly.<sup>1</sup> Also note that we reduced the space between the box and the text to 0.5em

<sup>1</sup>The command `\@secntformat` takes as an argument the section level identifier, which is appended to the `\the` prefix to generate the presentation form needed via the `\csname`, `\endcsname` command constructor. In our example, the `\@secntformat` command is called with the `section` argument and thus the replacement text `\fbox{\csname thesection\endcsname}\hspace{0.5em}` is generated. See the *T<sub>E</sub>Xbook* [82] for more details about the `\csname` command.

(instead of the default 1em). The definition of `\@seccntformat` applies to all headings defined with the `\@startsection` command (which is described in the next section). Therefore, if you wish to use different definitions of `\@seccntformat` for different headings, you must put the appropriate code into every heading definition.

### 2.2.2 Formatting headings

`LATEX` provides a generic command called `\@startsection` that can be used to define a wide variety of heading layouts. To define or change a sectioning command one should find out whether `\@startsection` can do the job. If the desired layout is not achievable that way, then `\secdef` can be used to produce sectioning formats with arbitrary layout.

Headings can be loosely subdivided into two major groups: display and run-in headings. A display heading is separated by a vertical space from the preceding and the following text—most headings in this book are of this type.

A run-in heading is characterized by a vertical separation from the preceding text, but the text following the title continues on the same line as the heading itself, only separated from the latter by a horizontal space.

**Run-in headings.** This example shows how a run-in heading looks like. Text in the paragraph following the heading continues on the same line as the heading.

`\paragraph{Run-in headings.}`  
This example shows how a run-in heading looks like. Text in the paragraph following the heading continues on the same line as the heading.

The generic command `\@startsection` allows both types of headings to be defined. Its syntax and argument description are as follows:

`\@startsection{name}{level}{indent}{beforeskip}{afterskip}{style}`

*name* The name used to refer to the heading counter<sup>1</sup> for numbered headings and to define the command that generates a running header or footer (see page 218). For example, *name* would be the counter name, `\thename` would be the command to display the current heading number, and `\namemark` would be the command for running headers. In most circumstances the *name* will be identical to the name of the sectioning command being defined, without the preceding backslash—but this is no requirement.

*level* A number denoting the depth level of the sectioning command. This level is used to decide whether the sectioning command gets a number (if the level is less than or equal to `secnumdepth`; see Section 2.2.1 on page 24) or shows up in the table of contents (if the value is less or equal to `tocdepth`, see Section 2.3.2 on page 49). It should therefore reflect the position in the command

<sup>1</sup>This counter must exist; it is not defined automatically.

...end of last line of preceding text.

$\|bforeskip\| + \text{\parskip}$  (of text font) +  $\text{\baselineskip}$  (of heading font)

*indent*

## 3.5 Heading Title

$afterskip + \text{\parskip}$  (of heading font) +  $\text{\baselineskip}$  (of text font)

This is the start of the after-heading text, which continues on ...  
second line of text following the heading ...

—  
2-2-5

Figure 2.1: The layout for a display heading (produced by layouts)

hierarchy of sectioning commands, where the outermost sectioning command has level zero.<sup>1</sup>

*indent* The indentation of the heading with respect to the left margin. By making the value negative, the heading will start in the outer margin. Making it positive will indent all lines of the heading by this amount.

*bforeskip* The absolute value of this parameter defines the space to be left in front of the heading. If the parameter is negative, then the indentation of the paragraph following the heading is suppressed. This dimension is a rubber length, that is, it can take a stretch and shrink component. Note that L<sup>A</sup>T<sub>E</sub>X starts a new paragraph before the heading, so that additionally the value of  $\text{\parskip}$  is added to the space in front.

*afterskip* The space to be left following a heading. It is the vertical space after a display heading or the horizontal space after a run-in heading. The sign of *afterskip* controls whether a display heading ( $afterskip \geq 0$ ) or a run-in heading ( $afterskip < 0$ ) is produced. In the first case a new paragraph is started so that the value of  $\text{\parskip}$  is added to the space after the heading. An unpleasant side effect of this parameter coupling is that it is impossible to define a display heading with an effective “after space” of less than  $\text{\parskip}$  using the `\@startsection` command. When you try to compensate for a positive  $\text{\parskip}$  value by using a negative *afterskip*, you change the display heading into a run-in heading.

*style* The style of the heading text. This argument can take any instruction that influences the typesetting of text, such as `\raggedright`, `\Large`, or `\bfseries` (see the examples below).

<sup>1</sup>In the book and report classes, the `\part` command actually has level  $-1$  (see Table 2.1).

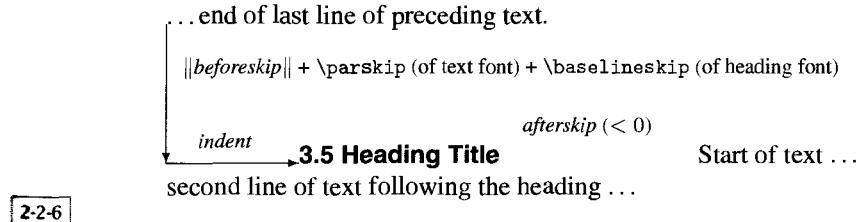


Figure 2.2: The layout for a run-in heading (produced by `layouts`)

Figures 2.1 and 2.2 show these parameters graphically for the case of display and run-in headings, respectively.

Next we show how these arguments are used in practice to define new sectioning commands. Suppose that you want to change the `\subsection` command of a class like `article` to look roughly like this:

```
% redefinition of \subsection shown below
\setcounter{section}{4} % simulate previous
% sections
\ldots some text above.
\subsection{Example of a Section Heading}
The heading is set in normal-sized italic
and the separation from the preceding text
is exactly one baseline. The separation
from the text following is one-half
baseline and this text is not indented.
```

In this case the following redefinition for `\subsection` is needed:

```
\makeatletter
\renewcommand{\subsection}{\@startsection
{subsection}{2}{0mm}{%
-\baselineskip}{%
0.5\baselineskip}{%
\normalfont\normalsize\itshape}}%
\makeatother
```

The first argument is the string `subsection` to denote that we use the corresponding counter for heading numbers. In the sectional hierarchy we are at level two. The third argument is `0mm` because the heading should start at the left margin. The absolute value of the fourth argument (`bforeskip`) specifies that a distance equal to one baseline must be left in front of the heading and, because the parameter is negative, that the indentation of the paragraph following the

heading should be suppressed. The absolute value of the fifth parameter (*afterskip*) specifies that a distance equal to one-half baseline must be left following the heading and, because the parameter is positive, that a display heading has to be produced. Finally, according to the sixth parameter, the heading should be typeset in an italic font using a size equal to the normal document type size.

In fact, the redefinition is a bit too simplistic because, as mentioned earlier, on top of the absolute value of *beforeskip* and *afterskip*, L<sup>A</sup>T<sub>E</sub>X always adds the current value of *\parskip*. Thus, in layouts where this parameter is nonzero, we need to subtract it to achieve the desired separation.

Another layout, which is sometimes found in fiction books, is given by the following definition:

```
\makeatletter
\renewcommand{\section}{\@startsection
{section}{1}{1em}%
{}% name, level, indent
{+\baselineskip}%
{}% beforeskip
{-\fontdimen2\font}%
{}% afterskip
    plus -\fontdimen3\font
    minus -\fontdimen4\font
}%
{\normalfont\normalsize\scshape}%
\makeatother
```

This defines a run-in heading using small capitals. The space definition for the horizontal *afterskip* deserves an explanation: it is the value of the stretchable space between words taken from the current font, negated to make a run-in heading. Details about *\fontdimens* can be found in Section 7.10.3 on page 426. The result is shown in the next example.

```
% redefinition of \section shown above
\setcounter{secnumdepth}{-2}
\ldots some text above.
\section{The man}
started to run away from the truck. He
saw that he was followed by
```

2-2-8

Of course, for such a layout one should turn off numbering of the headings by setting the counter *secnumdepth* to *-2*.

*Simple heading style  
changes*

Which commands can be used for setting the styles of the heading texts in the *style* argument of the *\@startsection* command? Apart from the font-changing directives (see Chapter 7), few instructions can be used here. A *\centering* command produces a centered display heading and a *\raggedright* declaration makes the text left justified. The use of *\raggedleft* is possible, but may give somewhat strange results. You can also use *\hrule*, *\medskip*, *\newline*, or

similar commands that introduce local changes. The next example shows some possible variations.

## 1 A very long heading that shows the default behavior of L<sup>A</sup>T<sub>E</sub>X's sectioning commands

### 1.1 A subsection heading

The heading is centered using an italic font.

### 1.2 A subsection heading

The heading is left-justified using a sans serif font.

### 1.3 A SUBSECTION HEADING

The heading is right-justified and uses uppercase letters.

### 1.4 A subsection heading

This heading has a horizontal rule above the text.

In the standard L<sup>A</sup>T<sub>E</sub>X document classes, words in long headings are justified and, if necessary, hyphenated as can be seen in the previous example. If this is not wanted, then justification can be turned off by using `\raggedright` in the *style* part of the `\@startsection` command. If line breaks are manually adjusted using `\backslash`, then one has to repeat the heading title, without the extra formatting instruction, in the optional argument. Otherwise, the line breaks will also show up in the table of contents.

*Hyphenation  
and line breaks  
in headings*

## 1 A very long heading that shows the default behavior of L<sup>A</sup>T<sub>E</sub>X's sectioning commands

2-2-10

```
\makeatletter
\newcommand{\Csub}{\@startsection{subsection}{2}%
{0pt}{-\baselineskip}{.2\baselineskip}{%
\centering\itshape}}
\newcommand{\Lsub}{\@startsection{subsection}{2}%
{0pt}{-\baselineskip}{.2\baselineskip}{%
\raggedright\sffamily}}
\newcommand{\Rsub}{\@startsection{subsection}{2}%
{0pt}{-\baselineskip}{.2\baselineskip}{%
\raggedleft\MakeUppercase}}
\newcommand{\Hsub}{\@startsection{subsection}{2}%
{0pt}{-\baselineskip}{.2\baselineskip}{%
\hrule\medskip\itshape}}
\makeatother
\section{A very long heading that shows
the default behavior of \LaTeX's
sectioning commands}
\Csub{A subsection heading}
The heading is centered using an italic font.
\Lsub{A subsection heading}
The heading is left-justified using a sans
serif font.
\Rsub{A subsection heading}
The heading is right-justified and uses
uppercase letters.
\Hsub{A subsection heading}
This heading has a horizontal rule above
the text.
```

```
\makeatletter
\renewcommand{\section}{\@startsection{section}{1}%
{0pt}{-\baselineskip}{.2\baselineskip}{%
\normalfont\Large\bfseries\raggedright}}
\makeatother
\section{A very long heading that shows
the default behavior of \LaTeX's
sectioning commands}
```

*Indentation after  
a heading*

Finally, a few words about the suppression of the indentation for the first paragraph after a display heading. Standard L<sup>A</sup>T<sub>E</sub>X document classes, following (American) English typographic tradition, suppress the indentation in this case. All first paragraphs after a display heading can be indented by specifying the package `indentfirst` (David Carlisle).

*Complex heading  
layout definitions*

In the standard L<sup>A</sup>T<sub>E</sub>X classes the highest-level sectioning commands `\part` and `\chapter` produce their titles without using `\@startsection` since their layout cannot be produced with that command. Similarly, you may also want to construct sectioning commands without limitations. In this case you must follow a few conventions to allow L<sup>A</sup>T<sub>E</sub>X to take all the necessary typesetting actions when executing them.

The command `\secdef` can help you when defining such commands by providing an easy interface to the three possible forms of section headings, as shown in the case of the `\part` command. With the definition

```
\newcommand{\part}{\secdef\cmda\cmdb}
```

the following actions take place:

<code>\part{title}</code>	will invoke	<code>\cmda[title]{title}</code>
<code>\part[toc-entry]{title}</code>	will invoke	<code>\cmda[toc-entry]{title}</code>
<code>\part*[title]</code>	will invoke	<code>\cmdb{title}</code>

The commands you have to provide are a (re)definition<sup>1</sup> of `\part` and a definition of the commands labeled `\cmda` or `\cmdb`, respectively. Note that `\cmda` has an optional argument containing the text to be entered in the table of contents `.toc` file, while the second (mandatory) argument, as well as the single argument to `\cmdb`, specifies the heading text to be typeset. Thus, the definitions must have the following structure:

```
\newcommand{\part}{... \secdef \cmda \cmdb }
\newcommand{\cmda}[2][default]{...}
\newcommand{\cmdb}[1]{...}
```

An explicit example is a simplified variant of `\appendix`. It redefines the `\section` command to produce headings for appendices (by invoking either the command `\Appendix` or `\sAppendix`), changing the presentation of the `section` counter and resetting it to zero. The modified `\section` command also starts a new page, which is typeset with a special page style (see Chapter 4) and with top floats suppressed. The indentation of the first paragraph in a section is also suppressed by using the low-level kernel command `\@afterheading` and setting the Boolean switch `@afterindent` to `false`. For details on the use of these commands see the `\chapter` implementation in the standard classes (file `classes.dtx`).

<sup>1</sup>Redefinition in case you change an existing heading command such as `\part` in the preamble of your document.

```
\makeatletter
\renewcommand\appendix{%
    \renewcommand\section{%
        \newpage\thispagestyle{plain}%
        \suppressfloats[t]\@afterindentfalse%
        \secdef\Appendix\sAppendix}%
    \setcounter{section}{0}\renewcommand\thesection{\Alph{section}}}}
```

In the definition below you can see how `\Appendix` advances the section counter using the `\refstepcounter` command (the latter also resets all subsidiary counters and defines the “current reference string”; see Section 2.4). It writes a line into the `.toc` file with the `\addcontentsline` command, performs the formatting of the heading title, and saves the title for running heads and/or feet by calling `\sectionmark`. The `\@afterheading` command handles the indentation of the paragraph following the heading.

```
\newcommand\Appendix[2][?]{%
    \refstepcounter{section}%                                % Complex form:
    \addcontentsline{toc}{appendix}{%                      % step counter/ set label
        \protect\numberline{\appendixname~\thesection\#1}%
        {\raggedleft\large\bfseries \appendixname\%
            \thesection\par\centering\#2\par}%
        \sectionmark{\#1}%                                 % generate toc entry
        \@afterheading%                                % add to running header
        \addvspace{\baselineskip}}%                         % prepare indentation handling
                                                % space after heading}
```

The `\sAppendix` command (starred form) performs only the formatting.

```
\newcommand\sAppendix[1]{%
    {\raggedleft\large\bfseries\appendixname\par\centering\#1\par}%
    \@afterheading\addvspace{\baselineskip}}
\makeatother
```

Applying these definitions will produce the following output:

## Appendix A The list of all commands

```
% Example needs commands introduced above!
\appendix
\section{The list of all commands}
```

Then follows the text of the first section in the appendix. Some more text in the appendix. Some more text in the appendix.

Then follows the text of the first section in the appendix. Some more text in the appendix. Some more text in the appendix.

2-2-11

Do not forget that the example shown above represents only a simplified version of a redefined `\section` command. Among other things, we did not take into account the `secnumdepth` counter, which contains the numbering threshold. You might also have to foresee code dealing with various types of document formats, such as one- and two-column output, or one- and two-sided printing.

<i>Command</i>	<i>Default</i>
\abstractname	Abstract
\appendixname	Appendix
\bibname	Bibliography
\chaptername	Chapter
\contentsname	Contents
\indexname	Index
\listfigurename	List of Figures
\listtablename	List of Tables
\partname	Part
\refname	References

Table 2.2: Language-dependent strings for headings

### 2.2.3 Changing fixed heading texts

Some of the standard heading commands produce predefined texts. For example, \chapter produces the string “Chapter” in front of the user-supplied text. Similarly, some environments generate headings with predefined texts. For example, by default the abstract environment displays the word “Abstract” above the text of the abstract supplied by the user. L<sup>A</sup>T<sub>E</sub>X defines these strings as command sequences (see Table 2.2) so that you can easily customize them to obtain your favorite names. This is shown in the example below, where the default name “Abstract”, as defined in the article class, is replaced by the word “Summary”.

<b>Summary</b> This book describes how to modify the appearance of documents produced with the L <sup>A</sup> T <sub>E</sub> X typesetting system.	<pre>\renewcommand\abstractname{Summary} \begin{abstract}   This book describes how to modify the   appearance of documents produced with   the L<sup>A</sup>T<sub>E</sub>X{} typesetting system. \end{abstract}</pre>
---	--

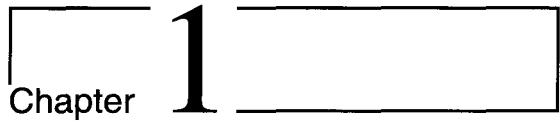
2-2-12

The standard L<sup>A</sup>T<sub>E</sub>X class files define a few more strings. See Section 9.1.3, and especially Table 9.2 on page 547, for a full list and a discussion of the babel system, which provides translations of these strings in more than twenty languages.

### 2.2.4 fncychap—Predefined chapter heading layouts

For those who wish to have fancy chapter headings without much work there exists the package fncychap (Ulf Lindgren). It provides six distinctive layout styles for the \chapter command that can be activated by loading the package with one of the following options: Sonny, Lenny, Glenn, Conny, Rejne, or Bjarne. Because the package is intended for modifying the \chapter command, it works only with

document classes that provide this command (e.g., `report` and `book`, but not `article` and its derivatives). As an example we show the results of using the option `Lenny`.



## 2-2-13 A Package Test

```
\usepackage[Lenny]{fncychap}
\chapter{A Package Test}
```

The package also offers several commands to modify the layouts in various ways. It comes with a short manual that explains how to provide your own layouts.

### 2.2.5 `quotchap`—Mottos on chapters

Another way to enhance `\chapter` headings is provided by the `quotchap` package created by Karsten Tinnefeld. It allows the user to specify quotation(s) that will appear on the top left of the chapter title area.

The quotation(s) for the next chapter are specified in a `savequote` environment; the width of the quotation area can be given as an optional argument defaulting to `10cm`. Each quotation should finish with a `\qauthor` command to denote its source, though it would be possible to provide your own formatting manually.

The default layout produced by the package can be described as follows: the quotations are placed flush left, followed by vertical material stored in the command `\chapterheadstartvskip`. It is followed by a very large chapter number, typeset flush right in 60% gray, followed by the chapter title text, also typeset flush right. After a further vertical separation, taken from the command `\chapterheadendvskip`, the first paragraph of the chapter is started without indentation.

The number can be printed in black by specifying the option `nogrey` to the package. To print the chapter number in one of the freely available PostScript fonts, you can choose among a number of options, such as `charter` for Bitstream's Charter BT or `times` for Adobe's Times. By default, Adobe's Bookman is chosen. Alternatively, you could redefine the `\chapnumfont` command, which is responsible for selecting the font for the chapter number. Finally, the font for the chapter title can be influenced by redefining the `\sectfont` command as shown in the example.

This, together with the possibilities offered by redefining the commands `\chapterheadstartvskip` and `\chapterheadendvskip`, allows you to produce a number of interesting layouts. The example below uses a negative vertical skip

to move the quotation on the same level as the number (in Avantgarde) and set the title and quotation in Helvetica.

*Cookies! Give me some cookies!*  
Cookie Monster



## A Package Test

```
\usepackage[avantgarde]{quotchap}
\renewcommand\chapterheadstartvskip
{\vspace*{-5\baselineskip}}
% select Helvetica for title and quote
\usepackage{helvet}
\renewcommand\sectfont{\sffamily\bfseries}
\begin{savequote}[10pc]
\sffamily
Cookies! Give me some cookies!
\qauthor{Cookie Monster}
\end{savequote}
\chapter{A Package Test}
```

Adding this package changes the chapter heading dramatically.

Adding this package changes the chapter heading dramatically.

2-2-14

If you want quotations on your chapters but prefer one of the layouts provided by `fncychap`, you can try to combine both packages. Load the latter package *after* `quotchap`. Of course, the customization possibilities described above are then no longer available but `savequote` will still work, although the quotations will appear always in a fixed position above the heading.

### 2.2.6 `titlesec`—A different approach to headings

The information presented so far in this chapter has focused on the tools and mechanisms provided by the L<sup>A</sup>T<sub>E</sub>X kernel for defining and manipulating headings, as well as a few packages that provide some extra features, such as predefined layouts, built on top of the standard tools.

The `titlesec` package created by Javier Bezos approaches the topic differently by providing a complete reimplementation for the heading commands. Javier's approach overcomes some of the limitations inherent in the original tools and provides a cleaner and more generic interface. The disadvantage is that this package might introduce some incompatibilities with extensions based on the original interfaces. Whether this possibility turns out to be a real issue clearly depends on the task at hand and is likely to vanish more or less completely the moment this interface comes into more widespread use.

The package supports two interfaces: a simple one for smaller adjustments, which is realized mainly by options to the package, and an extended interface to make more elaborate modifications.

### The basic interface

The basic interface lets you modify the font characteristics of all headings by specifying one or more options setting a font family (`rm`, `sf`, `tt`), a font series (`md`, `bf`), or a font shape (`up`, `it`, `s1`, `sc`). The title size can be influenced by selecting one of the following options: `big` (same sizes as for standard L<sup>A</sup>T<sub>E</sub>X classes), `tiny` (all headings except for chapters in text size), or `medium` or `small`, which are layouts between the two extremes. The alignment is controlled by `raggedleft`, `center`, or `raggedright`, while the vertical spacing can be reduced by specifying the option `compact`.

To modify the format of the number accompanying a heading, the command `\titlelabel` is available. Within it `\thetitle` refers to the current sectioning number, such as `\thesection` or `\thesubsection`. The declaration applies to all headings, as can be seen in the next example.

#### 1. A section

```
\usepackage[sf,bf,tiny,center]{titlesec}
\titlelabel{\thetitle.\enspace}
\section{A section}
```

#### 1.1. A subsection

```
\subsection{A subsection}
```

#### 1.1.1. A subsubsection

```
\subsubsection{A subsubsection}
```

Three headings following each other, a situation you will not see often ...

Three headings following each other, a situation you will not see often \ldots

2-2-15

`\titleformat*{cmd}{format}`

The basic interface offers one more command, `\titleformat*`, that takes two arguments. The first argument (*cmd*) is a sectioning command we intend to modify. The second argument (*format*) contains the formatting instruction that should be applied to this particular heading. This declaration works on individual sectioning commands, and its use overwrites all font or alignment specifications given as options to the package (i.e., the options `rm`, `it`, and `raggedleft` in the following example). The last command used in the second argument can be a command with one argument—it will receive the title text if present. In the next example we use this feature to set the `\subsubsection` title in lowercase (though this looks rather ugly with full-sized numbers).

I A section      \usepackage[rm,it,raggedleft,tiny,compact]{titlesec}

I.I A subsection      \titleformat\*{\subsubsection}{\scshape\MakeLowercase}

#### 1.1.1 A SUBSUBSECTION

```
\section{A section}
```

```
\subsection{A subsection}
```

```
\subsubsection{A subsubsection}
```

Three headings following each other, a situation you will not see often ...

Three headings following each other, a situation you will not see often \ldots

2-2-16

The `\part` heading is not influenced by settings for the basic interface. If you want to modify it, you must use the extended interface described below.

### The extended interface

The extended interface consists of two major commands, `\titleformat` and `\titlespacing`. They allow you to declare the “inner” format (i.e., fonts, label, alignment, ...) and the “outer” format (i.e., spacing, indentation, etc.), respectively. This scheme was adopted because people often wish to alter only one or the other aspect of the layout.

```
\titleformat{cmd}[shape]{format}{label}{sep}{before-code}[after-code]
```

The first argument (*cmd*) is the heading command name (e.g., `\section`) whose format is to be modified. In contrast to `\@startsection` this argument requires the command name—that is, with the backslash in front. The remaining arguments have the following meaning:

*shape* The basic shape for the heading. A number of predefined shapes are available: `hang`, the default, produces a hanging label (like `\section` in standard classes); `display` puts label and heading text on separate lines (like standard `\chapter`), while `runin` produces a run-in title (like standard `\paragraph`).

In addition, the following shapes, which have no equivalents in standard L<sup>A</sup>T<sub>E</sub>X, are provided: `frame` is similar to `display` but frames the title; `leftmargin` puts the title into the left margin; while `rightmargin` places it into the right margin. The last two shapes might conflict with `\marginpar` commands, that is, they may overlap.

A general-purpose shape is `block`, which typesets the heading as a single block. It should be preferred to `hang` for centered layouts.

Both `drop` and `wrap` wrap the first paragraph around the title, with `drop` using a fixed width for the title and `wrap` using the width of the widest title line (automatically breaking the title within the limit forced by the *left-sep* argument of `\titlespacing`).

As the interface is extensible (for programmers), additional shapes may be available with your installation.

*format* The declarations that are applied to the whole title—label and text. They may include only vertical material, which is typeset following the space above the heading. If you need horizontal material, it should be entered in the *label* or *before-code* argument.

*label* The formatting of the label, that is, the heading number. To refer to the number itself, use `\thesection` or whatever is appropriate. For defining `\chapter` headings the package offers `\chapertitle`, which produces `\chaptername` or `\appendixname`, depending on the position of the heading in the document.

*sep* Length whose value determines the distance between the label and title text. Depending on the *shape* argument, it might be a vertical or horizontal separa-

tion. For example, with the `frame` shape, it specifies the distance between the frame and heading text.

*before-code* Code executed immediately preceding the heading text. Its last command can take one argument, which will pick up the heading text and thus permits more complicated manipulations (see Example 2-2-19).

*after-code* Optional code to be executed after formatting the heading text (still within the scope of the declarations given in *format*). For `hang`, `block`, and `display`, it is executed in vertical mode; with `runin`, it is executed in horizontal mode. For other shapes, it has no effect.

If the starred form of a heading is used, the *label* and *sep* arguments are ignored because no number is produced.

The next example shows a more old-fashioned run-in heading, for which we define only the *format*, not the spacing around the heading. The latter is manipulated with the `\titlespacing` command.

```
\usepackage{titlesec}
\titleformat{\section}[runin]{\normalfont\scshape}
  {\S\,,\oldstylenums{\thesection}.}{.5em}{}[.\quad]
§ 1. THE TITLE. The heading is separated from the section text by a dot and a space of one quad. \section{The Title} The heading is separated from the section text by a dot and a space of one quad.
```

By default, L<sup>A</sup>T<sub>E</sub>X's `\section` headings are not indented (they are usually of *shape hang*). If you prefer a normal paragraph indentation with such a heading, you could add `\indent` before the `\S` sign or specify the indentation with the `\titlespacing` declaration, described next.

`\titlespacing*[cmd]{left-sep}{before-sep}{after-sep}[right-sep]`

The starred form of the command suppresses the paragraph indentation for the paragraph following the title, except with shapes where the heading and paragraph are combined, such as `runin` and `drop`. The *cmd* argument holds the heading command name to be manipulated. The remaining arguments are as follows:

*left-sep* Length specifying the increase of the left margin for headings with the `block`, `display`, `hang`, or `frame` shape. With `...margin` or `drop` shapes it specifies the width of the heading title, with `wrap` it specifies the maximum width for the title, and with `runin` it specifies the indentation before the title (negative values would make the title hang into the left margin).

*before-sep* Length specifying the vertical space added above the heading.

*after-sep* Length specifying the separation between the heading and the following paragraph. It can be a vertical or horizontal space depending on the shape deployed.

*right-sep* Optional length specifying an increase of the right margin, which is supported for the shapes `block`, `display`, `hang`, and `frame`.

The *before-sep* and *after-sep* arguments usually receive rubber length values to allow some flexibility in the design. To simplify the declaration you can alternatively specify *\*f* (where *f* is a decimal factor). This is equivalent to *f ex* with some stretchability as well as a small shrinkability inside *before-sep*, and an even smaller stretchability and no shrinkability inside *after-sep*.

<p>...some text before ...</p> <p>SECTION 1</p> <p><b>A Title Test</b></p> <p>Some text to prove that this paragraph is not indented and that the title has a margin of 1pc on either side.</p>	<pre>\usepackage{titlesec} \titleformat{\section}[frame]{\normalfont   {\footnotesize \enspace SECTION \thesection   \enspace}{6pt}{\large\bfseries\filcenter} \titlespacing*\{\section}{1pc}{*4}{*2.3}{1pc} \ldots some text before \ldots \section{A Title Test} Some text to prove that this paragraph is not indented and that the title has a margin of 1pc on either side.</pre>
---	--

2-2-18

*Spacing tools for headings* The previous example introduced `\filcenter`, but there also exist `\filleft`, `\filright`, and `\fillast`—the latter produces an adjusted paragraph but centers the last line. These commands should be preferred to `\raggedleft` or `\raggedright` inside `\titleformat`, as the latter would cancel *left-sep* or *right-sep* set up by the `\titlespacing` command. Alternatively, you can use `\filinner` or `\filouter`, which resolve to `\filleft` or `\filright`, depending on the current page. However, due to T<sub>E</sub>X's asynchronous page makeup algorithm, they are only supported for headings that start a new page—for example, `\chapter` in most designs. See Example 2-2-21 on page 43 for a solution to this problem for other headings. Another useful spacing command is `\wordsep`, which refers to the interword space (including stretch and shrink) of the current font.

*Indentation after heading* The paragraph indentation for the first paragraph following the headings can alternatively be globally specified using the package options `indentafter` or `noindentafter`, bypassing the presence or absence of a star in `\titlespacing`.

*Spacing between consecutive headings* By default, the spacing between two consecutive headings is defined to be the *after-sep* of the first one. If this result is not desired you can change it by specifying the option `largestsep`, which will put the spacing to the maximum of *after-sep* from the first heading and *before-sep* of the second.

*Headings at page bottom* After a heading L<sup>A</sup>T<sub>E</sub>X tries to ensure that at least two lines from the following paragraph appear on the same page as the heading title. If this proves impossible the heading is moved to the next page. If you think that two lines are not enough, try the option `nobottomtitles` or `nobottomtitles*`, which will move headings to a new page whenever the remaining space on the page is less than the current value of `\bottomtitlespace`. (Its default is  $.2\text{textheight}$ ; to change its value, use `\renewcommand` rather than `\setlength`.) The starred version is preferred, as it computes the remaining space with more accuracy, unless you use headings

with `drop`, `margin`, or `wrap shapes`, which may get badly placed when deploying the starred option.

In most heading layouts the number appears either on top or to the left of the heading text. If this placement is not appropriate, the `label` argument of `\titleformat` cannot be used. Instead, one has to exploit the fact that the `before-code` can pick up the heading text. In the next example, the command `\secformat` has one argument that defines the formatting for the heading text and number; we then call this command in the `before-code` argument of `\titleformat`. Note that the font change for the number is kept local by surrounding it with braces. Without them the changed font size might influence the title spacing in some circumstances.

*Handling unusual layouts*

A Title  
on Two Lines

1

```
\usepackage{titlesec}
\newcommand\secformat[1]{%
  \parbox[b]{.5\textwidth}{\filleft\bfseries #1}%
  \quad\rule[-12pt]{2pt}{70pt}\quad
  {\fontsize{60}{60}\selectfont\thesection}}
\titleformat{\section}[block]
  {\filleft\normalfont\sffamily}{\secformat}
  {\titlespacing*\secformat{0pt}{*3}{*2}{1pc}}
\section{A Title\\ on Two Lines}
```

In this example the heading number appears to the right of the heading text.

[2-2-19]

In this example the heading number appears to the right of the heading text.

The same technique can be applied to change the heading text in other ways. For example, if we want a period after the heading text we could define

```
\newcommand\secformat[1]{#1.}
```

and then call `\secformat` in the `before-code` of the `\titleformat` declaration as shown in the previous example.

The `wrap` shape has the capability to measure the lines in the title text and return the width of the widest line in `\titlewidth`. This capability can be extended to three other shapes (`block`, `display`, and `hang`) by loading the package with the option `calcwidth` and then using `\titlewidth` within the arguments of `\titleformat`, as needed.

*Measuring the width of the title*

For rules and leaders the package offers the `\titlerule` command. Used without any arguments it produces a rule of height `.4pt` spanning the full width of the column (but taking into account changes to the margins as specified with the `\titlespacing` declaration). An optional argument lets you specify a height for the produced rule. The starred form of `\titlerule` is used to produce leaders (i.e., repeated objects) instead of rules. It takes an optional `width` argument and a mandatory `text` argument. The `text` is repeatedly typeset in boxes with its natural width, unless a different `width` is specified in the optional argument. In that case,

*Rules and leaders*

only the first and the last boxes retain their natural widths to allow for proper alignment on either side.

The command `\titleline` lets you add horizontal material to arguments of `\titleformat` that expect vertical material. It takes an optional argument specifying the alignment and a mandatory argument containing the material to typeset. It produces a box of fixed width taking into account the marginal changes due to the `\titlespacing` declaration. Thus, either the material needs to contain some rubber space, or you must specify an alignment through the optional argument (allowed values are `l`, `r`, and `c`).

The `\titleline*` variant first typesets the material from its mandatory argument in a box of width `\titlewidth` (so you may have to add rubber space to this argument) and then uses this box as input to `\titleline` (i.e., aligns it according to the optional argument). Remember that you may have to use the option `calcwidth` to ensure that `\titlewidth` contains a sensible value.

In the next somewhat artificial example, which is worth studying though better not used in real life, all of these tools are applied together:

## Rules and Leaders

---

### Section 1

```
\usepackage[noindentafter,calcwidth]{titlesec}
\titleformat{\section}[display]
{\\filright\\normalfont\\bfseries\\sffamily}
{\\titleline[r]{Section \\Huge\\thesection}\\{1ex}
 {\\titleline*[1]{\\titlerule[1pt]}\\vspace{1pt}%
 {\\titleline*[1]{\\titlerule[2pt]}\\vspace{2pt}}%
 [{\\titleline*[1]{\\titlerule*{\\tiny\\LaTeX}}}]}
\\titlespacing{\\section}{1pc}{*3}{*2}}
\\section{Rules and Leaders}
Note that the last \\verb=\\titleline*= is
surrounded by braces. Without them its
optional argument would prematurely end the
outer optional argument of \\verb=\\titleformat=. 2-2-20
```

Note that the last `\titleline*` is surrounded by braces. Without them its optional argument would prematurely end the outer optional argument of `\titleformat`.

*Breaking before a heading* Standard LATEX considers the space before a heading to be a good place to break the page unless the heading immediately follows another heading. The penalty to break at this point is stored in the internal counter `\@secpenalty` and in many classes it holds the value -300 (negative values are bonus places for breaking). As only one penalty value is available for all heading levels, there is seldom any point in modifying its setting. With `titlesec`, however, you can exert finer control: whenever a command `\namebreak` is defined (where `\name` is the name of a sectioning command, such as `\sectionbreak`), the latter will be used instead of adding the default penalty. For example,

```
\newcommand\\sectionbreak{\\clearpage}
```

would result in sections always appearing on top of a page with all pending floats being typeset first.

In some layouts the space above a heading must be preserved, even if the heading appears on top of a page (by default, such spaces vanish at page breaks). This can be accomplished using a definition like the following:

*Always keeping the  
space above a  
heading*

```
\newcommand\sectionbreak{\addpenalty{-300}\vspace*{0pt}}
```

The `\addpenalty` command indicates a (good) break point, which is followed by a zero space that cannot vanish. Thus, the “before” space from the heading will appear as well at the top of the page if a break is taken at the penalty.

### Conditional heading layouts

So far we have seen how to define fixed layouts for a heading command using `\titleformat` and `\titlespacing`. The `titlesec` package also allows you to conditionally change the layout on verso and recto pages, and to use special layouts for numberless headings (i.e., those produced by the starred form of the heading command).

This is implemented through a keyword/value syntax in the first argument of `\titleformat` and `\titlespacing`. The available keys are `name`, `page` (values `odd` or `even`), and `numberless` (values `true` or `false`). In fact, the syntax we have seen so far, `\titleformat{\section}{...}...`, is simply an abbreviation for the general form `\titleformat{name=\section}{...}...`.

In contrast to the spacing commands `\filinner` and `\filouter`, which can only be used with headings that start a new page, the `page` keyword enables you to define layouts that depend on the current page without any restriction. To specify the layout for a verso (left-hand) page, use the value `even`; for a recto (right-hand) page, use the value `odd`. Such settings only affect a document typeset in `twoside` mode. Otherwise, all pages are considered to be recto in L<sup>A</sup>T<sub>E</sub>X. In the following example we use a `block` shape and shift the heading to one side, depending on the current page. In a similar fashion you could implement headings that are placed in the margin by using the shapes `leftmargin` and `rightmargin`.

<p><b>1. A Head</b></p> <p>Some text to fill the page. Some text to fill the page.</p>	<p>Some text to fill the page.</p> <p><b>2. Another</b></p> <p>Some text to fill the page.</p>	<pre>\usepackage{titlesec} \titleformat{name=\section,page=odd}[block]   {\normalfont\thesection.}{6pt}{\bfseries\filleft} \titleformat{name=\section,page=even}[block]   {\normalfont\thesection.}{6pt}{\bfseries\filright} \section{A Head} Some text to fill the page. Some text to fill the page. \newpage Some text to fill the page. \section{Another} Some text to fill the page.</pre>
--	--	--

2-2-21 ,

Similarly, the `numberless` key is used to specify that a certain `\titleformat` or `\titlespacing` declaration applies only to headings with (or without) numbers. By default, a heading declaration applies to both cases, so in the example the

second declaration actually overwrites part of the first declaration. To illustrate what is possible the example uses quite different designs for the two cases—do not mistake this for an attempt to show good taste. It is important to realize that neither the *label* nor the *sep* argument is ignored when *numberless* is set to *true* as seen in the example—in normal circumstances you would probably use `\{}{0pt}` as values.

### 1. A Head

Some text to fill the page. Some text to fill the page.

\*\*\* Another

Some text to fill this line.

```
\usepackage{titlesec}
\titleformat{name=\section}[block]
  {\normalfont{\thesection.\,}{6pt}{\bfseries\filright}}
\titleformat{name=\section,numberless=true}[block]
  {\normalfont{***}{12pt}{\itshape\filcenter}}
\section{A Head}
Some text to fill the page. Some text to fill the page.
\section*{Another}
Some text to fill this line.
```

2-2-22

### Changing the heading hierarchy

The commands described so far are intended to adjust the formatting and spacing of existing heading commands. With the `\titleclass` declaration it is possible to define new headings.

```
\titleclass{cmd}{class}
\titleclass{cmd}{class}[super-level-cmd]
\titleclass{cmd}[start-level]{class} (with loadonly option)
```

There are three classes of headings: the *page* class contains headings that fill a full page (like `\part` in L<sup>A</sup>T<sub>E</sub>X's report and book document classes); the *top* class contains headings that start a new page and thus appear at the top of a page; and all other headings are considered to be part of the *straight* class.

Used without any optional argument the `\titleclass` declaration simply changes the heading class of an existing heading *cmd*. For example,

```
\titleclass\section{top}
```

would result in sections always starting a new page.

If this declaration is used with the optional *super-level-cmd* argument, you introduce a new heading level below *super-level-cmd*. Any existing heading command at this level is moved one level down in the hierarchy. For example,

```
\titleclass\subchapter{straight}[\chapter]
```

introduces the heading `\subchapter` between `\chapter` and `\section`. The declaration does not define any layout for this heading (which needs to be defined by an additional `\titleformat` and `\titlespacing` command), nor does it initialize

the necessary counter. Most likely you also want to update the counter representation for `\section`:

```
\titleformat{\subchapter}{...}... \titlespacing{\subchapter}{...}...
\newcounter{subchapter}
\renewcommand\thesubchapter{\thechapter.\arabic{subchapter}}
\renewcommand\thesection{\thesubchapter.\arabic{section}}
```

The third variant of `\titleclass` is needed only when you want to build a heading structure from scratch—for example, when you are designing a completely new document class that is not based on one of the standard classes. In that case load the package with the option `loadonly` so that the package will make no attempt to interpret existing heading commands so as to extract their current layout. You can then start building heading commands, as in the following example:

```
\titleclass{\Ahead}[0]{top}
\titleclass{\Bhead[straight]}[\Ahead]
\titleclass{\Chead[straight]}[\Bhead]
\newcounter{\Ahead} \newcounter{\Bhead} \newcounter{\Chead}
\renewcommand{\Bhead}{\thead-\arabic{\Bhead}}
\renewcommand{\Chead}{\thead-\arabic{\Chead}}
\titleformat{name=\Ahead}{...}... \titlespacing{name=\Ahead}{...}...
\titleformat{name=\Bhead}{...}... ...
```

The *start-level* is usually 0 or -1; see the introduction in Section 2.2 for its meaning. There should be precisely one `\titleclass` declaration that uses this particular optional argument.

If you intend to build your own document classes in this way, take a look at the documentation accompanying the `titlesec` package. It contains additional examples and offers further tips and tricks.

## 2.3 Table of contents structures

A *table of contents* (TOC) is a special list in which the titles of the section units are listed, together with the page numbers indicating the start of the sections. This list can be rather complicated if units from several nesting levels are included, and it should be formatted carefully because it plays an important rôle as a navigation aid for the reader.

Similar lists exist containing reference information about the floating elements in a document—namely, the *list of tables* and the *list of figures*. The structure of these lists is simpler, as their contents, the captions of the floating elements, are normally all on the same level (but see Section 6.5.2).

Standard L<sup>A</sup>T<sub>E</sub>X can automatically create these three contents lists. By default, L<sup>A</sup>T<sub>E</sub>X enters text generated by one of the arguments of the sectioning commands into the *.toc* file. Similarly, L<sup>A</sup>T<sub>E</sub>X maintains two more files, one for the list of figures (*.lof*) and one for the list of tables (*.lot*), which contain the text specified as the argument of the `\caption` command for figures and tables.

The information written into these files during a previous L<sup>A</sup>T<sub>E</sub>X run is read and typeset (normally at the beginning of a document) during a subsequent L<sup>A</sup>T<sub>E</sub>X run by invoking these commands: `\tableofcontents`, `\listoffigures`, and `\listoftables`.

*A TOC needs two, sometimes even three, L<sup>A</sup>T<sub>E</sub>X runs*

To generate these cross-reference tables, it is always necessary to run L<sup>A</sup>T<sub>E</sub>X at least twice—once to collect the relevant information, and a second time to read back the information and typeset it in the correct place in the document. Because of the additional material to be typeset in the second run, the cross-referencing information may change, making a third L<sup>A</sup>T<sub>E</sub>X run necessary. This is one of the reasons for the tradition of using different page-numbering systems for the front matter and the main text: in the days of hand typesetting any additional iteration made the final product much more expensive.

The following sections will discuss how to typeset and generate these contents lists. It will also be shown how to enter information directly into one of these auxiliary files and how to open and write into a supplementary file completely under user control.

### 2.3.1 Entering information into the contents files

Normally the contents files are generated automatically by L<sup>A</sup>T<sub>E</sub>X. With some care this interface, which consists of the `\addcontentsline` and `\addtocontents` commands, can also be used to enter information directly.

```
\addcontentsline{ext}{type}{text}
```

The `\addcontentsline` command writes the *text* together with some additional information, such as the page number of the current page, into a file with the extension *ext* (usually *.toc*, *.lof*, or *.lot*). Fragile commands within *text* needed to be protected with `\protect`. The *type* argument is a string that specifies the kind of contents entry that is being made. For the table of contents (*.toc*), it is usually the name of the heading command without a backslash; for *.lof* or *.lot* files, *figure* or *table* is normally specified.

The `\addcontentsline` instruction is normally invoked automatically by the document sectioning commands or by the `\caption` commands within the float environments. Unfortunately, the interface has only one argument for the variable text, which makes it awkward to properly identify an object's number if present. Since such numbers (e.g., the heading number) typically need special formatting in the contents lists, this identification is absolutely necessary. The trick used by the current L<sup>A</sup>T<sub>E</sub>X kernel to achieve this goal is to surround such a number with the

command `\numberline` within the `text` argument as follows:

```
\protect\numberline{number}heading
```

For example, a `\caption` command inside a `figure` environment saves the caption text for the figure using the following line:

```
\addcontentsline{lof}{figure}
{\protect\numberline{\thefigure}caption text}
```

Because of the `\protect` command, `\numberline` will be written unchanged into the external file, while `\thefigure` will be executed along the way so that the actual figure number will end up in the file.

Later on, during the formatting of the contents lists, `\numberline` can be used to format the number in a special way, such as by providing extra space or a different font. The downside of this approach is that it is less general than a version that takes a separate argument for this number (e.g., you cannot easily do arbitrary transformation on this number) and it requires a suitable definition for `\numberline`—something that is unfortunately not always available (see the discussion in Section 2.3.2 on page 49).

Sometimes `\addcontentsline` is used in the source to complement the actions of standard L<sup>A</sup>T<sub>E</sub>X. For instance, in the case of the starred form of the section commands, no information is written to the `.toc` file. If you do not want a heading number (starred form) but you do want an entry in the `.toc` file, you can use `\addcontentsline` with or without `\numberline` as shown in the following example.

<b>Contents</b>  <b>Foreword</b>  <b>1 Thoughts</b> 1.1 Contact info . . .  <b>References</b>	<b>1 Thoughts</b>  1 We find all in [1].  <b>2 1.1 Contact info</b> E-mail Ben at [2].  <b>Foreword</b>  A starred heading with the TOC entry manually added. Compare this to the form used for the bibliography.	<b>References</b>  [1] Ben User, Some day will never come, 2010 [2] BUUser@earth.info	<pre>\tableofcontents \section*{Foreword} \addcontentsline{toc}{section} {\protect\numberline{}Foreword} A starred heading with the TOC entry manually added. Compare this to the form used for the bibliography. \section{Thoughts} We find all in \cite{k1}. \subsection{Contact info} E-mail Ben at \cite{k2}. \begin{thebibliography}{9} \addcontentsline{toc}{section}{\refname} \bibitem{k1} Ben User, Some day will never come, 2010 \bibitem{k2} BUUser@earth.info \end{thebibliography}</pre>
--	--	--	--

Using `\numberline` as in the “Foreword” produces an indented “section” entry in the table of contents, leaving the space where the section number would go free. Omitting the `\numberline` command (as was done for the bibliography entry) would typeset the heading flush left instead. Adding a similar line after the start of the `\theindex` means that the “Index” will be listed in the table of contents. Unfortunately, this approach cannot be used to get the list of figures or tables into the table of contents because `\listoffigures` or `\listoftables` might generate a listing of several pages and consequently the page number picked up by `\addcontentsline` might be wrong. And putting it before the command does not help either, because often these list commands start a new page. One potential solution is to copy the command definition from the class file and put `\addcontentsline` directly into it.

*Bibliography or index in table of contents*

In case of standard classes or close derivatives you can use the `tocbibind` package created by Peter Wilson to get the “List of…”, “Index”, or “Bibliography” section listed in the table of contents without further additions to the source. The package offers a number of options such as `notbib`, `notindex`, `nottoc`, `notlof`, and `notlot` (do not add the corresponding entry to the table of contents) as well as `numbib` and `numindex` (number the corresponding section). By default the “Contents” section is listed within the table of contents, which is seldom desirable; if necessary, use the `nottoc` option to disable this behavior.

```
\addtocontents{ext}{text}
```

The `\addtocontents` command does not contain a *type* parameter and is intended to enter special formatting information not directly related to any contents line. For example, the `\chapter` command of the standard classes places additional white space in the `.lof` and `.lot` files to separate entries from different chapters as follows:

```
\addtocontents{lof}{\protect\addvspace{10pt}}
\addtocontents{lot}{\protect\addvspace{10pt}}
```

By using `\addvspace` at most 10 points will separate the entries from different chapters without producing strange gaps if some chapters do not contain any figures or tables.

*Potential problems with \addvspace*

This example, however, shows a certain danger of the interface: while the commands `\addcontentsline`, `\addtocontents`, and `\addvspace` appear to be user-level commands (they do not contain any @ signs in their names), they can easily produce strange errors.<sup>1</sup> In particular, `\addvspace` can be used only in vertical mode, which means that a line like the above works correctly only if an earlier `\addcontentsline` ends in vertical mode. Thus, you need to understand

<sup>1</sup>For an in-depth discussion of `\addvspace`, see Appendix A.1.5, page 858.

how such lines are actually processed to be able to enter arbitrary formatting instructions between them. This is the topic of the next section.

If either `\addcontentsline` or `\addtocontents` is used within the source of a document, one important restriction applies: neither command can be used at the same level as an `\include` statement. That means, for example, that the sequence

```
\addtocontents{toc}{\protect\setcounter{tocdepth}{1}}
\include{sect1}
```

with `sect1.tex` containing a `\section` command would surprisingly result in a `.toc` file containing

```
\contentsline {section}{\numberline {1}Section from sect1}{2}
\setcounter {tocdepth}{1}
```

showing that the lines appear out of order. The solution is to move the `\addtocontents` or `\addcontentsline` statement into the file loaded via `\include` or to avoid `\include` altogether.

*Potential problems  
with \include*

### 2.3.2 Typesetting a contents list

As discussed above, contents lists are generated by implicitly or explicitly using the commands `\addcontentsline` and `\addtocontents`. The exact effect of

```
\addcontentsline{ext}{type}{text}
```

is to place the line

```
\contentsline{type}{text}{page}
```

into the auxiliary file with extension `ext`, where `page` is the current page number in the document. The command `\addtocontents{ext}{text}` is simpler: it just puts `text` into the auxiliary file. Thus, a typical contents list file consists of a number of `\contentsline` commands, possibly interspersed with further formatting instructions added as a result of `\addtocontents` calls. It is also possible for the user to create a table of contents by hand with the help of the command `\contentsline`.

A typical example is shown below. Note that most (though not all) heading numbers are entered as a parameter of the `\numberline` command to allow for formatting with the proper indentation. LATEX is unfortunately not consistent here; the standard classes do not use `\numberline` for `\part` headings but instead specify the separation between number and text explicitly. Since the 2001/06/01 release you can also use `\numberline` in this place, but with older releases the formatting will be unpredictable.

 *Inconsistency  
with \part*

<b>I Part</b>	<b>2</b>	\setcounter{tocdepth}{3}
<b>1 A-Head</b>	<b>2</b>	\contentsline{part}{I\hspace{1em}Part}{2}
1.1 B-Head . . . . .	<b>3</b>	\contentsline{chapter}{\numberline{1}A-Head}{2}
1.1.1 C-Head . . . . .	<b>4</b>	\contentsline{section}{\numberline{1.1}B-Head}{3}
With Empty Number .	<b>5</b>	\contentsline{subsection}{\numberline{1.1.1}C-Head}{4}
Unnumbered C-Head . . . . .	<b>6</b>	\contentsline{subsection}{\numberline{}With Empty Number}{5}
		\contentsline{subsection}{\numberline{}Unnumbered C-Head}{6}

2-3-2

The `\contentsline` command is implemented to take its first argument *type*, and then use it to call the corresponding `\l@type` command, which does the actual typesetting. One separate command for each of the types must be defined in the class file. For example, in the report class you find the following definitions:

```
\newcommand{\l@section} {\@dottedtocline{1}{1.5em}{2.3em}}
\newcommand{\l@subsection} {\@dottedtocline{2}{3.8em}{3.2em}}
\newcommand{\l@subsubsection} {\@dottedtocline{3}{7.0em}{4.1em}}
\newcommand{\l@paragraph} {\@dottedtocline{4}{10em}{5em}}
\newcommand{\l@ subparagraph} {\@dottedtocline{5}{12em}{6em}}
\newcommand{\l@figure} {\@dottedtocline{1}{1.5em}{2.3em}}
\newcommand{\l@table} {\l@figure}
```

By defining `\l@type` to call `\@dottedtocline` (a command with five arguments) and specifying three arguments (*level*, *indent*, and *numwidth*), the remaining arguments, *text* and *page*, of `\contentsline` will be picked up by `\@dottedtocline` as arguments 4 and 5.

Note that some section levels build their table of contents entries in a somewhat more complicated way, so that the standard document classes have definitions for `\l@part` and `\l@chapter` (or `\l@section` with article) that do not use `\@dottedtocline`. Generally they use a set of specific formatting commands, perhaps omitting the ellipses and typesetting the title in a larger font.

So to define the layout for the contents lists, we have to declare the appropriate `\l@type` commands. One easy way to do this, as shown above, is to use `\@dottedtocline`, an internal command that we now look at in some detail.

```
\@dottedtocline{level}{indent}{numwidth}{text}{page}
```

The last two parameters of `\@dottedtocline` coincide with the last parameters of `\contentsline`, which itself usually invokes a `\@dottedtocline` command. The other parameters are the following:

*level* The command `\contentsline` nesting level of an entry. This parameter allows the user to control how many nesting levels will be displayed. Levels greater than the value of counter `tocdepth` will not appear in the table of contents.

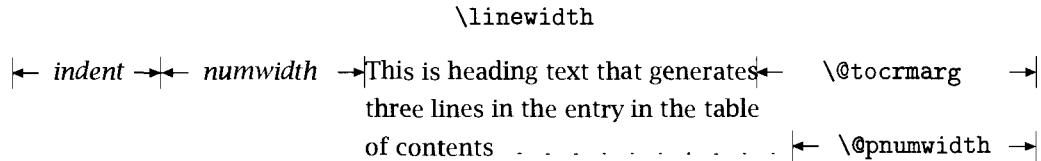


Figure 2.3: Parameters defining the layout of a contents file

*indent* The total indentation from the left margin.

*numwidth* The width of the box that contains the number if *text* has a `\numberline` command. It is also the amount of extra indentation added to the second and later lines of a multiple-line entry.

Additionally, the command `\@dottedtocline` uses the following global formatting parameters, which specify the visual appearance of all entries:

`\@pnumwidth` The width of the box in which the page number is set.

`\@tocrmarg` The indentation of the right margin for all but the last line of multiple-line entries. Dimension, but changed with `\renewcommand!` It can be set to a rubber length, which results in the TOC being set unjustified.

`\@dotsep` The separation between dots, in `\mu` (math units).<sup>1</sup> It is a pure number (like 1.7 or 2). By making this number large enough you can get rid of the dots altogether. To be changed with `\renewcommand!`

A pictorial representation of the effects described is shown in Figure 2.3. The field identified by *numwidth* contains a left-justified section number, if present. You can achieve the proper indentation for nested entries by varying the settings of *indent* and *numwidth*.

One case in which this is necessary, while using a standard class (`article`, `report`, or `book`), arises when you have ten or more sections and within the later ones more than nine subsections. In that case numbers and text will come too close together or even overlap if the *numwidth* argument on the corresponding calls to `\@dottedtocline` is not extended, as seen in the following example.

*Problem with many headings on one level*

## 10 A-Head

10.9 B-Head . . . . .	4	<code>\contentsline{section}{\numberline{10}A-Head}{3}</code>
10.10B-Head . . . . .	4	<code>\contentsline{subsection}{\numberline{10.9}B-Head}{4}</code>

2-3-3

<sup>1</sup>There are 18 `\mu` units to an `\em`, where the latter is taken from the `\fontdimen2` of the math symbol font `symbols`. See Section 7.10.3 for more information about `\fontdimens`.

Redefining `\l@subsection` to leave more space for the number (third argument to `\@dottedtocline`) gives a better result in this case. You will probably have to adjust the other commands, such as `\l@subsubsection`, as well to produce a balanced look for the whole table.

			<code>\makeatletter</code>
			<code>\renewcommand{\l@subsection}{\@dottedtocline{2}{1.5em}{3em}}</code>
			<code>\makeatother</code>
<b>10 A-Head</b>	<b>3</b>		<code>\contentsline{section}{\numberline{10}A-Head}{3}</code>
10.9 B-Head . . . .	4		<code>\contentsline{subsection}{\numberline{10.9}B-Head}{4}</code>
10.10 B-Head . . . .	4		<code>\contentsline{subsection}{\numberline{10.10}B-Head}{4}</code>

2-3-4

Another example that requires changes is the use of unusual page numbering. For example, if the pages are numbered by part and formatted as “A-78”, “B-328”, and so on, then the space provided for the page number is probably too small, resulting at least in a large number of annoying “Overfull hbox” warnings, but more likely in some bad spacing around them. In that case the remedy is to set `\@pnumwidth` to a value that fits the widest entry—for example, via

```
\makeatletter \settowidth{\pnumwidth}{\textbf{A--123}} \makeatother
```

When adjusting `\@pnumwidth` this way it is likely that the value of `\@tocrmarg` needs to be changed as well to keep the layout of the table of contents consistent.

The level down to which the heading information is displayed in the table of contents is controlled by the counter `tocdepth`. It can be changed, for example, with the following declaration:

```
\setcounter{tocdepth}{2}
```

In this case section heading information down to the second level (e.g., in the `report` class part, chapter, and section) will be shown.

### 2.3.3 Combining contents lists

By default, L<sup>A</sup>T<sub>E</sub>X produces separate lists for the table of contents, the list of figures, and the list of tables, available via `\tableofcontents`, `\listoffigures`, and `\listoftables`, respectively. None of the standard classes support combining those lists, such as having all tables and figures in a single list, or even combining all three in a single table of contents as is sometimes requested.

How could such a request be fulfilled? The first requirement is that we make L<sup>A</sup>T<sub>E</sub>X write to the appropriate auxiliary file when it internally uses `\addcontentsline`. For example, all `\caption` commands need to write to a single file if we want to combine figures and tables in a single list. Looking at the L<sup>A</sup>T<sub>E</sub>X sources reveals that this goal is easy to achieve: figure captions write to a file with the extension specified by `\ext@figure`, while table captions use `\ext@table` for this purpose.

So using an appropriate redefinition of, say, `\ext@table` we can force L<sup>A</sup>T<sub>E</sub>X to assemble all references to figures and tables in the `.lof` file. But is this enough? The example clearly shows that it is probably not enough to force the entries together. When looking at the generated list we cannot tell which entry refers to a figure or a table. The only indication that something is amiss is given by the identical numbers on the left.

A figure

Figure 1: Figure-Caption

## Figures and Tables

1	Figure-Caption . . . . .	1
1	Table-Caption . . . . .	1

### 1 A Section

Some text ... Some text referencing figure 1 ...

A table

Table 1: Table-Caption

[2-3-5]

```
\makeatletter
\renewcommand{\ext@table}{\lof}
\makeatother
\renewcommand{\listfigurename}{Figures and Tables}
\listoffigures
\section{A Section}
Some text \ldots
\begin{table}[b]
\centering
\fbox{\scriptsize A table}
\caption{Table-Caption}
\end{table}
Some text referencing
figure~\ref{fig} \ldots
\begin{figure}
\centering
\fbox{\scriptsize A figure}
\caption{Figure-Caption}\label{fig}
\end{figure}
```

The situation would be slightly better if the figures and tables share the same counter, so that we do not end up with identical numbers in the left column of the list. Unfortunately, this result is fairly difficult to achieve, because one must directly manipulate the low-level float definitions.

Another possible remedy is to define `\l@figure` and `\l@table` in such a way that this information is present. The example on the following page shows a possible solution that appends the string “(figure)” or “(table)” to each entry. In theory it would also be possible to annotate the number to indicate the type of float, but that would require redefining a lot of L<sup>A</sup>T<sub>E</sub>X’s internals such as `\numberline`.

What happens if we force all entries into a single list—that is, into the table of contents? In that case we get a list ordered according to the final appearance of the objects in the document, which may not be what we would expect to see. In the next example, the figure, which actually came last in the source, shows up before the section in which it is referenced, because the float algorithm places it on the top of the page. This outcome might be acceptable within books or reports where the major heading starts a new page and prevents top floats on the heading page, but is probably not desirable in other cases.

A figure

Figure 1: Figure-Caption

## Contents

1	Figure-Caption (figure) . . . . .	1
<b>1</b>	<b>A Section</b>	<b>1</b>
1	Table-Caption (table) . . . . .	1

### 1 A Section

Some text ... Some text referencing figure 1 ...

A table

Table 1: Table-Caption

```
\makeatletter
\renewcommand{\ext@figure{toc}}
\renewcommand{\ext@table{toc}}
\renewcommand{\l@figure[2]{\@dottedtocline
    {1}{1.5em}{2.3em}{#1~(figure)}{#2}}}
\renewcommand{\l@table [2]{\@dottedtocline
    {1}{1.5em}{2.3em}{#1~(table)}{#2}}}
\makeatother
\tableofcontents
\section{A Section}
Some text \ldots
\begin{table}[b]
    \centering \fbox{\scriptsize A table}
    \caption{Table-Caption}
\end{table}
Some text referencing
figure~\ref{fig} \ldots
\begin{figure}
    \centering \fbox{\scriptsize A figure}
    \caption{Figure-Caption}\label{fig}
\end{figure}
```

2-3-6

In summary, while it is possible to combine various types of contents lists, the results may not be what one would expect. In any case such an approach requires a careful redesign of all `\l@type` commands so that the final list will be useful to the reader.

#### 2.3.4 Providing additional contents files

If you want to make a list comprising all of the examples in a book, you need to create a new contents file and then make use of the facilities described above. First, two new commands must be defined. The first command, `\ecaption`, associates a caption with the current position in the document by writing its argument and the current page number to the contents file. The second command, `\listoffexamples`, reads the information written to the contents file on the previous run and typesets it at the point in the document where the command is called.

The `\listoffexamples` command invokes `\@starttoc{ext}`, which reads the external file (with the extension `ext`) and then reopens it for writing. This command is also used by the commands `\tableofcontents`, `\listoffigures`, and `\listoftables`. The supplementary file could be given an extension such as `xmp`. A command like `\chapter*{List of examples}` can be put in front or inside of `\listoffexamples` to produce a title and, if desired, a command `\addcontentsline` can signal the presence of this list to the reader by entering it into the `.toc` file.

The actual typesetting of the individual entries in the `.xmp` file is controlled by `\l@example`, which needs to be defined. In the example below, the captions are typeset as paragraphs followed by an italicized page number.

## 1 Selection of recordings

Ravel's Boléro by Jacques Loussier Trio.  
 Davis' Blue in Green by Cassandra Wilson.

### Comments

Loussier: A strange experience, *I*  
 Wilson: A wonderful version, *I*

```
\newcommand\ecaption[1]
  {\addcontentsline{xmp}{example}{#1}}
\makeatletter \newcommand\listofexamples
  {\section*{Comments}\@starttoc{xmp}}
\newcommand\l@example[2]
  {\par\noindent#1,\textit{#2}\par} \makeatother
\section{Selection of recordings}
Ravel's Bol\`ero by Jacques Loussier
Trio.\ecaption{Loussier: A strange experience}

Davis' Blue in Green by Cassandra
Wilson.\ecaption{Wilson: A wonderful version}
\listofexamples
```

The `float` package described in Section 6.3.1 on page 291 implements the above mechanism with the command `\listof`, which generates a list of floats of the type specified as its argument.

### 2.3.5 shorttoc—Summary table of contents

With larger documents it is sometimes helpful to provide a summary table of contents (with only the major sections listed) as well as a more detailed one. This can be accomplished with the `shorttoc` package created by Jean-Pierre Drucbert.

```
\shorttableofcontents{title}{depth}
```

This `\shorttableofcontents` command (or `\shorttoc` as a synonym) must be specified before the `\tableofcontents` command; otherwise, the summary table of contents will be empty. The table's heading is given by the *title* argument and the *depth* down to which contents entries are shown is defined by the second argument. Thus, to show only chapters and sections in the summary and everything down to subsubsections in the detailed table of contents, you would specify:

```
\shorttableofcontents{Summary table of contents}{1}
\setcounter{tocdepth}{3}
\tableofcontents
```

The package supports two options, `loose` (default) and `tight`, that deal with the vertical spacing of the summary table.

### 2.3.6 minitoc—Multiple tables of contents

The `minitoc` package, originally written by Nigel Ward and Dan Jurafsky and completely redesigned by Jean-Pierre Drucbert, enables the creation of mini-tables of contents (a “minitoc”) for chapters, sections, or parts. It also supports the creation of mini-tables for the list of figures and list of tables contained in a chapter, section, or part. A similar functionality, albeit using a completely different approach, is provided by the `titletoc` package described in Section 2.3.7.

Here we describe in some detail the use of the package to generate such tables on a per-chapter basis. The generation of per-section or per-part tables is completely analogous (using differently named commands); an overview appears at the end of the section.

The package supports almost all language options of the `babel` system (see Section 9.1.3), which predefine the heading texts used. In addition, the formatting of the generated tables can be influenced by the options `loose` (default) or `tight` and `dotted` (default) or `undotted`. Further control over the appearance is provided by a number of parameters that can be set in the preamble (see Table 2.3 on the next page).

To initialize the `minitoc` system, you place a `\dominitoc` command before the `\tableofcontents` command. If you do not want a full table of contents but only mini-tables, replace the latter command with `\faketableofcontents`. Mini-lists of figures or tables are initialized similarly, by using `\dominilof` or `\dominilot`, if necessary together with `\fakelistoffigures` or `\fakelistoftables`.

The `\domini...` commands accept one optional argument to denote the position of the table titles: `l` for left (default), `c` for center, `r` for right, or `n` for no title (a supported synonym is `e` for empty). The declaration is global for all tables in the document.

The actual mini-tables of contents are then generated by putting the command `\minitoc` in suitable places (typically directly after a `\chapter` command) inside the document. The actual placement is at your discretion. For instance, you may put some text before it or place a box around it. If one of the tables is empty, the package suppresses the heading and issues a warning to alert you about possible formatting problems due to the material added by you around the command.

If you want to generate mini-lists of figures or tables, you use `\minilof` or `\minilot` after initializing the system as explained above.

For each mini-table of contents, an auxiliary file with the extension `.mtc<n>`, where `<n>` is the chapter number, will be created.<sup>1</sup> For mini-lists of figures and tables, files with the extensions `.mlf<n>` and `.mlt<n>` are created, respectively.

By default, the mini-tables contain only references to sections and subsections. The `minitocdepth` counter, similar to `tocdepth`, allows the user to modify this behavior. The fonts used for individual entries can also be modified by chang-

<sup>1</sup>A different scheme is automatically used for operating systems in which file extensions are limited to three characters, like MS-DOS. It can be explicitly requested using the option `shortext` on the `\usepackage` command.

<code>\minitocdepth</code>	A <i>L<small>A</small>T<small>E</small>X</i> counter that indicates how many levels of headings will be displayed in the mini-table (default value is 2).
<code>\mtcindent</code>	The length of the left/right indentation of the mini-table (default value is 24pt).
<code>\mtcfont</code>	Command defining the default font that is used for the mini-table entries (default definition is a small Roman font).
<code>\mtcSfont</code>	Command defining the font that is used for <code>\section</code> entries (default definition is a small bold Roman font).
<code>\mtcSSfont</code>	If defined, font used for <code>\subsection</code> entries (default is to use <code>\mtcfont</code> for this and the following).
<code>\mtcSSSfont</code>	If defined, font used for <code>\subsubsection</code> entries.
<code>\mtcPfont</code>	If defined, font used for <code>\paragraph</code> entries.
<code>\mtcSPfont</code>	If defined, font used for <code>\ subparagraph</code> entries.
<code>\mtctitle</code>	Title text for the mini-table of contents (preset by language option).
<code>\nomtcrule</code>	Declaration that disables rules above and below the mini-tables ( <code>\mtcrule</code> enables them).
<code>\nomtcpagenumbers</code>	Declaration that suppresses page numbers in the mini-tables ( <code>\mtcpagenumbers</code> enables them).

Table 2.3: A summary of the `minitoc` parameters

ing the definitions of `\mtcfont` and its companions shown in Table 2.3. You can influence the use of rules around the mini-tables by specifying `\mtcrule` (default) or `\nomtcrule` in the preamble or before individual mini-tables. Similarly, you can request the use of page numbers in the mini-table by using the `\mtcpagenumbers` declaration (default) or their suppression by using `\nomtcpagenumbers`.

As the mini-tables and mini-lists take up room within the document, their use will alter the page numbering. Therefore, three runs normally are needed to ensure correct information in the mini-table of contents.

For mini-tables and mini-lists on the `\part` level, commands similar to those in Table 2.3 are provided. The only difference is that their names contain the string `part` instead of `mini` or `ptc` instead of `mtc`. Thus, you would use `\doparttoc` to initialize the system, `\parttoc` to print a mini-table, `\noptcrules` to suppress rules, and so on. The only addition is the declaration `\ptcCfont`, which defines the font to use for chapter entries and which naturally has no equivalent.

For mini-tables and mini-lists on the `\section` command level, the situation is again similar: replace `mini` by `sect` or `mtc` by `stc`— for example, use `\dosecttoc`,

*Mini-tables on part or section level*

\secttoc, and \stcfont. If \sectlof or \sectlot commands are used, you may want to try the option placeins, which confines floats to their sections by using the placeins package with its options `below` and `section` (see Section 6.2.1 on page 288).

## 1 Afghanistan

<b>1.1 Geography . . .</b>	<b>1</b>
1.1.1 Total area . . .	1
1.1.2 Land area . . .	1
<b>1.2 History . . . . .</b>	<b>2</b>

### 1.1 Geography

#### 1.1.1 Total area

647,500 km<sup>2</sup>

#### 1.1.2 Land area

647,500 km<sup>2</sup>

1

## 1.2 History

...

## 2 Albania

<b>2.1 Geography . . .</b>	<b>2</b>
2.1.1 Total area . . .	2
2.1.2 Land area . . .	3
<b>2.2 History . . . . .</b>	<b>3</b>

### 2.1 Geography

#### 2.1.1 Total area

28,750 km<sup>2</sup>

2

```
\usepackage{minitoc}
\setlength\stcindent{0pt}
\renewcommand\stctitle{}
\renewcommand\stcfont
  {\footnotesize}
\setcounter{secttocdepth}{3}
\dosecttoc \faketableofcontents

\section{Afghanistan}\secttoc
\subsection{Geography}
\subsubsection{Total area}
  647,500 km\textsuperscript{2}
\subsubsection{Land area}
  647,500 km\textsuperscript{2}
\subsection{History} \ldots

\section{Albania} \secttoc
\subsection{Geography}
\subsubsection{Total area}
  28,750 km\textsuperscript{2}
\subsubsection{Land area}
  27,400 km\textsuperscript{2}
\subsection{History} \ldots
```

2-3-8

To turn off the \minitoc commands, merely replace the package minitoc with mtcoff on your \usepackage command. This step ensures that all minitoc-related commands in the source will be ignored.

### 2.3.7 titletoc—A different approach to contents lists

The titletoc package written by Javier Bezos was originally developed as a companion package to titlesec but can be used on its own. It implements its own interface to lay out contents structures, thereby avoiding some of the limitations of the original L<sup>A</sup>T<sub>E</sub>X code.

The actual generation of external contents files and their syntax is left unchanged so that it works nicely with other packages generating such files. There is one exception, however: contents files should end with the command \contentsfinish. For the standard file extensions .toc, .lof, and .lot, this is handled automatically. But if you provide your own type of contents lists (see

*Relation to standard L<sup>A</sup>T<sub>E</sub>X*

Section 2.3.4), you have to announce it to `\titletoc`, as in the following example:

```
\contentsuse{example}{xmp}
```

As explained in Section 2.3.2 a contents file consists of `\contentsline` commands that are sometimes separated by some arbitrary code due to the use of `\addtocontents`. To format such contents lines with standard L<sup>A</sup>T<sub>E</sub>X we had to define commands of the form `\l@type`; with `\titletoc`, this step is no longer needed. Instead, we declare the desired formatting using the `\titlecontents` declaration (for vertically oriented entries) or its starred form (for run-in entries).

```
\titlecontents{type}[left-indent]{above-code}{numbered-entry-format}
            {numberless-entry-format}{page-format}[below-code]
```

The first argument of `\titlecontents` contains the *type* of contents line for which we set up the layout—it corresponds to the first argument of `\contentsline`. In other words, for each *type* of sectioning command that can appear in the document, we need one `\titlecontents` declaration.<sup>1</sup> The remaining arguments have the following meaning:

*left-indent* Argument that specifies the indentation from the left margin for all lines of the entry. It is possible to place material (e.g., the heading number) in this space. Even though this argument has to be given in square brackets, it is *not* optional in the current package release!

*above-code* Code to be executed before the entry is typeset. It can be used to provide vertical space, such as by using `\addvspace`, and to set up formatting directives, such as font changes, for the whole entry. You can also use `\filleft`, `\filright`, `\filcenter`, or `\fillast`, already known from the `titlesec` package, at this point.

*numbered-entry-format* Code to format the entry including its number. It is executed in horizontal mode (after setting up the indentation). The last token can be a command with one argument, in which case it will receive the entry *text* as its argument. The unformatted heading number is available in the command `\thecontentslabel`, but see below for other possibilities to access and place it.

*numberless-entry-format* Code to format the entry if the current entry does not contain a number. Again the last token may be a command with one argument.

*page-format* Code that is executed after formatting the entry but while still being in horizontal mode. It is normally used to add some filling material, such as a dotted line, and to attach the page number stored in `\thecontentspage`. You can use the `\titlerule` command, discussed on page 41, to produce leaders.

<sup>1</sup>The package honors existing `\l@type` declarations made, for example, by the document class. Thus, it can be used to change the layout of only some types.

*below-code* An (optional) argument used to specify code to be executed in vertical mode after the entry is typeset—for example, to add some extra vertical space after the entry.

To help with placing and formatting the heading and page numbers, the `titletoc` package offers two useful tools: `\contentslabel` and `\contentspage`.

`\contentslabel[text]{size}`

The purpose of the `\contentslabel` command is to typeset the *text* (which by default contains `\thecontentslabel`) left aligned in a box of width *size* and to place that box to the left of the current position. Thus, if you use this command in the *numbered-entry-format* argument of `\titlecontents`, then the number will be placed in front of the entry text into the margin or indentation set up by *left-indent*. For a more refined layout you can use the optional argument to specify your own formatting usually involving `\thecontentslabel`.

*Package options* The package offers three options to influence the default outcome of the `\contentslabel` command when used without the *text* argument. With `rightlabels` the heading number is right aligned in the space. The default, `leftlabels`, makes it left aligned. With `dotinlabels` a period is added after the number.

`\contentspage[text]`

In similar fashion `\contentspage` typesets *text* (which by default contains `\thecontentspage`) right aligned in a box and arranges for the box to be placed to the right of the current position but without taking up space. Thus, if placed at the right end of a line, the box will extend into the margin. In this case, however, no mandatory argument specifies the box size: it is the same for all entries. Its value is the same as the space found to the right of all entries and can be set by the command `\contentsmargin` described below.

*A note on the examples in this section*

For the examples in this section we copied some parts of the original `.toc` file generated by L<sup>A</sup>T<sub>E</sub>X for this book (Chapter 2 and parts of Chapter 3) into the file `partial.toc`. Inside the examples we then loaded this file with `\input` and manually added `\contentsfinish`. Of course, in a real document you would use the command `\tableofcontents` instead, so that the `.toc` file for *your* document is loaded and processed.

In our first example we provide a new formatting for chapter entries, while keeping the formatting for the section entries as defined by the standard L<sup>A</sup>T<sub>E</sub>X document class. The chapter entries are now set ragged right (`\filright`) in bold typeface, get one pica space above, followed by a thick rule. The actual entry is indented by six picas. In that space we typeset the word “Chapter” in small caps followed by a space and the chapter number (`\thecontentslabel`) using the `\contentslabel` directive with its optional argument. There is no special handling for entries without numbers, so they would be formatted with an indenta-

tion of six picas. We fill the remaining space using `\hfill` and typeset the page number in the margin via `\contentspage`. Finally, after the entry we add another two points of space so that the entry is slightly separated from any section entry following.

<b>CHAPTER 2      The Structure of a L<sup>A</sup>T<sub>E</sub>X Document</b>	<b>15</b>	<code>\usepackage{dotinlabels}\title{toc}</code>
2.1. The structure of a source file . . . . .	15	<code>\titlecontents{chapter}{[6pc]</code>
2.2. Sectioning commands . . . . .	22	<code>{\addvspace{1pc}}\bfseries</code>
2.3. Table of contents structures . . . . .	45	<code>\titlerule[2pt]\filright}</code>
2.4. Managing references . . . . .	66	<code>{\contentslabel</code>
		<code>{[\textsc{\chaptername}]\thecontentslabel}{[6pc]}</code>
		<code>}</code> <code>{\hfill\contentspage}</code>
<b>CHAPTER 3      Basic Formatting Tools</b>	<b>9</b>	<code>[{\addvspace{2pt}}]</code>
3.1. Phrases and paragraphs . . . . .	80	<code>% Show only chapter/section entries:</code>
3.2. Footnotes, endnotes, and marginals . . . . .	109	<code>\setcounter{tocdepth}{1}</code>
3.3. List structures . . . . .	128	<code>\input{partial.toc}</code>
3.4. Simulating typed text . . . . .	151	<code>\contentsfinish</code>

Instead of indenting the whole entry and then moving some material into the left margin using `\contentslabel`, you can make use of `\contentspush` to achieve a similar effect.

`\contentspush{text}`

This command typesets *text* and then increases the *left-indent* by the width of *text* for all additional lines of the entry (if any). As a consequence, the indentation will vary if the width of the *text* changes. In many cases such variation is not desirable, but in some cases other solutions give even worse results. Consider the case of a document with many chapters, each containing dozens of sections. A rigid *left-indent* needs to be able to hold the widest number, which may have five or six digits. In that case a label like “1.1” will come out unduly separated from its entry text. Given below is a solution that grows with the size of the entry number.

		<code>\usepackage{title{toc}}</code>
		<code>\titlecontents{section}{[0pt]{\addvspace{2pt}}\filright}</code>
		<code>{\contentspush{\thecontentslabel}\ }}</code>
		<code>{\titlerule*[8pt]{.}\contentspage}</code>
12.8 Some section that is wrapped in the TOC . . .	87	<code>\contentsline{section}{\numberline{12.8}Some section that is wrapped in the TOC}{87}</code>
12.9 Another section . . . . .	88	<code>\contentsline{section}{\numberline{12.9}Another section}{88}</code>
12.10 And yet another wrapping section . . . . .	90	<code>\contentsline{section}{\numberline{12.10}And yet another wrapping section}{90}</code>
2-3-10 12.11 Final section. . . . .	92	<code>\contentsline{section}{\numberline{12.11}Final section}{92}</code>
		<code>\contentsfinish</code>

---

**\contentsmargin[*correction*] {*right-sep*}**

The right margin for all entries can be set to *right-sep* using the `\contentsmargin` declaration. The default value for this margin is `\@pnumwidth`, which is set by the standard classes to be wide enough to contain up to three digits. The optional *correction* argument will be added to all lines of an entry except the last. This argument can, for example, be used to fine-tune the contents layout, so that dots from a row of leaders align with the text of previous lines in a multiple-line entry.

### Contents entries combined in a paragraph

Standard L<sup>A</sup>T<sub>E</sub>X only supports contents entries formatted on individual lines. In some cases, however, it is more economical to format lower-level entries together in a single paragraph. With the `titletoc` package this becomes possible.

```
\titlecontents*{type} [left-indent] [{before-code}]{numbered-entry-format}
    [{numberless-entry-format}]{page-format} [mid-code]
\titlecontents*{type}... {page-format} [mid-code] [final-code]
\titlecontents*{type}... {page-format} [start-code] [mid-code] [final-code]
```

The `\titlecontents*` declaration is used for entries that should be formatted together with other entries of the same or lower level in a single paragraph. The first six arguments are identical to those of `\titlecontents` described on page 59. But instead of a vertically oriented *below-code* argument, `\titlecontents*` provides one to three optional arguments that handle different situations that can happen when entries are about to be joined horizontally. All three optional arguments are by default empty. The joining works recursively as follows:

- If the current entry is the first entry to participate in joining, then its *start-code* is executed before typesetting the entry.
- Otherwise, there has been a previous entry already participating.
  - If both entries are on the same level, then the *mid-code* is inserted.
  - Otherwise, if the current entry is of a lower level, then the *start-code* for it is inserted and we recur.
  - Otherwise, the current entry is of a higher level. First, we execute for each level that has ended the *final-code* (in reverse order). Then, if the current entry is not participating in joining, we are done. Otherwise, the *mid-code* for the entry is executed, as a previous entry of the same level should already be present (assuming a hierarchically structured document).

If several levels are to be joined, then you have to specify any paragraph layout information in the *before-code* of the highest level participating. Otherwise, the scope of your settings will not include the paragraph end and thus will not be applied. In the following example, `\footnotesize` applies only to the section entries—the `\baselineskip` for the whole paragraph is still set in `\normalsize`.

This artificial example shows how one can join two different levels using the three optional arguments. Note in particular the spaces added at the beginning of some arguments to get the right result when joining.

```

\usepackage{titletoc}
\contentsmargin{0pt}
\titlecontents*{chapter}[Opt]{\sffamily}
    {}{}{}, \thecontentspage}[\textbullet\ ] [~\p]
\titlecontents*{section}[Opt]{\footnotesize\slshape}
    {}{}{}[ \{}[; ]\}]
\contentsline{chapter}{\numberline{1}A first}{1}
\contentsline{chapter}{\numberline{2}A second}{4}
\contentsline{section}{\numberline{2.1}sec-A}{5}
\contentsline{section}{\numberline{2.2}sec-B}{6}
\contentsline{chapter}{\numberline{3}A third}{8}
\contentsline{section}{\numberline{3.1}sec-C}{8}
\contentsfinish

```

A first, 1 • A second, 4 {sec-A; sec-B} •  
A third, 8 {sec-C} ¶

2-3-11

Let us now see how this works in practice. In the next example we join the section level, separating entries by a bullet surrounded by some stretchable space (\xquad) and finishing the list with a period. The chapter entries are interesting as well, because we move the page number to the left. Both types omit the heading numbers completely in this design. As there are no page numbers at the right, we also set the right margin to zero.

15

# The Structure of a L<sup>A</sup>T<sub>E</sub>X Document

- The structure of a source file, 15
- Sectioning commands, 22
- Table of contents structures, 45
- Managing references, 66.
- References, 66.

79

## Basic Formatting Tools

- Phrases and paragraphs, 80 • Footnotes, endnotes, and marginals, 109 • List structures, 128 • Simulating typed text, 151

2-3-12

```

\usepackage{titletoc}
\contentsmargin{0pt}
\titlecontents{chapter}[0pt]
    {\addvspace{1.4pc}\bfseries}
    {{\Huge \thecontentspage}\quad}{}{{}}
\newcommand\xquad
    {\hspace{1em plus .4em minus .4em}}
\titlecontents*{section}[0pt]
    {\filright\small}{}{{},~\thecontentspage}
    [\xquad\textbullet\xquad][.]
\setcounter{tocdepth}{1}
\input{partial.toc}\contentsfinish

```

As a second example we look at a set-up implementing a layout close to the one used in *Methods of Book Design* [170]. This design uses non-lining digits, something we achieve by using the `eco` package. The `\chapter` titles are set in small capitals. To arrange that we use `\scshape` and turn all letters in the title to lower-case using `\MakeLowercase` (remember that the last token of the *numbered-entry-format* and the *numberless-entry-format* arguments can be a command with one argument to receive the heading text). The sections are all run together in a paragraph with the section number getting a `\S` sign prepended. Separation between

entries is a period followed by a space, and the final section is finished with a period as well.

## 2 THE STRUCTURE OF A L<sup>A</sup>T<sub>E</sub>X DOCUMENT 15

§2.1 The structure of a source file, 15. §2.2 Sectioning commands, 22. §2.3 Table of contents structures, 45. §2.4 Managing references, 66.

## 3 BASIC FORMATTING TOOLS

79

§3.1 Phrases and paragraphs, 80. §3.2 Footnotes, endnotes, and marginals, 109. §3.3 List structures, 128. §3.4 Simulating typed text, 151.

```
\usepackage{eco,titletoc}
\contentsmargin{0pt}
\titlecontents{chapter}[1.5pc]
  {\addvspace{2pc}\large}
  {\contentslabel{2pc}%
    \scshape\MakeLowercase}
  {\scshape\MakeLowercase}
  {\hfill\thecontentspage}
  [\vspace{2pt}]
\titlecontents*{section}[1.5pc]
  {\small{\S\thecontentslabel\ }}{. }{. }
  {,\~\thecontentspage}{. }{. }
\setcounter{tocdepth}{1}
\input{partial.toc}
\contentsfinish
```

2-3-13

### Generating partial table of contents lists

It is possible to generate partial contents lists using the `titletoc` package; it provides four commands for this purpose.

`\startcontents[name]`

A partial table of contents is started with `\startcontents`. It is possible to collect data for several partial TOCs in parallel, such as one for the current `\part` as well as one for the current `\chapter`. In that case the optional *name* argument allows us to distinguish between the two (its default value is the string `default`). Concurrently running partial TOCs are allowed to overlap each other, although normally they will be nested. All information about these partial TOCs is stored in a single file with the extension `.ptc`; this file is generated once a single `\startcontents` command is executed.

`\printcontents[name][prefix]{start-level}{toc-code}`

This command prints the current partial TOC started earlier by `\startcontents`; if the optional *name* argument is used, then a partial contents list with that *name* must have been started.<sup>1</sup>

It is quite likely that you want to format the partial TOC differently from the main table of contents. To allow for this the *prefix* argument is prepended to any entry type when looking for a layout definition provided via `\titlecontents` or its starred form. In the example below we used `p-` as the *prefix* and then defined a formatting for `p-subsection` to format `\subsection` entries in the partial TOC.

<sup>1</sup>The package is currently (as of 2003) quite unforgiving if you try to print a contents list without first starting it—you will receive an unspecific low-level TeX error.

The *start-level* argument defines the first level that is shown in the partial TOC; in the example we used the value 2 to indicate that we want to see all subsections and lower levels.

The depth to which we want to include entries in the partial TOC can be set in *toc-code* by setting the `tocdepth` to a suitable value. Other initializations for typesetting the partial TOC can be made there as well. In the example we cancel any right margin, because the partial TOC is formated as a single paragraph.

Integrating partial TOCs in the heading definitions so that there is no need to change the actual document is very easy when `titletoc` is used together with the `titlesec` package. Below we extend Example 2-2-18 from page 40 so that the `\section` command now automatically prints a partial TOC of all its subsections. This is done by using the optional *after-code* argument of the `\titleformat` declaration. We first add some vertical space, thereby ensuring that no page break can happen at this point. We next (re)start the default partial TOC with `\startcontents`. We then immediately typeset it using `\printcontents`; its arguments have been explained above. Finally, we set up a formatting for subsections in a partial TOC using `\titlecontents*` to run them together in a justified paragraph whose last line is centered (`\filllast`). Stringing this all together gives the desired output without any modification to the document source. Of course, a real design would also change the look and feel of the subsection headings in the document to better fit those of the sections.

	<pre>\usepackage{titlesec,titletoc} \titleformat{\section}[frame]{\normalfont}   {\footnotesize \enspace SECTION \thesection    \enspace}{6pt}{\large\bfseries\filcenter}   [\vspace*{5pt}\startcontents    \printcontents{p-}{2}{\contentsmargin{0pt}}]   \titlespacing*\{\section}{1pc}{*4}{*2.3}{1pc}   \titlecontents*[p-subsection][0pt]     {\small\itshape\filllast}{\hskip 1em}{\hskip 1em}[\hskip 1em][]{\hskip 1em}[].   \section{A Title Test}   Some text to prove that this paragraph is not indented.   \subsection{A first} Some text \ldots \newpage   \subsection{A longer second} Some more text.   \stopcontents \subsection{A third} \resumecontents   \subsection{An even longer fourth}</pre>
SECTION 1	
<b>A Title Test</b>	

*A first — A longer second — An even longer fourth.*

Some text to prove that this paragraph is not indented.

## 1.1 A first

Some text ...

If necessary, one can temporarily (or permanently) stop collecting entries for a partial TOC. We made use of this feature in the previous example by suppressing the third subsection.

`\stopcontents[name]            \resumecontents[name]`

The `\stopcontents` command stops the entry collection for the `default` partial TOC or, if used with the *name* argument, for the TOC with that *name*. At a

later point the collection can be restarted using `\resumecontents`. Note that this is quite different from calling `\startcontents`, which starts a *new* partial TOC, thereby making the old entries inaccessible.

## 2.4 Managing references

L<sup>A</sup>T<sub>E</sub>X has commands that make it easy to manage references in a document. In particular, it supports *cross-references* (internal references between elements within a document), *bibliographic* citations (references to external documents), and *indexing* of selected words or expressions. Indexing facilities will be discussed in Chapter 11, and bibliographic citations in Chapters 12 and 13.

To allow cross-referencing of elements inside a document, you should assign a “key” (consisting of a string of ASCII letters, digits, and punctuation) to the given structural element and then use that key to refer to that element elsewhere.

```
\label{key} \ref{key} \pageref{key}
```

The `\label` command assigns the *key* to the currently “active” element of the document (see below for determining which element is active at a given point). The `\ref` command typesets a string, identifying the given element—such as the section, equation, or figure number—depending on the type of structural element that was active when the `\label` command was issued. The `\pageref` command typesets the number of the page where the `\label` command was given. The *key* strings should, of course, be unique. As a simple aid it can be useful to prefix them with a string identifying the structural element in question: `sec` might represent sectional units, `fig` would identify figures, and so on.

## 4 A Section

A reference to this section looks like this: “see section 4 on page 6”.

A reference to this section looks like this: “see section-\ref{sec:this} on page-\pageref{sec:this}”.

2-4-1

*Restrictions on the characters used in keys*

There is a potential danger when using punctuation characters such as a colon. In certain language styles within the `babel` system (see Chapter 9), some of these characters have special meanings and behave essentially like commands. The `babel` package tries hard to allow such characters as part of `\label` keys but this can fail in some situations. Similarly, characters outside the ASCII range, made available through packages such as `inputenc`, are not officially supported in such keys and are likely to produce errors if used.

For building cross-reference labels, the “currently active” structural element of a document is determined in the following way. The sectioning commands (`\chapter`, `\section`, ...), the environments `equation`, `figure`, `table`, and the `theorem` family, as well as the various levels of the `enumerate` environment, and

\footnote set the *current reference string*, which contains the number generated by L<sup>A</sup>T<sub>E</sub>X for the given element. This reference string is usually set at the beginning of an element and reset when the scope of the element is exited.

Notable exceptions to this rule are the `table` and `figure` environments, where the reference string is defined by the `\caption` commands. This allows several `\caption` and `\label` pairs inside one environment.<sup>1</sup> As it is the `\caption` directive that generates the number, the corresponding `\label` command must follow the `\caption` command in question. Otherwise, an incorrect number will be generated. If placed earlier in the float body, the `\label` command will pick up the *current reference string* from some earlier entity, typically the current sectional unit.

 Problems with  
wrong references  
on floats

The problem is shown clearly in the following example, where only the labels “fig:in2” and “fig:in3” are placed correctly to generate the needed reference numbers for the figures. In the case of “fig:in4” it is seen that environments (in this case, `center`) limit the scope of references, since we obtain the number of the current section, rather than the number of the figure.

### 3 A section

#### 3.1 A subsection

Text before is referenced as ‘3.1’.

... figure body ...

Figure 1: First caption

... figure body ...

Figure 2: Second caption

The labels are: ‘before’ (3.1), ‘fig:in1’ (3.1),  
 ‘fig:in2’ (1), ‘fig:in3’ (2), ‘fig:in4’ (3.1),  
 ‘after’ (3.1).

2-4-2

```
\section{A section}
\subsection{A subsection}\label{sec:before}
Text before is referenced as ‘\ref{sec:before}’.
```

```
\begin{figure}[ht] \label{fig:in1}
\begin{center}
\fbox{\ldots} figure body \ldots
\caption{First caption} \label{fig:in2}
\bigskip
\fbox{\ldots} figure body \ldots
\caption{Second caption} \label{fig:in3}
\end{center}
\end{figure} \label{fig:in4}
\label{sec:after}
```

```
\raggedright
The labels are: ‘before’ (\ref{sec:before}),  

‘fig:in1’ (\ref{fig:in1}), ‘fig:in2’  

(\ref{fig:in2}), ‘fig:in3’ (\ref{fig:in3}),  

‘fig:in4’ (\ref{fig:in4}), ‘after’  

(\ref{sec:after}).
```

For each `key` declared with `\label{key}`, L<sup>A</sup>T<sub>E</sub>X records the current reference string and the page number. Thus, multiple `\label` commands (with different key identifiers `key`) inside the same sectional unit will generate an identical reference string but, possibly, different page numbers.

<sup>1</sup>There are, however, good reasons for not placing more than one `\caption` command within a float environment. Typically proper spacing is difficult to achieve and, more importantly, future versions of L<sup>A</sup>T<sub>E</sub>X might make this syntax invalid.

### 2.4.1 showkeys—Displaying the reference keys

When writing a larger document many people print intermediate drafts. With such drafts it would be helpful if the positions of `\label` commands as well as their keys could be made visible. This becomes possible with the `showkeys` package written by David Carlisle.

When this package is loaded, the commands `\label`, `\ref`, `\pageref`, `\cite`, and `\bibitem` are modified in a way that the used key is printed. The `\label` and `\bibitem` commands normally cause the key to appear in a box in the margin, while the commands referencing a key print it in small type above the formatted reference (possibly overprinting some text). The package tries hard to position the keys in such a way that the rest of the document's formatting is kept unchanged. There is, however, no guarantee for this, and it is best to remove or disable the `showkeys` package before attempting final formatting of the document.

## 1 An example

`sec`

Section <sup>sec</sup>I shows the use of the `showkeys` package with a reference to equation (I).

$$a = b$$

(I) eq

```
\usepackage{showkeys}
\section{An example}\label{sec}
Section~\ref{sec} shows the use of the
\texttt{showkeys} package with a
reference to equation~(\ref{eq}).
\begin{equation}
a = b \label{eq}
\end{equation}
```

2-4-3

The package supports the `fleqn` option of the standard classes and works together with the packages of the *AMS-L<sup>A</sup>T<sub>E</sub>X* collection, `variorref`, `natbib`, and many other packages. Nevertheless, it is nearly impossible to ensure its safe working with all packages that hook into the reference mechanisms.

If you want to see only the keys on the `\label` command in the margin, you can suppress the others by using the package option `notref` (which disables the redefinition of `\ref`, `\pageref`, and related commands) or the option `notcite` (which does the same for `\cite` and its cousins from the `natbib` and `harvard` packages). Alternatively, you might want to use the option `color` to make the labels less obtrusive.

Finally, the package supports the options `draft` (default) and `final`. While the latter is useless when used on the package level, because you can achieve the same result by not specifying the `showkeys` package, it comes in handy if `final` is specified as a global option on the class.

### 2.4.2 variorref—More flexible cross-references

In many cases it is helpful, when referring to a figure or table, to put both a `\ref` and a `\pageref` command into the document, especially when one or more pages

separate the reference and the object. Some people use a command like

```
\newcommand\fullref[1]{\ref{#1} on page~\pageref{#1}}
```

to reduce the number of keystrokes necessary to make a complete reference. But because one never knows with certainty where the referenced object finally falls, this method can result in a citation to the current page, which is disturbing and should therefore be avoided. The package `varioromam`, written by Frank Mittelbach, tries to solve that problem. It provides the commands `\vref` and `\vpageref` to deal with single references, as well as `\vrefrange` and `\vpagerefrange` to handle multiple references. In addition, its `\labelformat` declaration offers the ability to format references differently depending on the counter used in the reference.

`\vref*[key]`

The command `\vref` is like `\ref` when the reference and `\label` are on the same page. If the label and reference differ by one page, `\vref` creates one of these strings: “on the facing page”, “on the preceding page”, or “on the following page”. The word “facing” is used when both label and reference fall on a double spread. When the difference is larger than one page, `\vref` produces both `\ref` and `\pageref`. Note that when a special page numbering scheme is used instead of the usual Arabic numbering (for example, `\pagenumbering{roman}`), there will be no distinction between being one or many pages off.

There is one other difference between `\ref` and `\vref`: the latter removes any preceding space and inserts its own. In some cases, such as after an opening parenthesis, this is not desirable. In such cases, use `\vref*`, which acts like `\vref` but does not add any space before the generated text.

`\vpageref*[samepage] [otherpage] {key}`

Sometimes you may only want to refer to a page number. In that case, a reference should be suppressed if you are citing the current page. For this purpose the `\vpageref` command is defined. It produces the same strings as `\vref` except that it does not start with `\ref`, and it produces the string saved in `\reftextcurrent` if both label and reference fall on the same page.

Defining `\reftextcurrent` to produce something like “on this page” ensures that text like

```
... see the diagram \vpageref{ex:foo} which shows ...
```

does not come out as “... see the diagram which shows ...”, which could be misleading.

You can put a space in front of `\vpageref`; it will be ignored if the command does not create any text at all. If some text is added, an appropriate space is automatically placed in front of the text. The variant form `\vpageref*` removes

preceding white space before the generated text but does not reinsert its own. Use it if the space otherwise generated poses a problem.

In fact, `\vpageref` and `\vpageref*` allow even more control when used with their two optional arguments. The first argument specifies the text to be used if the label and reference fall on the same page. This is helpful when both are close together, so that they may or may not be separated by a page break. In such a case, you will usually know whether the reference comes before or after the label so that you can code something like the following:

```
... see the diagram \vpageref[above]{ex:foo} which shows ...
```

The resultant text will be “... see the diagram above which shows ...” when both are on the same page, or “... see the diagram on the page before which shows ...” (or something similar, depending on the settings of the `\reftext..before` and `\reftext..after` commands) if they are separated by a page break. Note, however, that if you use `\vpageref` with the optional argument to refer to a figure or table, depending on the float placement parameters, the float may show up at the top of the current page and therefore before the reference, even if it follows the reference in the source file.<sup>1</sup>

Maybe you even prefer to say “... see the above diagram” when both diagram and reference fall on the same page—that is, reverse the word order compared to our previous example. In fact, in some languages the word order automatically changes in that case. To allow for this variation the second optional argument `otherpage` can be used. It specifies the text preceding the generated reference if both object and reference do not fall on the same page. Thus, one would write

```
... see the \vpageref[above diagram][diagram]{ex:foo} which shows ...
```

to achieve the desired effect.

The `amsmath` package provides a `\eqref` command to reference equations. It automatically places parentheses around the equation number. To utilize this, one could define

```
\newcommand\eqvref[1]{\eqref{\#1}\ \vpageref{\#1}}
```

to automatically add a page reference to it.

```
\vrefrange[here-text]{start-key}{end-key}
```

This command is similar to `\vref` but takes two mandatory arguments denoting a range of objects to refer to (e.g., a sequence of figures or a sequence of equations). It decides what to say depending on where the two labels are placed in

<sup>1</sup>To ensure that a floating object always follows its place in the source use the `fafter` package, which is described in Section 6.2.

relation to each other; it is essentially implemented using `\vpagerefrange` (described below). The optional argument that the command may take is the text to use in case both labels appear on the current page. Its default is the string stored in `\ref{textcurrent}`.

## 1 Test

Observe equations 1.1 to 1.3 on pages 6–7 and in particular equations 1.2 to 1.3 on the facing page.

$$a = b + c \quad (1.1)$$

2-4-4

6

Here is a second equation...

$$b = a + c \quad (1.2)$$

...and finally one more equation:

$$c = a + b \quad (1.3)$$

7

```
\usepackage{varioref}
\renewcommand{\theequation}{\thesubsection.\arabic{equation}}
\section{Test}
Observe equations~\vrefrange{A}{C} and
in particular equations~\vrefrange{B}{C}.
\begin{equation}
a=b+c\label{A} \end{equation}
Here is a second equation\ldots
\begin{equation}
b=a+c\label{B} \end{equation}
\ldots and finally one more equation:
\begin{equation}
c=a+b\label{C} \end{equation}
```

`\vpagerefrange*[here-text] {start-key}{end-key}`

This command is similar to `\vpageref` but takes two mandatory arguments—two labels denoting a range. If both labels fall on the same page, the command acts exactly like `\vpageref` (with a single label); otherwise, it produces something like “on pages 15–18” (see the customization possibilities described below). Like `\vrefrange` it has an optional argument that defaults to the string stored in `\ref{textcurrent}` and is used if both labels appear on the current page.

Again there exists a starred form, `\vpagerefrange*`, which removes preceding white space before the generated text without reinserting its own space.

A reference via `\ref` produces, by default, the data associated with the corresponding `\label` command (typically a number); any additional formatting must be provided by the user. If, for example, references to equations are always to be typeset as “equation (*number*)”, one has to code “`equation (\ref{key})`”.

Fancy labels

`\labelformat{counter}{formatting-code}`

With `\labelformat` the `varioref` package offers a possibility to generate such frills automatically.<sup>1</sup> The command takes two arguments: the name of a counter and its representation when referenced. Thus, for a successful usage, one has to know the counter name being used for generating the label, though in practice this should not pose a problem. The current counter number (or, more exactly, its representation) is picked up as an argument, so the second argument should contain #1.

<sup>1</sup>This command is also available separately with the `fncylab` package written by Robin Fairbairns.

A side effect of using `\labelformat` is that, depending on the defined formatting, it becomes impossible to use `\ref` at the beginning of a sentence (if its replacement text starts with a lowercase letter). To overcome this problem `varioref` introduces the commands `\Ref` and `\Vref` (including starred forms) that behave like `\ref` and `\vref` except that they uppercase the first token of the generated string. In the following example (which you should compare to Example 2-4-3 on page 68), you can observe this behavior when “section” is turned into “Section”.

## 1 An example

Section 1 shows the use of the `\labelformat` declaration with a reference to equation (1).

$$a = b \quad (1)$$

```
\usepackage{varioref}
\labelformat{section}{\textbf{section}\#1}
\labelformat{equation}{\textbf{equation}\#1}
\section{An example}\label{sec}
\Ref{sec} shows the use of the \verb=\labelformat= declaration with a reference to \ref{eq}.
\begin{equation} a = b \label{eq} \end{equation}
```

2-4-5

To make `\Ref` or `\Vref` work properly the first token in the second argument of `\labelformat` has to be a simple ASCII letter; otherwise, the capitalization will fail or, even worse, you will end up with some error messages. If you actually need something more complicated in this place (e.g., an accented letter), you have to explicitly surround it with braces, thereby identifying the part that needs to be capitalized. For example, for figure references in the Hungarian language you might want to write `\labelformat{figure}{\{'a\}bra-\thefigure}`.

As a second example of the use of `\labelformat` consider the following situation: in the `report` or `book` document class, footnotes are numbered per chapter. Referencing them would normally be ambiguous, given that it is not clear whether we refer to a footnote in the current chapter or to a footnote from a different chapter. This ambiguity can be resolved by always adding the chapter information in the reference, or by comparing the number of the chapter in which the `\label` occurred with the current chapter number and adding extra information if they differ. This is achieved by the following code:

```
\usepackage{ifthen,varioref}
\labelformat{footnote}{\#1\protect\iscurrentchapter{\thechapter}}
\newcommand\iscurrentchapter[1]{%
  \ifthenelse{\equal{\#1}{\thechapter}}{}{ in Chapter\#1}}
```

The trick is to use `\protect` to prevent `\iscurrentchapter` from being evaluated when the label is formed. Then when the `\ref` command is executed, `\iscurrentchapter` will compare its argument (i.e., the chapter number current when the label was formed) to the now current chapter number and, when they differ, typeset the appropriate information.

*Providing your own  
reference  
commands*

The package also provides the `\vrefpagenum` command, which allows you to write your own small commands that implement functions similar to those provided by the two previous commands. It takes two arguments: the second is a

label (i.e., as used in `\label` or `\ref`) and the first is an arbitrary command name (make sure you use your own) that receives the page number related to this label. Thus, if you have two (or more) labels, you could retrieve their page numbers, compare them, and then decide what to print.

The next example shows a not very serious application that compares two equation labels and prints out text depending on their relative positions. Compare the results of the tests on the first page with those on the second.

Test: the equations 1 and 2 on this page  
 Test: the equation 1 on the current page and 3 on page 8

$$a = b + c \quad (1)$$

$$b = a + c \quad (2)$$

2-4-6

6

Test: the equations 1 and 2 on the preceding page  
 Test: the equation 1 on the facing page and 3 on the next page

.

7

```
\usepackage{varioref,ifthen}
\newcommand{\veqns}[2]{%
  \vrefpagenum{\firstnum{#1}}%
  \vrefpagenum{\secondnum{#2}}%
  the equation%
  \ifthenelse
    {\equal{\firstnum}{\secondnum}}%
    {s \ref{#1}}%
    { \ref{#1}\vpageref{#1}}%
  \space and \ref{#2}\vpageref{#2}%
}

Test: \veqns{A}{B} \par Test: \veqns{A}{C}
\begin{equation} a=b+c \label{A} \end{equation}
\begin{equation} b=a+c \label{B} \end{equation}
\newpage
Test: \veqns{A}{B} \par Test: \veqns{A}{C}
\newpage
\begin{equation} c=a+b \label{C} \end{equation}
```

The package supports the options defined by the `babel` system (see Section 9.1.3); thus a declaration like `\usepackage[german]{varioref}` will produce texts suitable for the German language. In addition, the package supports the options `final` (default) and `draft`; the latter changes certain error messages (described on page 75) into warnings. This ability can be useful during the development of a document.

To allow further customization, the generated text strings (which will be predefined by the language options) are all defined via macros. Backward references use `\ref{textbefore}` if the label is on the preceding page but invisible, and `\ref{textfacebefore}` if it is on the facing page (that is, if the current page number is odd).

Similarly, `\ref{textafter}` is used when the label comes on the next page but one has to turn the page, and `\ref{textfaceafter}` when it is on the next, but facing, page. These four strings can be redefined with `\renewcommand`.

The command `\ref{textfaraway}` is used when the label and reference differ by more than one page, or when they are non-numeric. This macro is a bit different from the preceding ones because it takes one argument, the symbolic reference string, so that you can make use of `\pageref` in its replacement text. For instance,

*Package options**Individual customization*

if you wanted to use your macros in German language documents, you would define something like:

```
\renewcommand\ref{textfaraway}[1]{auf Seite~\pageref{#1}}
```

The `\ref{textpage range}` command takes two arguments and produces the text that describes a page range (the arguments are keys to be used with `\pageref`). See below for the English language default.

Similarly, `\ref{textlabel range}` takes two arguments and describes the range of figures, tables, or whatever the labels refer to. The default for English is “`\ref{#1} to~\ref{#2}`”.

To allow some random variation in the generated strings, you can use the command `\ref{textvario` inside the string macros. This command takes two arguments and selects one or the other for printing depending on the number of `\vref` or `\vpageref` commands already encountered in the document.

The default definitions of the various macros described in this section are shown below:

```
\newcommand\ref{textfaceafter
  {on the \ref{textvario{facing}{next} page}
\newcommand\ref{textfacebefore
  {on the \ref{textvario{facing}{preceding} page}
\newcommand\ref{textafter
  {on the \ref{textvario{following}{next} page}
\newcommand\ref{textbefore
  {on the \ref{textvario{preceding page}{page before}}
\newcommand\ref{textcurrent
  {on \ref{textvario{this}{the current} page}
\newcommand\ref{textfaraway [1]{on page~\pageref{#1}}
\newcommand\ref{textpage range [2]{on pages~\pageref{#1}--\pageref{#2}}
\newcommand\ref{textlabelrange[2]{\ref{#1} to~\ref{#2}}}
```

If you want to customize the package according to your own preferences, just write appropriate redefinitions of the above commands in a file with the extension `.sty` (e.g., `vrflocal.sty`). If you also put `\RequirePackage{varioref}` (see Section A.4 on page 877) at the beginning of this file, then your local package will automatically load the `varioref` package. If you use the `babel` system, redefinitions for individual languages should be added using `\addto`, as explained in Section 9.5.

Some people do not like textual references to pages but want to automatically suppress a page reference when both label and reference fall on the same page. This can be achieved with the help of the `\thepagerefnum` command as follows:

```
\renewcommand\ref{textfaceafter {on page~\thepagerefnum}
\renewcommand\ref{textfacebefore{on page~\thepagerefnum}
\renewcommand\ref{textafter      {on page~\thepagerefnum}
\renewcommand\ref{textbefore    {on page~\thepagerefnum}
```

Within one of the `\ref{text}...` commands, `\the\pageref{num}` evaluates to the current page number if known, or to two question marks otherwise.

Defining commands, like the ones described above, poses some interesting problems. Suppose, for example, that a generated text like “on the next page” gets broken across pages. If this happens, it is very difficult to find an acceptable algorithmic solution and, in fact, this situation can even result in a document that will always change from one state to another (i.e., inserting one string; finding that this is wrong; inserting another string on the next run which makes the first string correct again; inserting ...). The current implementation of the package `varioref` considers the end of the generated string as being relevant. For example,

Table 5 on the current (*page break*) page

would be true if Table 5 were on the page containing the word “page”, not the one containing the word “current”. However, this behavior is not completely satisfactory, and in some cases may actually result in a possible loop (where  $\text{\LaTeX}$  is requesting an additional run over and over again). Therefore, all such situations will produce a  $\text{\LaTeX}$  error message, so that you can inspect the problem and perhaps decide to use a `\ref` command in that place.

Also, be aware of the potential problems that can result from using `\ref{text}vario`: if you reference the same object several times in nearby places, the change in wording every second time will look strange.

A final warning: every use of `\vref` or `\vpageref` internally generates two macro names. As a result, you may run out of name space or main memory if you make heavy use of this command on a small  $\text{\TeX}$  installation. For this reason the command `\fullref` is also provided. It can be used whenever you are sure that both label and reference cannot fall on nearby pages.

### 2.4.3 prettyref—Adding frills to references

One problem with  $\text{\LaTeX}$ ’s cross-referencing mechanism is that it only produces the element number (or the page number) but leaves the surrounding formatting as the responsibility of the author. This means that uniform references are difficult to achieve. For example, if the publisher’s house style requires that figures be referenced as “Fig.xx” one has to manually go through the source document and change all references to that format.

The `prettyref` package written by Kevin Ruland provides automatic support for such additional formatting strings, provided the *keys* used on the `\label` commands obey a certain structure. They have to be of the form “`<prefix>:<name>`” with neither *prefix* nor *name* containing a colon (e.g., `fig:main`), a form used by many people anyway. The extra formatting strings are produced when using the command `\prettyref`; standard `\ref` and `\pageref` are not affected by the package. Note that this is different from the `\labelformat` declaration, as provided by `varioref`, which changes the display of the reference labels in all circumstances.

*A few warnings*

```
\newrefformat{prefix}{code}
```

This command defines the formatting for references having the *prefix* as the prefix in their key. The *code* argument uses #1 to refer to the key used so that it can be passed to \ref, \vref, and so on. This format can be accessed when using the key with the command \prettyref.

## 4 A Section

A reference to the equation in this section looks like: “see (1) in Section 4”.

```
\usepackage{prettyref}
\newrefformat{sec}{Section~\ref{#1}}
\section{A Section}\label{sec:this}
A reference to the equation in this section looks like:
‘‘see \prettyref{eq:a} in \prettyref{sec:this}’’.
`           (1) \begin{equation} a = b \label{eq:a} \end{equation}
```

2-4-7

The example shows that the prettyref package has formatting for the *prefix* “eq” already built in. In fact, it knows about several other predefined formats, but since most of them allow breaking between the generated text and the number you should probably define your own.

Because this package does not make any distinction between references used at the beginning of a sentence and references used in mid-sentence, it may not be usable in all circumstances. It is also impossible to replace the colon that separates *prefix* and *name*, which means that it cannot be combined with some language packages that use the colon in special ways. In that case you might consider using the fancyref package written by Axel Reichert, which provides a similar functionality but internally uses a much more complex set-up.

### 2.4.4 titleref—Non-numerical references

In some documents it is required to reference sections by displaying their title texts instead of their numbers, either because there is no number to refer to or because the house style asks for it. This functionality is available through the titleref package written by Donald Arseneau, which provides the command \titleref to cross-reference the titles of sections and float captions.

*Unnumbered sections get ↗ moving arguments*

For numbered sections and floats with captions, the titles are those that would be displayed in the contents lists (regardless of whether such a list is actually printed). That is, if a short title is provided via the optional argument of a sectioning command or caption, then this title is printed by \titleref. Unnumbered sections take their title reference from the printed title. As a consequence, the arguments of unnumbered sectioning commands are turned into moving arguments, which will cause weird errors if they contain un\protected fragile commands.

A \titleref to a label unrelated to a title (e.g., a label in a footnote, or an enumeration item) will simply pick up any earlier title, typically the one from the surrounding section.

As shown in the next example, the title of the current section is available through `\currenttitle`, independently of whether it was associated with a `\label` key. The example also shows that `\titleref` and `\ref` can coexist.

## 1 Textual References

In section “Textual References” we prove that it is possible to reference unnumbered sections by referencing section “Example”.

### A Small Example

[2-4-8] The current section is referenced in section 1.

```
\usepackage{titleref}
\setcounter{secnumdepth}{1}
\section{Textual References}\label{num}
In section ``\currenttitle{}'' we prove that
it is possible to reference unnumbered sections
by referencing section ``\titleref{ex}''.
```

```
\subsection[Example]{A Small Example}\label{ex}
The current section is referenced in
section~\ref{num}.
```

The format of the title reference can be controlled by redefining the command `\theTitleReference`. It takes two arguments: a *number* as it would be displayed by `\ref`, and a *title*. If a document contains references to unnumbered titles, the *number* argument should not be used in the replacement text as it will contain an arbitrary number. For instance, the `\titleref` command in the next example displays “1”, even though the reference is to an unnumbered section.

## 1 Textual References

In section 1 *Textual References* we prove that it is possible to reference unnumbered sections by referencing section 1 *Example*.

### A Small Example

[2-4-9] The current section is referenced in section 1.

```
\usepackage{titleref}
\renewcommand{\theTitleReference}[2]{\emph{\#1 \ #2}}
\setcounter{secnumdepth}{1}
\section{Textual References}\label{num}
In section \currenttitle{} we prove that
it is possible to reference unnumbered
sections by referencing section \titleref{ex}.
```

```
\subsection[Example]{A Small Example}\label{ex}
The current section is referenced in
section~\ref{num}.
```

By default, the package works by inserting additional code into commands that are typically used to build headings, captions, and other elements. If combined with other packages that provide their own methods for typesetting titles, it might create conflicts. In that case you can tell the package to use a completely different approach by specifying the option `usetoc`. As the name implies, it directs the package to record the titles from the data written to the contents lists by redefining `\addcontentsline`. A consequence of this approach is that the `\label` command is not allowed within the *title* argument but has to follow it. In addition, no unrelated `\addcontentsline` command is allowed to intervene between heading and label. As starred sectioning commands do not generate contents entries, they are still redefined. This can be prevented by additionally specifying the option `nostar`, although then one can no longer refer to their titles.

Conflicts with other packages

### 2.4.5 hyperref—Active references

Sebastian Rahtz (with contributions by Heiko Oberdiek and David Carlisle) has developed the package `hyperref`, which makes it possible to turn all cross-references (citations, table of contents, and so on) into hypertext links. It works by extending the existing commands with functionality to produce \special commands that suitably equipped drivers can use to turn the references into hypertext links. The package is described in detail in [56, pp.35–67] and comes with its own manual, which itself contains hypertext links produced using the package.

The usage of `hyperref` can be quite easy. Just including it in your list of loaded packages (as the *last* package) suffices to turn all cross-references in your document into hypertext links. The package has a number of options to change the way the hypertext links look or work. The most important options are `colorlinks`, which makes the text of the link come out in color instead of with a box around it, and `backref`, which inserts links in the bibliography pointing to the place where an entry was cited.

The package offers a number of ways to influence the behavior of the PDF file produced from your document as well as ways to influence the behavior of the PDF viewer, such as the Adobe Reader.

### 2.4.6 xr—References to external documents

David Carlisle, building on earlier work of Jean-Pierre Drucbert, developed a package called `xr`, which implements a system for external references.

If, for instance, a document needs to refer to sections of another document—say, `other.tex`—then you can specify the `xr` package in the main file and give the command `\externaldocument{other}` in the preamble. Then you can use `\ref` and `\pageref` to refer to anything that has been defined with a `\label` command in either `other.tex` or your main document. You may declare any number of such external documents.

If any of the external documents or the main document uses the same `\label` key, then a conflict will occur because the key will have been multiply defined. To overcome this problem, `\externaldocument` takes an optional argument. If you declare `\externaldocument[A-]{other}`, then all references from the file `other.tex` are prefixed by `A-`. So, for instance, if a section in the file `other.tex` had a `\label{intro}`, then it could be referenced with `\ref{A-intro}`. The prefix need not be `A-`; it can be any string chosen to ensure that all the labels imported from external files are unique.

Note, however, that if one of the packages you are using declares certain active characters (e.g., `:` in French or `"` in German), then these characters should not be used inside `\label` commands. Similarly, you should not use them in the optional argument to `\externaldocument`.

The package does not work together with the `hyperref` package because both modify the internal reference mechanism. Instead, you can use the `xr-hyper` package, which is a reimplementation tailored to work with `hyperref`.

## C H A P T E R 3

# Basic Formatting Tools

The way information is presented visually can influence, to a large extent, the message as it is understood by the reader. Therefore, it is important that you use the best possible tools available to convey the precise meaning of your words. It must, however, be emphasized that visual presentation forms should aid the reader in understanding the text, and should not distract his or her attention. For this reason, visual consistency and uniform conventions for the visual clues are a must, and the way given structural elements are highlighted should be the same throughout a document. This constraint is most easily implemented by defining a specific command or environment for each document element that has to be treated specially and by grouping these commands and environments in a package file or in the document preamble. By using exclusively these commands, you can be sure of a consistent presentation form.

This chapter explains various ways for highlighting parts of a document. The first part looks at how short text fragments or paragraphs can be made to stand out and describes tools to manipulate such elements.

The second part deals with the different kind of “notes”, such as footnotes, marginal notes, and endnotes, and explains how they can be customized to conform to different styles, if necessary.

Typesetting lists is the subject of the third part. First, the various parameters and commands controlling the standard L<sup>A</sup>T<sub>E</sub>X lists, `enumerate`, `itemize`, and `description`, are discussed. Then, the extensions provided by the `paralist` package and the concept of “headed lists” exemplified by the `amsthm` package are presented. These will probably satisfy the structure and layout requirements of most readers. If not, then the remainder of this part introduces the generic `list`

environment and explains how to build custom layouts by varying the values of the parameters controlling it.

The fourth part explains how to simulate “verbatim” text. In particular, we have a detailed look at the powerful packages `fancyvrb` and `listings`.

The final part presents packages that deal with line numbering, handling of columns, such as parallel text in two columns, or solving the problem of producing multiple columns.

## 3.1 Phrases and paragraphs

In this section we deal with small text fragments and explain how they can be manipulated and highlighted in a consistent manner by giving them a visual appearance different from the one used for the main text.

We start by discussing how to define commands that take care of the space after them, then show a way to produce professional-looking marks of omission.

For highlighting text you can customize the font shape, weight, or size (see Section 7.3.1 on page 338). Text can also be underlined, or the spacing between letters can be varied. Ways for performing such operations are offered by the four packages `relsize`, `textcase`, `ulem`, and `soul`.

The remainder of this section then turns to paragraph-related issues, such as producing large initial letters at the start of a paragraph, modifying paragraph justification, altering the vertical spacing between lines of a paragraph, and introducing rectangular holes into it, that can be filled with small pictures, among other things.

### 3.1.1 `xspace`—Gentle spacing after a macro

The small package `xspace` (by David Carlisle) defines the `\xspace` command, for use at the end of macros that produce text. It adds a space unless the macro is followed by certain punctuation characters.

The `\xspace` command saves you from having to type `\_` or `{}` after most occurrences of a macro name in text. However, if either of these constructs follows `\xspace`, a space is not added by `\xspace`. This means that it is safe to add `\xspace` to the end of an existing macro without making too many changes in your document. Possible candidates for `\xspace` are commands for abbreviations such as “e.g.” and “i.e.”.

```
\newcommand\eg{e.g.\xspace}
\newcommand\ie{i.e.\xspace}
\newcommand\etc{etc.\@\xspace}
```

Notice the use of the `\@` command to generate the correct kind of space. If used to the right of a punctuation character, it prevents extra space from being added: the

dot will not be regarded as an end-of-sentence symbol. Using it on the left forces  $\text{\LaTeX}$  to interpret the dot as an end-of-sentence symbol.

Sometimes  $\text{\xspace}$  may make a wrong decision and add a space when it is not required. In such cases, follow the macro with  $\{ \}$ , which will suppress this space.

<p>Great Britain was unified in 1707. Great Britain, the United States of America, and Canada have close cultural links.</p>	<pre>\usepackage{xspace} \newcommand\USA{United States of America\xspace} \newcommand\GB {Great Britain\xspace} \GB was unified in 1707.\ \GB, the \USA, and Canada have close cultural links.</pre>
--	--

### 3.1.2 ellipsis, lips—Marks of omission

Omission marks are universally represented by three consecutive periods (also known as an *ellipsis*). Their spacing, however, depends on house style and typographic conventions, and significant difference are observed. In French, according to Hart [63] or *The Chicago Manual of Style* [38], “points de suspension” are set close together and immediately follow the preceding word with a space on the right:

C'est une chose... bien difficile.

In German, according to the Duden [44], “Auslassungspunkte” have space on the left *and* right unless they mark missing letters within a word or a punctuation after them is kept:

Du E... du! Scher dich zum ...!

Elsewhere, such as in British and American typography, the dots are sometimes set with full word spaces between them and rather complex rules determine how to handle other punctuation marks at either end.

$\text{\LaTeX}$  offers the commands  $\text{\dots}$  and  $\text{\textellipsis}$  to produce closely spaced omission marks. Unfortunately, the standard definition (inherited from plain  $\text{\TeX}$ ) produces uneven spacing at the left and right—unsuitable to typeset some of the above examples properly. The extra thin space at the right of the ellipsis is correct in certain situations (e.g., when a punctuation character follows). If the ellipsis is followed by space, however, it looks distinctly odd and is best canceled as shown in the example below (though removing the space in the second instance brings the exclamation mark a bit too close).

<p>Compare the following: Du E... du! Scher dich zum ...! Du E... du! Scher dich zum ...!</p>	<pre>\newcommand\lips{\dots\unkern} Compare the following:\ \ Du E\lips\ du! Scher dich zum \dots!\ \ Du E\lips\ du! Scher dich zum \lips!</pre>
---	--

3-1-2

This problem is addressed in the package `ellipsis` written by Peter Heslin, which redefines the `\dots` command to look at the following character to decide whether to add a final separation. An extra space is added if the following character is listed in the command `\ellipsispunctuation`, which defaults to “`, . : ; ! ?`”. When using some of the language support packages that make certain characters active, this list may have to be redeclared afterwards to enable the package to still recognize the characters.

The spacing between the periods and the one possibly added after the ellipsis can be controlled through the command `\ellipsisgap`. To allow for automatic adjustments depending on the font size use a font-dependent unit like `em` or a fraction of a `\fontdimen` (see page 428).

```
\usepackage{ellipsis}
Compare the following:\\
Du E\.\dots\ du! Scher dich zum \dots!\\\
\renewcommand\ellipsisgap{1.5\fontdimen3\font}\\
Du E\.\dots\ du! Scher dich zum \dots!\\\
\renewcommand\ellipsisgap{0.3em}\\
Du E\.\dots\ du! Scher dich zum \dots!
```

3-1-3

For the special case when you need an ellipsis in the middle of a word (or for other reasons want a small space at either side), the package offers the command `\midwordellipsis`. If the package is loaded with the option `mla` (Modern Language Association style), the ellipsis is automatically bracketed without any extra space after the final period.

If one follows *The Chicago Manual of Style* [38], then an ellipsis is set with full word spaces between the dots. For this, one can deploy the `lips` package<sup>1</sup> by Matt Swift. It implements the command `\lips`, which follows the recommendations in this reference book. For example, an ellipsis denoting an omission at the end of a sentence should, according to [38, §10.48–63], consist of four dots with the *first* dot being the sentence period.<sup>2</sup> The `\lips` command implements this by interpreting “`\lips.`” like “`.\lips`”, as can be seen in the next example.

```
\usepackage{moredefs,lips}
Elsewhere \dots the dots are normally set with full word spaces between them. An example would be this paragraph.
```

3-1-4

The `\lips` command looks for punctuation characters following it and ensures that in case of `, : ; ? ! )` / the ellipsis and the punctuation are not separated by a line break. In other cases (e.g., an opening parenthesis), a line break would be possible. The above list is stored in `\LPNobreakList` and can be adjusted if

<sup>1</sup> `lips` is actually part of a larger suite of packages. If used on a stand-alone basis, you also have to load the package `moredefs` by the same author.

<sup>2</sup> Not that the authors of this book can see any logic in this.

necessary. To force an unbreakable space following `\lips`, follow the command with a tie (~).

When applying the `mla` option the ellipsis generated will be automatically bracketed and a period after the `\lips` command will not be moved to the front. If necessary, `\olips` will produce the original unbracketed version.

Elsewhere . . . the dots are normally set with full word spaces between them [ . . ]. An example would be this paragraph.

```
\usepackage{moredefs}\usepackage[mla]{lips}
Elsewhere \olips the dots are normally set with
full word spaces between them \lips. An example
would be this paragraph.
```

### 3.1.3 amsmath—Nonbreaking dashes

The `amsmath` package, extensively discussed in Chapter 8, also offers one command for use within paragraphs. The command `\nobreakdash` suppresses any possibility of a line break after the following hyphen or dash. A very common use of `\nobreakdash` is to prevent undesirable line breaks in usages such as “*p*-adic” but here is another example: if you code “Pages 3–9” as `Pages 3\nobreakdash--9` then a line break will never occur between the dash and the 9.

This command must be used *immediately* before a hyphen or dash (~, --, or ---). The following example shows how to prohibit a line break after the hyphen but allow normal hyphenation in the following word (it suffices to add a zero-width space after the hyphen). For frequent use, it’s advisable to make abbreviations, such as `\p`. As a result “dimension” is broken across the line, while a break after “*p*-” is prevented (resulting in a overfull box in the example) and “3–9” is moved to the next line.

```
\usepackage{amsmath}
\newcommand{\p}[1]{#1\nobreakdash--} % "\p-adic"
\newcommand{\Ndash}{\nobreakdash--} % "3\Ndash 9"
\newcommand{\n}[1]{#1\nobreakdash-\hspace{0pt}} % "\n-dimensional"
\noindent The generalization to the \n-dimensional
case (using the standard \p-adic topology) can be found
on Pages 3\Ndash 9 of Volume IV.
```

The generalization to the *n*-dimensional case (using the standard *p*-adic topology) can be found on Pages 3–9 of Volume IV.

### 3.1.4 `relsize`—Relative changes to the font size

Standard L<sup>A</sup>T<sub>E</sub>X offers 10 predefined commands that change the overall font size (see Table 7.1 on page 342). The selected sizes depend on the document class but are otherwise absolute in value. That is, `\small` will always select the same size within a document regardless of surrounding conditions.

However, in many situations it is desirable to change the font size relative to the current size. This can be achieved with the `relsize` package, originally developed by Bernie Cosell and later updated and extended for  $\text{\LaTeX} 2\epsilon$  by Donald Arseneau and Matt Swift.

The package provides the declarative command `\relsize`, which takes a number as its argument denoting the number of steps by which to change the size. For example, if the current size is `\Large` then `\relsize{-2}` would change to `\normalsize`. If the requested number of steps is not available then the smallest (i.e., `\tiny`) or largest (i.e., `\Huge`) size command is selected. This means that undoing a relative size change by negating the argument of `\relsize` is not guaranteed to bring you back to the original size—it is better to delimit such changes by a brace group and let  $\text{\LaTeX}$  undo the modification.

The package further defines `\smaller` and `\larger`, which are simply abbreviations for `\relsize` with the arguments `-1` and `1`, respectively. Convenient variants are `\textsmaller` and `\textlarger`, whose argument is the text to reduce or enlarge in size. These four commands take as an optional argument the number of steps to change if something different from `1` (the default) is needed.

Some large text with a few small words inside.

SMALL CAPS (faked)

SMALL CAPS (real; compare the running length and stem thickness to previous line).

```
\usepackage{relsize}
\Large Some large text with a few
      {\relsize{-2}small words} inside.
\par\medskip
\normalsize\noindent
S\textsmaller[2]{MALL} C\textsmaller[2]{APS} (faked)\\
\textsc{Small Caps} (real; compare the running length
      and stem thickness to previous line).
```

3-1-7

In fact, the above description for `\relsize` is not absolutely accurate: it tries to increase or decrease the size by 20% for each step and selects the  $\text{\LaTeX}$  font size command that is closest to the resulting target size. It then compares the selected size and target size. If they differ by more than the current value of `\RSpcentTolerance` (interpreted as a percentage), the package calls `\fontsize` with the target size as one of the arguments. If this happens it is up to  $\text{\LaTeX}$ 's font selection scheme to find a font matching this request as closely as possible. By default, `\RSpcentTolerance` is an empty macro, which is interpreted as 30 (percent) when the current font shape group is composed of only discrete sizes (see Section 7.10.3), and as 5 when the font shape definition covers ranges of sizes.

Using a fixed factor of 1.2 for every step may be too limiting in certain cases. For this reason the package additionally offers the more general declarative command `\rescale{factor}` and its variant `\textscale{factor}{text}`, to select the size based on the given `factor`, such as `1.3` (enlarge by 30%).

There are also two commands, `\mathsmaller` and `\mathlarger`, for use in math mode.  $\text{\LaTeX}$  recognizes only four different math sizes, of which two (`\displaystyle` and `\textstyle`) are nearly identical for most symbols, so the application domain of these commands is somewhat limited. With `exscale` addi-

tionally loaded the situation is slightly improved: the `\mathlarger` command, when used in `\displaystyle`, will then internally switch to a larger text font size and afterwards select the `\displaystyle` corresponding to that size.

<span style="border: 1px solid black; padding: 2px;">3-1-8</span>	$\sum \neq \sum$ $\text{and } \frac{1}{2} \neq \frac{1}{2} \text{ but } N = N$	<pre>\usepackage{exscale,relsize} \[\sum \neq \mathlarger{\sum}\] and \$\frac{1}{2} \neq \frac{1}{2}\$ but \$N = \mathlarger{N}\$</pre>
---	---	---

These commands will attempt to correctly attach superscripts and subscripts to large operators. For example,

<span style="border: 1px solid black; padding: 2px;">3-1-9</span>	$\sum_{i=1}^n \neq \sum_{i=1}^n \neq \sum_{i=1}^n$ $\int_0^\infty \neq \int_0^\infty \neq \int_0^\infty$	<pre>\usepackage{exscale,relsize} \[\mathsmaller{\sum_{i=1}^n} \neq \mathlarger{\sum_{i=1}^n} \qqquad \mathsmaller{\int_0^\infty} \neq \mathlarger{\int_0^\infty}</pre>
---	---	---

Be aware that the use of these commands inside formulas will hide the true nature of the math atoms inside the argument, so that the spacing in the formula, without further help, might be wrong. As shown in following example, you may have to explicitly use `\mathrel`, `\mathbin`, or `\mathop` to get the correct spacing.

<span style="border: 1px solid black; padding: 2px;">3-1-10</span>	$a \times b \neq a \times b \neq a \times b$	<pre>\usepackage{exscale,relsize} \[ a \times b \neq a \mathlarger{\times} b \neq a \mathbin{\mathlarger{\times}} b</pre>
--	--	---

Due to these oddities, the `\mathlarger` and `\mathsmaller` commands should not be trusted blindly, and they will not be useful in every instance.

### 3.1.5 `textcase`—Change case of text intelligently

The standard L<sup>A</sup>T<sub>E</sub>X commands `\MakeUppercase` and `\MakeLowercase` change the characters in their arguments to uppercase or lowercase, respectively, thereby expanding macros as needed. For example,

```
\MakeUppercase{On \today}
```

will result in “ON 2ND AUGUST 2004”. Sometimes this will change more characters than desirable. For example, if the text contains a math formula, then uppercasing this formula is normally a bad idea because it changes its meaning. Similarly, arguments to the commands `\label`, `\ref`, and `\cite` represent semantic information, which, if modified, will result in incorrect or missing references, because L<sup>A</sup>T<sub>E</sub>X will look for the wrong labels.

```
\MakeTextUppercase{text}      \MakeTextLowercase{text}
```

The package `textcase` by David Carlisle overcomes these defects by providing two alternative commands, `\MakeTextUppercase` and `\MakeTextLowercase`, which recognize math formulas and cross-referencing commands and leave them alone.

## 1 Textcase example

TEXT IN SECTION 1, ABOUT  $a = b$  AND  $\alpha \neq a$

```
\usepackage{textcase}
\section{Textcase example}\label{exa}
\MakeTextUppercase{Text in section~\ref{exa},
about $a=b$ and $(\alpha \neq a)$}
```

3-1-11

Sometimes portions of text should be left unchanged for one reason or another. With `\NoCaseChange` the package provides a generic way to mark such parts. For instance:

```
\usepackage{textcase}
\MakeTextUppercase{Some text
\noCaseChange{Some More} text}
```

3-1-12

If necessary, this method can be used to hide syntactic information, such as

```
\noCaseChange{\begin{tabular}{ll} ... \noCaseChange{\end{tabular}}
```

thereby preventing `tabular` and `ll` from incorrectly being uppercased.

All this works only as long as the material is on the top level. Anything that is inside a group of braces (other than the argument braces to `\label`, `\ref`, `\cite`, or `\NoCaseChange`) will be uppercased or lowercased regardless of its nature.

BOTH OF THESE WILL FAIL  $A + B = C$   
UNFORTUNATELY

```
\usepackage{textcase}
\MakeTextUppercase{Both of these will
\textbf{fail $a+b=c$}}
\emph{\noCaseChange{unfortunately}}
```

3-1-13

In the above case you could avoid this pitfall by taking the formula out of the argument to `\textbf` and moving `\emph` inside the argument to `\NoCaseChange`. In other situations this kind of correction might be impossible. In such a case the (somewhat cumbersome) solution is to hide the problem part inside a private macro and protect it from expansion during the case change; this method works for the standard `LATEX` commands as well, as shown in the next example.

BUT THIS WILL WORK  $a + b = c$  ALWAYS

```
\newcommand{\mymath}{$a+b=c$}
\MakeUppercase{But this will
\textbf{work \protect\mymath} always}
```

3-1-14

Some classes and packages employ `\MakeUppercase` internally—for example, in running headings. If you wish to use `\MakeTextUppercase` instead, you should

load the `textcase` package with the option `overload`. This option will replace the standard L<sup>A</sup>T<sub>E</sub>X commands with the variants defined by the package.

### 3.1.6 `ulem`—Emphasize via underline

L<sup>A</sup>T<sub>E</sub>X encourages the use of the `\emph` command and the `\em` declaration for marking emphasis, rather than explicit font-changing declarations, such as `\bfseries` and `\itshape`. The `ulem` package (by Donald Arseneau) redefines the command `\emph` to use underlining, rather than italics. It is possible to have line breaks and even primitive hyphenation in the underlined text. Every word is typeset in an underlined box, so automatic hyphenation is normally disabled, but explicit discretionary hyphens (`\-`) can still be used. The underlines continue between words and stretch just like ordinary spaces do. As spaces delimit words, some difficulty may arise with syntactical spaces (e.g., "2.3 pt"). Some effort is made to handle such spaces. If problems occur you might try enclosing the offending command in braces, since everything inside braces is put inside an `\mbox`. Thus, braces suppress stretching and line breaks in the text they enclose. Note that nested emphasis constructs are not always treated correctly by this package (see the gymnastics performed below to get the interword spaces correct in which each nested word is put separately inside an `\emph` expression).

`\usepackage{ulem}`

No, I did not act in the movie The Persecution and Assassination of Jean-Paul Marat, as performed by the Inmates of the Asylum of Charenton under the direction of the Marquis de Sade! But I did see it.

3-1-15 Alternatively, underlining can be explicitly requested using the `\uline` command. In addition, a number of variants are available that are common in editorial markup. These are shown in the next example.

3-1-16 Double underlining (under-line),  
a wavy underline (wave),  
a line through text (strike-out),  
crossing out text (cross-out/X-out),

`\usepackage{ulem}`

Double underlining (`\uline{under-line}`),  
a wavy underline (`\uline{under-wave}`),  
a line through text (`\uline{strike out}`),  
crossing out text (`\uline{cross out}`, `\uline{X out}`),

The redefinition of `\emph` can be turned off and on by using `\normalem` and `\ULforem`. Alternatively, the package can be loaded with the option `normalem` to suppress this redefinition. Another package option is `UWforbf`, which replaces `\textbf` and `\bfseries` by `\uline` whenever possible.

The position of the line produced by `\uline` can be set explicitly by specifying a value for the length `\ULdepth`. The default value is font-dependent, denoted

by the otherwise senile value `\maxdimen`. Similarly, the thickness of the line can be controlled via `\ULthickness`, which, for some historical reason, needs to be redefined using `\renewcommand`.

### 3.1.7 soul—Letterspacing or stealing sheep

Frederic Goudy supposedly said, “Anyone who would letterspace black letter would steal sheep”. Whether true or a myth, the topic of letterspacing clearly provokes heated discussions among typographers and is considered bad practice in most situations because it changes the “grey” level of the text and thus disturbs the flow of reading. Nevertheless, there are legitimate reasons for undertaking letterspacing. For example, display type often needs a looser setting and in most fonts uppercased text is improved this way. You may also find letterspacing being used to indicate emphasis, although this exhibits the grey-level problem.

TEX is ill equipped when it comes to supporting letterspacing. In theory, the best solution is to use specially designed fonts rather than trying to solve the problem with a macro package. But as this requires the availability of such fonts, it is not an option for most users. Thus, in practice, the use of a macro-based solution is usually easier to work with, even though it means dealing with a number of restrictions. Some information about the font approach can be found in the documentation for the `fontinst` package [74, 75].

The `soul` package written by Melchior Franz provides facilities for letterspacing and underlining, but maintains TEX’s ability to automatically hyphenate words, a feature not available in `ulem`. The package works by parsing the text to be letterspaced or underlined, token by token, which results in a number of peculiarities and restrictions. Thus, users who only wish to underline a few words and do not need automatic hyphenation are probably better off with `ulem`, which is far less picky about its input.

```
\caps{text}      \hl{text}      \so{text}      \st{text}      \ul{text}
```

The use of the five main user commands of `soul` are shown in the next example. In cases where TEX’s hyphenation algorithm fails to find the appropriate hyphenation points, you can guide it as usual with the `\-` command. If the `color` package is loaded, `\hl` will work like a text marker, coloring the background using yellow as the default color; otherwise, it will behave like `\ul` and underline its argument.

```
\usepackage{soul,color}
With the \texttt{\textit{soul}} package you can
\so{letter}-space words and phrases}. Capitals
are \caps{LETTERSPACED} with a different
command. Interfaces for \ul{underlining},
\st{strikeouts}, and \hl{highlighting} are
also provided.
```

With the `soul` package you can letter-space words and phrases. Capitals are LETTERSPACED with a different command. Interfaces for underlining, ~~strikeouts~~, and highlighting are also provided.

Normally, the `soul` package interprets one token after another in the argument of `\so`, `\st`, and so on. However, in case of characters that are represented by more than one token (e.g., accented characters) this might fail with some low-level TeX error messages. Fortunately, the package already knows about all common accent commands, so these are handled correctly. For others, such as those provided by the `textcomp` package, you can announce them to `soul` with the help of a `\soulaccent` declaration. The alternative is to surround the tokens by braces.

3-1-18

```
\usepackage{soul} \usepackage{textcomp}
\soulaccent{\capitalgrave}
\Huge \st{"a `u ~O `X ^Y}
```

The `soul` package already knows that quotation characters, en dash, and em dash consist of several tokens and handles them correctly. In case of other syntactical ligatures, such as the Spanish exclamation mark, you have to help it along with a brace group.

3-1-19 “So there,” he said. \usepackage{soul}
 |HOLA—MY FRIEND! \so{“So there,”} he said. \caps{!'}Hola---my \textbf{friend}!`

The `soul` package also knows about math formulas as long as they are surrounded by \$ signs (the form `\(...\)` is not supported) and it knows about all standard font-changing commands, such as `\textbf`. If you have defined your own font-switching command or use a package that provides additional font commands, you have to register them with `soul` using `\soulregister`. This declaration expects the font command to be registered as its first argument and the number of arguments (i.e., 0 or 1) for that command to appear as its second argument. Within the `soul` commands none of the font commands inserts any (necessary) italic correction. If needed, one has to provide it manually using `\vphantom`.

3-1-20 Here we see **soul** \usepackage{soul} \soulregister{\textsf}{1}
 | in action:  $x \neq y$  OK? \so{Here we see \textsf{soul} in \emph{action}: \$x \neq y\$ OK?}`

If you look carefully, you will see that the font commands suppress letterspacing directly preceding and following them, such as between “action” and the colon. This can be corrected by adding `\>`, which forces a space.

3-1-21 b lo od y viz. b lo o dy \so{bl\textbf{oo}dy} viz. bl\>\textbf{oo}\>dy`

Text inside a brace group is regarded as a single object during parsing and is therefore not spaced out. This is handy if certain ligatures are to be kept intact inside spaced-out text. However, this method works only if the text inside the brace group contains no hyphenation points. If it does, you will receive the package error message “Reconstruction failed”. To hide such hyphenation points

you need to put the text inside an `\mbox`, as shown in the second text line of the next example (TeX would hyphenate this as “Es-cher”—that is, between the “sch” that we try to keep together). You can also use `\soulomit` to achieve this effect, but then your text will work only when the `soul` package is loaded.

Schüss vorrichtung Gödel, Escher, Bach Temporarily disabling the scanner	<pre>\usepackage{soul,yfonts} \usepackage[latin1]{inputenc} \textfrak{\so{S{ch}u{tz}vorri{ch}tung}} \par \so{Gödel, E\mbox{sch}er, Bach} \par \ul{Temporarily dis\soulomit{abl}ing the scanner}</pre>	3-1-2
--	---	-------

One of the most important restrictions of the above commands is that they cannot be nested; any attempt to nest `soul` commands will result in low-level TeX errors. If you really need nesting you will have to place the inner material in a `box`, which means you lose the possibility to break the material at a line ending.

This is hell for all of us!	<pre>\usepackage{soul} \newsavebox\soulbox \sbox\soulbox{\so{ is hell }} \ul{This\mbox{\usebox{\soulbox}}for all of us!}</pre>	3-1-2
-----------------------------	--	-------

A few other commands are special within the argument of `\so` and friends. Spacing out at certain points can be canceled using `\<` or forced with `\>` as we saw above. As usual with L<sup>A</sup>T<sub>E</sub>X a `\~` will produce an unbreakable space. The `\`` command is supported, though only in its basic form—no star, no optional argument. You can also use `\linebreak` to break a line at a certain point, but again the optional argument is not supported. Other L<sup>A</sup>T<sub>E</sub>X commands are likely to break the package—some experimentation will tell you what is safe and what produces havoc. The next example shows applications of these odds and ends.

“So there” he said. Let’s produce a spaced out line, OK?	<pre>\usepackage{soul} \so{“\&lt;So there\&gt;” he said. Let’s\` produce a spaced out line\&gt;, \linebreak OK?}</pre>	3-1-2
--	--	-------

`\sodef{cmd}{font}{inter-letter space}{word space}{outer space}`

The `\sodef` declaration allows you to define your own letterspacing commands. It can also be used to overwrite the defaults for `\so`.

The letterspacing algorithm works by putting a certain *inter-letter space* between characters of a word, a certain *word space* between words, and a certain *outer space* at the beginning and end of the letterspaced text section. The latter space is added only if it is appropriate at that point. The default values for these spaces are adjusted for typesetting texts in Fraktur fonts but with the help of the `\sodef` declaration it is easy to adjust them for your own needs. The *font* argument allows you to specify font attributes; in most cases it will be empty. Rather than using explicit dimensions in the other arguments it is advisable to resort to

`em` values, thereby making your definition depend on the current font and its size.

```
\usepackage{soul}
\sodef\sobf{\bfseries}{.3em}{1em plus .1em}
{1.3em plus .1em minus .2em}
```

**[3-1-25]** Here we **e m p h a s i z e w o r d s** a lot. Here we `\sobf{emphasize words}` a lot.

While `\so` or any new command defined via `\sodef` simply retrieves and executes its stored definition, the `\caps` command works somewhat differently. It examines the current font and tries to find it (or a close match) in an internal database. It then uses the letterspacing values stored there. You can extend this database using the `\capsdef` declaration by providing values for individual fonts or groups of fonts. In this way you can fine-tune the letterspacing—for example, for text in headings. It is even possible to keep several such databases and change them on the fly within a document.

`\capsdef{match spec}{font}{inter-letter space}{word space}{outer space}`

Apart from the first argument, which is totally different, the other arguments to `\capsdef` are identical to those of `\sodef`. The first argument, *match spec*, defines the font (or fonts) to which the current declaration applies.

Its syntax is *encoding*, *family*, *series*, *shape*, and *size* separated by slashes using the naming conventions of NFSS. Empty values match anything, so `///` matches any font, `/ptm///10` matches all Times fonts in 10 points, and `OT1/cmr/m/n/` matches Computer Modern (cmr) medium series (m) normal shape (n) encoded in OT1 in any size. It is also possible to specify size ranges. For example, `5-14` means  $5 \text{ pt} \leq \text{size} < 14 \text{ pt}$  and `14-` matches all sizes equal or greater 14 pt. Refer to the tables in Chapter 7 for details on the NFSS font naming conventions.

As with `\sodef`, in most declarations the *font* argument will be empty. On some occasions it may make sense to use `\scshape` in this place, such as to change the font shape to small caps before applying letterspacing.

Because `\caps` uses the first matching entry in its database, the order of `\capsdef` declarations is important. Later declarations are examined first so that it is possible to overwrite or extend existing declarations.

```
\usepackage[titlesec,soul]
\newcommand\allcaps[1]{\MakeUppercase{\caps{#1}}}
\titleformat{\section}[block]{\centering\sffamily}
{\thesection}{.5em}{\allcaps}
\titlespacing*\{\section}{0pt}{8pt}{3pt}
\capsdef{/phv///}{\scshape}{.17em}{.55em}{.4em}
\section*{A Sample Heading}
The \verb=\capsdef= declaration applies here, because the heading definition specifies sans serif and our examples are typeset with Times and Helvetica (\texttt{phv}).
```

## A SAMPLE HEADING

The `\capsdef` declaration applies here, because the heading definition specifies sans serif and our examples are typeset with Times and Helvetica (`\texttt{phv}`).

**[3-1-26]**

The previous example also contained an interesting combination of `\caps` and `\MakeUppercase`: the command `\allcaps` changes its argument to uppercase and then uses `\caps` to letterspace the result.

```
\capssave{name}    \capsselect{name}    \capsreset
```

*Customized  
letterspacing for  
different occasions*

With `\capsreset` the database is restored to its initial state containing only a generic default. You can then add new entries using `\capsdef`. The current state of the `\caps` database can be stored away under a *name* by using `\capssave`. You can later retrieve this state by recalling it with `\capsselect`. If you use the `capsdefault` option when loading the package, then all uses of `\caps` that have no matching declaration are flagged by underlining the text.

```
\usepackage{titlesec} \usepackage[capsdefault]{soul}
\capsdef{/phv///}{\scshape}{.17em}{.55em}{.4em}
\capssave{display}    \capsreset
\capsdef{/phv///}{\scshape}{.04em}{.35em}{.35em}
\titlespacing*{\section}{0pt}{8pt}{3pt}
\titleformat{\section}[block]{\centering\sffamily}
{\thesection.}{5em}{\capsselect{display}\caps}
\section*{A Sample Heading}
Notice the different letterspacing in the heading and RUNNING TEXT. For Times we have no definition above so that the DEFAULT will match.
```

3-1-27

## A SAMPLE HEADING

Notice the different letterspacing in the heading and **RUNNING TEXT**. For Times we have no definition above so that the `DEFAULT` will match.

*Customizing  
underlinings*

The position and the height of the line produced by the `\ul` command can be customized using either `\setul` or `\setuldepth`. The command `\setul` takes two dimensions as arguments: the position of the line in relation to the baseline and the height of the line. Alternatively, `\setuldepth` can be used to specify that the line should be positioned below the text provided as an argument. Finally, `\resetul` will restore the default package settings.

Here we test  
a number of  
different settings.  
And back to normal!

```
\usepackage{soul}
\setul{.4pt} \ul{Here we test} \par
\setul{-.6ex}{.3ex} \ul{a number of} \par
\setuldepth{g} \ul{different settings.} \par
\resetul \ul{And back to normal!}
```

3-1-28

Both `\ul` and `\st` use a black rule by default. If you additionally load the `color` package, you can use colored rules instead and, if desired, modify the highlighting color as demonstrated below:

Rules can be in black blue.

```
\usepackage{soul,color}
\sethlcolor{green} \setulcolor{blue} \setstcolor{red}
Rules \hl{can} be in \st{black} \ul{blue}.
```

3-1-29

### 3.1.8 url—Typesetting URLs, path names, and the like

E-mail addresses, URLs, path or directory names, and similar objects usually require some attention to detail when typeset. For one thing, they often contain characters with special significance to L<sup>A</sup>T<sub>E</sub>X, such as ~, #, &, {, or }. In addition, breaking them across lines should be avoided or at least done with special care. For example, it is usually not wise to break at a hyphen, because then it is not clear whether the hyphen was inserted because of the break (as it would be the case with normal words) or was already present. Similar reasons make breaks at a space undesirable. To help with these issues, Donald Arseneau wrote the `url` package, which attempts to solve most of these problems.

```
\url{text} \url!text! \path{text} \path=text=
```

The base command provided by the package is `\url`, which is offered in two syntax variants: the *text* argument either can be surrounded by braces (in which case the *text* must not contain unbalanced braces) or, like `\verb`, can be delimited by using an arbitrary character on both sides that is not used inside *text*. (The syntax box above uses ! and = but these are really only examples.) In that second form one can have unbalanced braces in the argument.

The `\path` command is the same except that it always uses typewriter fonts (`\ttfamily`), while `\url` can be customized as we will see below. The argument to both commands is typeset pretty much verbatim. For example, `\url{~}` produces a tilde. Spaces are ignored by default, as can be seen in the following example.

<pre>\usepackage{url}</pre> <p>The L<sup>A</sup>T<sub>E</sub>X project web pages are at http://www.latex-project.org and my home directory is ~frank (sometimes).</p>	<pre>\usepackage{url}</pre> <p>The L<sup>A</sup>T<sub>E</sub>X project web pages are at \url{http://www.latex-project.org} and my home directory is \path+~frank+ (sometimes).</p>
---	--

3-1-30 Line breaks can happen at certain symbols (by default, not between letters or hyphens) and in no case can the commands add a hyphen at the break point. Whenever the *text* contains either of the symbols % or #, or ends with \, it cannot be used in the argument to another command without producing errors (just like the `\verb` command). Another case that does not work properly inside the argument of another command is the use of two ^ characters in succession. However, the situation is worse in that case because one might not even get an error but simply incorrect output<sup>1</sup> as the next example shows.

<pre>\usepackage{url}</pre> <p><sup>3-1-31</sup> ^frank and ^frank (OK)</p>	<pre>\usepackage{url}</pre> <p>\url{^frank} and \mbox{\url{^frank}} (OK) \par</p>
	<pre>\usepackage{url}</pre> <p>~~frank but &amp;rank (bad)</p>

<sup>1</sup>It depends on the letter that is following. An uppercase F instead of the lowercase f would produce an error.

Even if the *text* does not contain any critical symbols, it is always forbidden to use such a command inside a moving argument—for instance, the argument of a `\section`. If used there, you will get the error message

```
! Undefined control sequence.
\Url Error ->\url used in a moving argument.
```

followed by many strange errors. Even the use of `\protect` will not help in that case. So what can be done if one needs to cite a path name or a URL in such a place? If you are prepared to be careful and only use “safe” characters inside *text*, then you can enable the commands for use in moving arguments by specifying the option `allowmove` when loading the package. But this does not help if you actually need a character like “#”. In that case the solution is to record the information first using `\urldef` and then reuse it later.

```
\urldef{cmd}{url-cmd}{text}      \urldef{cmd}{url-cmd}=text=
```

The declaration `\urldef` defines a new command *cmd* to contain the *url-cmd* (which might be `\url`, `\path`, or a newly defined command—see below) and the *text* in a way such that they can be used in any place, including a moving argument. The *url-cmd* is not executed at this point, which means that style changes can still affect the typesetting (see Example 3-1-33 on the facing page). Technically, what happens is that the `\catcodes` of characters in *text* are frozen during the declaration, so that they cannot be misinterpreted in places like arguments.

## 1 `^^frank~#$\ works?`

It does—in contrast to the earlier example.

```
\usepackage{url}
\urldef\test\path{^^frank~#$\$}
\section{\test{} works?}
```

It does---in contrast to the earlier example.

3-1-3-

`\urlstyle{style}`

We have already mentioned style changes. For this task the `url` package offers the `\urlstyle` command, which takes one mandatory argument: a named *style*. Predefined styles are `rm`, `sf`, `tt`, and `same`. The first three select the font family of that name, while the `same` style uses the current font and changes only the line breaking.

The `\url` command uses whatever style is currently in force (the default is `tt`, i.e., typewriter), while `\path` internally always switches to the `tt` style. In the following example we typeset a URL saved in `\lproject` several times using different styles. The particular example may look slightly horrifying, but imagine how

it would have looked if the URL had not been allowed to split at all in this narrow measure.

```
Zapf Chancery!      http://www.latex-project.org (default set-up) http://www.latex-project.org (CM Roman) http://www.latex-project.org (CM Sans Serif) http://www.latex-project.org (CM Typewriter) http://www.latex-project.org (Zapf Chancery)
\usepackage[hyphens]{url}
\urldef{\lproject}\url{http://www.latex-project.org}
\fontfamily{pzc}\selectfont Zapf Chancery!
\lproject\ (default set-up) \quad
\urlstyle{rm}\lproject\ (CM Roman) \quad
\urlstyle{sf}\lproject\ (CM Sans Serif) \quad
\urlstyle{tt}\lproject\ (CM Typewriter) \quad
\urlstyle{same}\lproject\ (Zapf Chancery)
```

3-1-33

If you studied the previous example closely you will have noticed that the option `hyphens` was used. This option allows breaking at explicit hyphens, something normally disabled for `\url`-like commands. Without this option breaks would have been allowed only at the periods, after the colon, or after “//”.

As mentioned earlier spaces inside `text` are ignored by default. If this is not desired one can use the option `obeyspaces`. However, this option may introduce spurious spaces if the `\url` command is used inside the argument of another command and `text` contains any “\” character. In that case `\urldef` solves the problem. Line breaks at spaces are not allowed unless you also use the option `spaces`.

*Spaces in the argument*

The package automatically detects which font encoding is currently in use. In case of T1 encoded fonts it will make use of the additional glyphs available in this encoding, which improves the overall result.

The package offers two hooks, `\UrlLeft` and `\UrlRight`, that by default do nothing but can be redefined to typeset material at the left or right of `text`. The material is typeset in the same fashion as the `text`. For example, spaces are ignored unless one uses `\_` or specifies `obeyspaces` as an option. If the commands are redefined at the top level, they act on every `\url`-like command. See Example 3-1-34 on the next page for a possibility to restrict their scope.

*Appending material at left or right*

`\DeclareUrlCommand{cmd}{style-information}`

It is sometimes helpful to define your own commands that work similarly to `\url` or `\path` but use their own fonts, and so on. The command `\DeclareUrlCommand` can be used to define a new `\url`-like command or to modify an existing one. It takes two arguments: the command to define or change and the *style-information* (e.g., `\urlstyle`).

*Defining URL-like commands*

In the next example, we define `\email` to typeset e-mail addresses in `rm` style, prepending the string “e-mail: ” via `\UrlLeft`. The example clearly shows that the scope for this redefinition is limited to the `\email` command. If you look closely,

you can see that a space inside `\UrlLeft` (as in the top-level definition) has no effect, while `\_` produces the desired result.

```
\usepackage{url}
\renewcommand\UrlLeft{<url: }
\renewcommand\UrlRight{>}
\DeclareUrlCommand\email{\urlstyle{rm}%
  \renewcommand\UrlLeft{e-mail:\_ }%
  \renewcommand\UrlRight{}}
<url:> http://www.latex-project.org \url{http://www.latex-project.org} \par
e-mail: frank.mittelbach@latex-project.org \email{frank.mittelbach@latex-project.org} \par
<url:$HOME/figures> oops, wrong! \path{$HOME/figures} oops, wrong!
```

3-1-34

The `url` package offers a number of other hooks that influence line breaking, among them `\UrlBreaks`, `\UrlBigBreaks`, and `\UrlNoBreaks`. These hooks can be redefined in the *style-information* argument of `\DeclareUrlCommand` to set up new or special conventions. For details consult the package documentation, which can be found at the end of the file `url.sty`.

### 3.1.9 euro—Converting and typesetting currencies

To ease the calculations needed to convert between national units and the euro, Melchior Franz developed the package `euro`. In fact, the package converts arbitrary currencies using the euro as the base unit. The calculations are done with high precision using the `fp` package written by Michael Mehlich. The formatting is highly customizable on a per-currency basis, so that this package can be used for all kind of applications involving currencies whether or not conversions are needed.

`\EURO{from-currency} [to-currency] {amount}`

The main command `\EURO` converts an *amount* in *from-currency* into *to-currency* or, if this optional argument is missing, into euros. The arguments *from-currency* and *to-currency* are denoted in ISO currency codes, as listed in Table 3.1 on the facing page. When inputting the *amount* a dot must separate the integer value from any fractional part, even if the formatted number uses a different convention.

With the default settings the *amount* is displayed in the *from-currency* with the converted value in the *to-currency* shown in parentheses.

```
\usepackage{euro}
\EURO{DEM}[FRF]{7}\quad \EURO{FRF}[DEM]{23.48}
\\
\EURO{EUR}[DEM]{10.00}\quad \EURO{DEM}{20}
```

3-1-35

EUR	Europe	GRD	Greece
ATS	Austria	IEP	Ireland
BEF	Belgium	ITL	Italy
DEM	Germany	LUF	Luxembourg
ESP	Spain	NLG	The Netherlands
FIM	Finland	PTE	Portugal
FRF	France		

Table 3.1: ISO currency codes of the *euro* and the 12 *euro-zone* countries

The package offers a number of options to influence the general style of the output (unless overwritten by the more detailed formatting declarations discussed below). With `eco` the ISO codes precede the value and no customized symbols are used; with `dots` a period is inserted between every three-digit group (the default is to use a small space).

*The package options*

By default, integer amounts are printed as such, without adding a decimal separator and a (zero) fractional part. If the `table` option is specified this behavior is globally changed and either a — (option `emdash`, also the default), a – (option `endash`), or the right number of zeros (option `zeros`) is used.

<span style="border: 1px solid black; padding: 2px;">3-1-36</span>	<pre>\usepackage[eco,table,endash]{euro}</pre>
--	--

<span style="border: 1px solid black; padding: 2px;">3-1-36</span>	<pre>DEM 7,— (FRF 23,48)  FRF 23,48 (DEM 7,—)</pre>	<pre>\EURO{DEM}{FRF}{7}\quad \EURO{FRF}{DEM}{23.48}</pre>
--	---	---

The more detailed output customizations, which we discuss below, can be placed anywhere in the document. It is, however, advisable to keep them together in the preamble, or even to put them into the file `euro.cfg`, which is consulted upon loading the package.

The monetary symbols typeset can be adjusted with a `\EUROSYM` declaration; as defaults the package uses the ISO codes for most currencies. The example below changes the presentation for lira and euro using the currency symbols from the `textcomp` package. It also uses `dots` to help with huge lira amounts.

<span style="border: 1px solid black; padding: 2px;">3-1-37</span>	<pre>10.000 £ (5,16 €)  1.000 DM (989.999 £)</pre>	<pre>\usepackage{textcomp}\usepackage[dots]{euro}</pre>
--	--	---

<span style="border: 1px solid black; padding: 2px;">3-1-37</span>	<pre>\EUROSYM{ITL}{\textlira}\EUROSYM{EUR}{\texteuro}</pre>
--	---

The package is well prepared for new countries to join the euro-zone. In fact, it is well prepared to deal with conversions from and to any currency as long as the conversion rate to the euro is known. To add a new currency use the `\EUROADD` declaration, which takes three arguments: the ISO currency code, the symbol or text to display for the currency, and the conversion rate to the euro. The next

example makes the British pound available. Note the abbreviation \GBP, which makes the input a bit easier.

14,90 £ (23,29 €) 10 £ (102,54 FRF) 10 € (6,40 £)	<pre>\usepackage{eurosans,euro} \EUROADD{\textsterling}{0.6397} % 2002/12/21 \newcommand*\GBP{\text{\texteuro{}}} \newcommand*\EUR{\text{\texteuro{}}} \noindent \GBP{14.9}\ \ \GBP[FRF]{10}\ \ \EUR[GBP]{10}</pre>	3-1-38
---	---	--------

The conversion rates for the national currencies of the euro-zone countries are fixed (and predefined by the package). With other currencies the rates may change hourly, so you have to be prepared for frequent updates.

The package allows you to tailor the presentation via \EUROFORMAT declarations, either to provide new defaults or to adjust the typesetting of individual currencies. The first argument specifies which part of the formatting should be adjusted, and the second argument describes the formatting.

The main format specifies how the source and target currencies are to be arranged using the reserved keywords \in and \out to refer to the source and target currencies, respectively. In the example below the first line implements a format close to the default, the second line displays the result of the conversion, and the third line does not show the conversion at all (although it happens behind the scenes). The latter is useful if you want to make use of the currency formatting features of the package without being interested in any conversion.

1 000 DM (= 3 353,85 FRF) 3 353,85 FRF 1 000 DM	<pre>\usepackage{euro} \EUROFORMAT{main}{\in\ (=,\out)} \EURO{DEM}[FRF]{1000}\par \EUROFORMAT{main}{\out} \EURO{DEM}[FRF]{1000}\par \EUROFORMAT{main}{\in} \EURO{DEM}{1000}</pre>	3-1-39
---	---	--------

The `in` and `out` formats specify how the source and target currencies should be formatted using the reserved keywords `\val` (monetary amount), `\iso` (currency code), and `\sym` (currency symbol if defined; ISO code otherwise).

DM 1 000 (FRF 3 353,85)	<pre>\usepackage{euro} \EUROFORMAT{in}{\sym~\val} \EUROFORMAT{out}{\iso~\val} \EURO{DEM}[FRF]{1000}</pre>	3-1-40
-------------------------	---	--------

Perhaps more interesting are the possibilities to influence the formatting of monetary amounts, for which the package offers five declarations to be used in the second argument to \EUROFORMAT. The `\round` declaration specifies where to round the monetary amount: positive values round to the integer digits and negative values to the fractional digits. For example, `\round{-3}` means show and round to three fractional digits. The `\form` declaration takes three arguments: the integer group separator (default `\,`), the decimal separator (default a comma), and the fractional group separator (default `\,`).

The first argument can be either `all` to define the default number formatting or an ISO currency code to modify the formatting for a single currency.

<b>3-1-41</b> 1,022.5838 Euro –335.3855 FRF 9,900,000 Lit.	<code>\usepackage{euro} \EUROFORMAT{main}{\out}</code> <code>\EUROFORMAT{all}{\round{-4}\form{,}{\textperiodcentered}{}}</code> <code>\EUROFORMAT{ITL}{\round{2}}</code> <code>\noindent \EURO{DEM}{2000} \EURO{DEM}[FRF]{-100} \EURO{DEM}[ITL]{10000}</code>
---	--

The `\minus` declaration formats negative values by executing its first argument before the number and its second argument after it (default `\minus{$-$}{}`). The number itself is typeset unsigned, so that a minus sign has to be supplied by the declaration. The `\plus` declaration is the analogue for dealing with positive numbers (default `\plus{}{}`).

<b>3-1-42</b> +1 022,58 Euro   –335,39 FRF	<code>\usepackage{color,euro} \EUROFORMAT{main}{\out}</code> <code>\EUROFORMAT{all}{\plus{\$+\$}{}}\minus{\color{blue}{\$-\$}}{}}</code> <code>\EURO{DEM}{2000}\quad \EURO{DEM}[FRF]{-100}</code>
---	---

The `\zero` declaration takes three arguments to describe what to do if everything is zero, the integer part is zero, or the fractional part is zero. In the first and third arguments, the decimal separator has to be entered as well, so it should correspond to the default or the value given in the `\form` command.

<b>3-1-43</b> 0,00 €   0,51 €   1,- €	<code>\usepackage{eurosans,euro}</code> <code>\EUROFORMAT{main}{\out} \EUROSYM{EUR}{\euro}</code> <code>\EUROFORMAT{all}{\zero{0,00}{0}{--}}</code> <code>\EURO{DEM}{0}\quad \EURO{DEM}{1}\quad \EURO{EUR}{1}</code>
--	---

### 3.1.10 `lettrine`—Dropping your capital

In certain types of publications you may find the first letter of some paragraphs being highlighted by means of an enlarged letter often dropped into the paragraph body (so that the paragraph text flows around it) and usually followed by the first phrase or sentence being typeset in a special font. Applications range from chapter openings in novels, or indications of new thoughts in the text, to merely decorative elements to produce lively pages in a magazine. This custom can be traced back to the early days of printing, when such initials were often hand-colored after the printing process was finished. It originates in the manuscripts of the Middle Ages; that is, it predates the invention of printing.

```
\lettrine[key/val-list]{initial}{text}
```

The package `lettrine` written by Daniel Flipo lets you create such initials by providing the command `\lettrine`. In its simplest form it takes two arguments: the

letter to become an initial and the follow-up text to be typeset in a special font, by default in `\scshape`.

**L**A MOITIÉ DES PASSAGERS, affaiblis, expirants de ces angoisses inconcevables que le roulis d'un vaisseau porte dans les nerfs et dans toutes les humeurs du corps agitées en sens contraire,...

```
\usepackage{lettrine} \usepackage[latin1]{inputenc}
\usepackage[french]{babel}
\lettrine[L]{a moitié des passagers,} affaiblis,
expirants de ces angoisses inconcevables que le
roulis d'un vaisseau porte dans les nerfs et
dans toutes les humeurs du corps agitées en sens
contraire, \ldots
```

3-1-44

The font used for the initial is, by default, a larger size of the current text font. Alternatively, you can specify a special font family by redefining the command `\LettrineFontHook` using standard NFSS commands. Similarly, the font used for the text in the second argument can be modified by changing `\LettrineTextFont`.

Because the `\lettrine` command calculates the initial size to fit a certain number of lines, you need scalable fonts to obtain the best results. As the examples in this book are typeset in Adobe Times and Helvetica by default, we have no problems here. Later examples use Palatino, which is also a scalable Type 1 font. But if you use a bitmapped font, such as Computer Modern, you might have to use special .fd files (see Chapter 7, pages 419ff) to achieve acceptable results.

**L**A MOITIÉ DES PASSAGERS, affaiblis, expirants de ces angoisses inconcevables que le roulis d'un vaisseau porte dans les nerfs et dans toutes les humeurs du corps agitées en sens contraire,...

```
\usepackage{lettrine} \usepackage[latin1]{inputenc}
\usepackage[french]{babel}
\renewcommand\LettrineFontHook{\sfamily\bfseries}
\renewcommand\LettrineTextFont{\sfamily\scshape}
\lettrine[L]{a moitié des passagers,} affaiblis,
expirants de ces angoisses inconcevables que le
roulis d'un vaisseau porte dans les nerfs et
dans toutes les humeurs du corps agitées en sens
contraire, \ldots
```

3-1-45

Many books on typography give recommendations about how to best set large initials with respect to surrounding text. For highest quality it is often necessary to manually adjust the placement depending on the shape of the initial. For example, it is often suggested that letters with a projecting left stem should overhang into the margin. The `\lettrine` command caters to this need by supporting an optional argument in which you can specify adjustments in the form of a comma-separated list of key/value pairs.

The size of the initial is calculated by default to have a height of two text lines (stored in `\DefaultLines`); with the keyword `lines` you can change this value to a different number of lines. There is an exception: if you specify `lines=1` the initial is still made two lines high, but instead of being dropped is placed onto the baseline of the first text line.

If you want a dropped initial that also extends above the first line of text, then use the keyword `oversize`. A value of `.2` would enlarge the initial by 20%. The default value for this keyword is stored in `\DefaultLoversize`. This keyword is also useful in conjunction with `lraise` (default 0 in `\DefaultLraise`). In case of an initial with a large descender such as a “Q” you may have to raise the initial to avoid it overprinting following lines. In that case `oversize` can be used to reduce the height so as to align the initial properly.

With the keyword `lhang` you specify how much the initial extends into the margin. The value is specified as a fraction—that is, between 0 and 1. Its document default is stored in `\DefaultLhang`.

**3-1-46** **Q**UAND ILS FURENT revenus un peu à eux, ils marchèrent vers Lisbonne ; il leur restait quelque argent, avec lequel ils espéraient se sauver de la faim après avoir échappé à la tempête ...

```
\usepackage{palatino,lettrine}
\usepackage[latin1]{inputenc}
\usepackage[french]{babel}
\lettrine[lines=3, oversize=-0.1, lraise=0.1,
lhang=.2]{Q}{uand ils furent} revenus un peu à eux,
ils marchèrent vers Lisbonne ; il leur restait quelque
argent, avec lequel ils espéraient se sauver de la
faim après avoir échappé à la tempête \ldots
```

The distance between the initial and the following text in the first line is controlled by the command `\DefaultFindent` (default `Opt`) and can be overwritten using the keyword `findent`. The indentation of following lines is by default `0.5em` (stored in `\DefaultNindent`) but can be changed through the keyword `nindent`. If you want to specify a sloped indentation you can use the keyword `slope`, which applies from the third line onward. Again the default value can be changed via the command `\DefaultSlope`, though it seems questionable that you would ever want anything different than `Opt` since a slope is normally only used for letters like “A” or “V”.

**3-1-47** **A**PEINE ONT-ILS MIS le pied dans la ville en pleurant la mort de leur bienfaiteur, qu'ils sentent la terre trembler sous leurs pas ; ...

```
\usepackage{palatino,lettrine}
\usepackage[latin1]{inputenc}
\usepackage[french]{babel}
\lettrine[lines=4, slope=0.6em, findent=-1em,
nindent=0.6em]{A}{apeine ont-ils mis} le pied dans
la ville en pleurant la mort de leur bienfaiteur,
qu'ils sentent la terre trembler sous leurs pas; \ldots
```

The example above clearly demonstrates that the size calculation for the initial does not take accents into account, which is normally the desired behavior. It is nevertheless possible to manually adjust the size using `oversize`.

To attach material to the left of the initial, such as some opening quote, you can use the keyword `ante`. It is the only keyword for which no command exists to set the default.

By modifying the default settings you can easily adapt the package to typeset initials the way you like. This can be done either in the preamble or in a file with the name `lettrine.cfg`, which is loaded if found.

### 3.1.11 Paragraph justification in L<sup>A</sup>T<sub>E</sub>X

For formatting paragraphs L<sup>A</sup>T<sub>E</sub>X deploys the algorithms already built into the T<sub>E</sub>X program, which by default produce justified paragraphs. In other words, spaces between words will be slightly stretched or shortened to produce lines of equal length. T<sub>E</sub>X achieves this outcome with an algorithm that attempts to find an optimal solution for a whole paragraph, using the current settings of about 20 internal parameters. They include aspects such as trying to produce visually compatible lines, such that a tight line is not followed by one very loosely typeset, or considering several hyphens in a row as a sign of bad quality. The interactions between these parameters are very subtle and even experts find it difficult to predict the results when tweaking them. Because the standard settings are suitable for nearly all applications, we describe only some of the parameters in this book. Appendix B.3.3 discusses how to trace the algorithm. If you are interested in delving further into the matter of automatic paragraph breaking, refer to *The T<sub>E</sub>Xbook* [82, chap. 14], which describes the algorithm in great detail, or to the very interesting article by Michael Plass and Donald Knuth on the subject, which is reprinted in *Digital Typography* [98].

The downside of the global optimizing approach of T<sub>E</sub>X, which you will encounter sooner or later, is that making small changes, like correcting a typo near the end of a paragraph, can have drastic and surprising effects, as it might affect the line breaking of the whole paragraph. It is possible, and not even unlikely, that, for example, the *removal* of a word might actually result in making a paragraph one line *longer*. This behavior can be very annoying if you are near the end of finishing an important project (like the second edition of this book) and a correction wreaks havoc on your already manually adjusted page breaks. In such a situation it is best to place `\linebreak` or `\pagebreak` commands into strategic places to force T<sub>E</sub>X to choose a solution that it would normally consider inferior. To be able to later get rid of such manual corrections you can easily define your own commands, such as

```
\newcommand\finallinebreak{\linebreak}
```

rather than using the standard L<sup>A</sup>T<sub>E</sub>X commands directly. This helps you to distinguish the layout adjustments for a particular version from other usages of the original commands—a method successfully used in the preparation of this book.

The interword spacing in a justified paragraph (the white space between individual words) is controlled by several T<sub>E</sub>X parameters—the most important ones are `\tolerance` and `\emergencystretch`. By setting them suitably for your document you can prevent most or all of the “Overfull box” messages without any manual line breaks. The `\tolerance` command is a means for setting how much the interword space in a paragraph is allowed to diverge from its optimum value.<sup>1</sup> This command is a T<sub>E</sub>X (not L<sup>A</sup>T<sub>E</sub>X) counter and therefore it has an uncommon

<sup>1</sup>The optimum is font defined; see Section 7.10.3 on page 428.

assignment syntax—for example, `\tolerance=500`. Lower values make  $\text{\TeX}$  try harder to stay near the optimum; higher values allow for loose typesetting. The default value is often 200. When  $\text{\TeX}$  is unable to stay in the given tolerance you will find overfull boxes in your output (i.e., lines sticking out into the margin like this).

Enlarging the value of `\tolerance` means that  $\text{\TeX}$  will also consider poorer but still acceptable line breaks, instead of turning the problem over to you for manual intervention. Sensible values are between 50 and 9999. Do not use 10000 or higher, as it allows  $\text{\TeX}$  to produce arbitrarily bad lines (like this)

 Careful with  
 $\text{\TeX}$ 's idea about  
infinitely bad

one.

If you really need fully automated line breaking, it is better to set the length parameter `\emergencystretch` to a positive value. If  $\text{\TeX}$  cannot break a paragraph without producing overfull boxes (due to the setting of `\tolerance`) and `\emergencystretch` is positive, it will add this length as stretchable space to every line, thereby accepting line-breaking solutions that have been rejected before. You may get some underfull box messages because all the lines are now set in a loose measure, but this result will still look better than a single horrible line in the middle of an otherwise perfectly typeset paragraph.

$\text{\LaTeX}$  has two predefined commands influencing the above parameters: `\fussy`, which is the default, and `\sloppy`, which allows for relatively bad lines. The `\sloppy` command is automatically applied by  $\text{\LaTeX}$  in some situations (e.g., when typesetting `\marginpar` arguments or `p` columns in a `tabular` environment) where perfect line breaking is seldom possible due to the narrow measure.

### Unjustified text

While the theory on producing high-quality justified text is well understood (even though surprisingly few typesetting systems other than  $\text{\TeX}$  use algorithms that can produce high quality other than by chance), the same cannot be said for the situation when unjustified text is being requested. This may sound strange at first hearing. After all, why should it be difficult to break a paragraph into lines of different length? The answer lies in the fact that we do not have quantifiable quality measures that allow us to easily determine whether a certain breaking is good or bad. In comparison to its work with justified text,  $\text{\TeX}$  does a very poor job when asked to produce unjustified paragraphs. Thus, to obtain the highest quality we have to be prepared to help  $\text{\TeX}$  far more often by adding explicit line breaks in strategic places. A good introduction to the problems in this area is given in an article by Paul Stiff [154].

The main type of unjustified text is the one in which lines are set flush left but are unjustified at the right. For this arrangement  $\text{\LaTeX}$  offers the environment `flushleft`. It typesets all text in its scope “flush left” by adding very stretchable white space at the right of each line; that is, it sets the internal parameter `\rightskip` to `0pt plus 1fil`. This setting often produces very ragged-looking paragraphs as it makes all lines equally good independent of the amount of text they contain. In addition, hyphenation is essentially disabled because a hyphen

adds to the “badness” of a line and, as there is nothing to counteract it,  $\TeX$ ’s paragraph-breaking algorithm will normally choose line breaks that avoid them.

“The  $\LaTeX$  document preparation system is a special version of Donald Knuth’s  $\TeX$  program.  $\TeX$  is a sophisticated program designed to produce high-quality typesetting, especially for mathematical text.”

```
\begin{flushleft}
‘‘The \LaTeX{} document preparation system is a special
version of Donald Knuth’s \TeX{} program. \TeX{} is a
sophisticated program designed to produce high-quality
typesetting, especially for mathematical text.’’
\end{flushleft}
```

3-1-48

In summary,  $\LaTeX$ ’s `flushleft` environment is not particularly well suited to continuous unjustified text, which should vary at the right-hand boundary only to a certain extent and where appropriate should use hyphenation (see the next section for alternatives). Nevertheless, it can be useful to place individual objects, like a graphic, flush left to the margin, especially since this environment adds space above and below itself in the same way as list environments do.

Another important restriction is the fact that the settings chosen by this environment have no universal effect, because some environments (e.g., `minipage` or `tabular`) and commands (e.g., `\parbox`, `\footnote`, and `\caption`) restore the alignment of paragraphs to full justification. That is, they set the `\rightskip` length parameter to `0pt` and thus cancel the stretchable space at the right line endings. A way to automatically deal with this problem is provided by the package `ragged2e` (see next section).

Other ways of typesetting paragraphs are flush right and centered, with the `flushright` and `center` environments, respectively. In these cases the line breaks are usually indicated with the `\backslash` command, whereas for ragged-right text (the `flushleft` environment discussed above) you can let  $\LaTeX$  do the line breaking itself (if you are happy with the resulting quality).

The three environments discussed in this section work by changing declarations that control how  $\TeX$  typesets paragraphs. These declarations are also available as  $\LaTeX$  commands, as shown in the following table of correspondence:

<i>environment:</i>	<code>center</code>	<code>flushleft</code>	<code>flushright</code>
<i>command:</i>	<code>\centering</code>	<code>\raggedright</code>	<code>\raggedleft</code>

The commands neither start a new paragraph nor add vertical space, unlike the corresponding environments. Hence, the commands can be used inside other environments and inside a `\parbox`, in particular, to control the alignment in `p` columns of an `array` or `tabular` environment. Note, however, that if they are used in the last column of a `tabular` or `array` environment, the `\backslash` is no longer available to denote the end of a row. Instead, the command `\tabularnewline` can be used for this purpose (see also Section 5.2.1).

### 3.1.12 ragged2e—Enhancing justification

The previous subsection discussed the deficiencies of L<sup>A</sup>T<sub>E</sub>X's `flushleft` and `flushright` environments. The package `ragged2e` written by Martin Schröder sets out to provide alternatives that do not produce such extreme raggedness. This venture is not quite as simple as it sounds, because it is not enough to set `\rightskip` to something like `0pt plus 2em`. Notwithstanding the fact that this would result in T<sub>E</sub>X trying hard to keep the line endings within the `2em` boundary, there remains a subtle problem: by default, the interword space is also stretchable for most fonts. Thus, if `\rightskip` has only finite stretchability, T<sub>E</sub>X will distribute excess space equally to all spaces. As a result, the interword spaces will have different width, depending on the amount of material in the line. The solution is to redefine the interword space so that it no longer can stretch or shrink by specifying a suitable (font-dependent) value for `\spaceskip`. This internal T<sub>E</sub>X parameter, if nonzero, represents the current interword space, overwriting the default that is defined by the current font.

By default, the package does not modify the standard L<sup>A</sup>T<sub>E</sub>X commands and environments discussed in the previous section, but instead defines its own using the same names except that some letters are uppercased.<sup>1</sup> The new environments and commands are given in the following correspondence table:

<i>environment:</i>	<code>Center</code>	<code>FlushLeft</code>	<code>FlushRight</code>
<i>command:</i>	<code>\Centering</code>	<code>\RaggedRight</code>	<code>\RaggedLeft</code>

They differ from their counterparts of the previous section not only in the fact that they try to produce less ragged output, but also in their attempt to provide additional flexibility by easily letting you change most of their typesetting aspects.

As typing the mixed-case commands and environments is somewhat tedious, you can overload the original commands and environments, such as `\raggedright`, with the new definitions by supplying the `newcommands` option when loading the package.

*Overloading the  
original commands*

The package offers a large number of parameters to define the exact behavior of the new commands and environments (see Table 3.2 on the next page). For `\RaggedRight` or `FlushLeft` the white space added at the right of each line can be specified as `\RaggedRightRightskip`, the one at the left can be specified as `\RaggedRightLeftskip`, the paragraph indentation to use is available as `\RaggedRightParindent`, and even the space added to fill the last line is available as `\RaggedRightParfillskip`. Similarly, the settings for `\Centering` and `\RaggedLeft` can be altered; just replace `RaggedRight` in the parameter names with either `Centering` or `RaggedLeft`.

To set a whole document unjustified, specify `document` as an option to the `ragged2e` package. For the purpose of justifying individual paragraphs the

*Unjustified setting  
as the default*

<sup>1</sup>This is actually against standard naming conventions. In most packages mixed-case commands indicate interface commands to be used by designers in class files or in the preamble, but not commands to be used inside documents.

\RaggedLeftParindent	Opt	\RaggedLeftLeftskip	Opt plus 2em
\RaggedLeftRightskip	Opt	\RaggedLeftParfillskip	Opt
\CenteringParindent	Opt	\CenteringLeftskip	Opt plus 2em
\CenteringRightskip	Opt plus 2em	\CenteringParfillskip	Opt
\RaggedRightParindent	Opt	\RaggedRightLeftskip	Opt
\RaggedRightRightskip	Opt plus 2em	\RaggedRightParfillskip	Opt plus 1fil
\JustifyingParindent	1 em	\JustifyingParfillskip	Opt plus 1fil

Table 3.2: Parameters used by `ragged2e`

package offers the command `\justifying` and the environment `justify`. Both can be customized using the length parameters `\JustifyingParindent` and `\JustifyingParfillskip`.

Thus, to produce a document with a moderate amount of raggedness and paragraphs indented by 12pt, you could use a setting like the one in the following example (compare it to Example 3-1-48 on page 104).

“The `\LaTeX` document preparation system is a special version of Donald Knuth’s `\TeX` program. `\TeX` is a sophisticated program designed to produce high-quality typesetting, especially for mathematical text.”

```
\usepackage[document]{ragged2e}
\setlength{\RaggedRightRightskip}{0pt plus 1cm}
\setlength{\RaggedRightParindent}{12pt}
‘‘The \LaTeX{} document preparation system is a special
version of Donald Knuth’s \TeX{} program. \TeX{} is a
sophisticated program designed to produce high-quality
typesetting, especially for mathematical text.’’
```

3-1-

*Unjustified settings in narrow columns* In places with narrow measures (e.g., `\marginpar`, `\parbox`, `\minipage` environments, or p-columns of `tabular` environments), the justified setting usually produces inferior results. With the option `raggedrightbox`, paragraphs in such places are automatically typeset using `\RaggedRight`. If necessary, `\justifying` can be used to force a justified paragraph in individual cases.

*The default values* The use of `em` values in the defaults (see Table 3.2) means that special care is needed when loading the package, as the `em` is turned into a real dimension at this point! The package should therefore be loaded *after* the body font and size have been established—for example, after font packages have been loaded.

Instead of using the defaults listed in Table 3.2, one can instruct the package to mimic the original `\TeX` definitions by loading it with the option `originalparameters` and then changing the parameter values as desired.

### 3.1.13 `setspace`—Changing interline spacing

The `\baselineskip` command is `\TeX`’s parameter for defining the *leading* (normal vertical distance) between consecutive baselines. Standard `\LaTeX` defines a leading approximately 20% larger than the design size of the font (see Section 7.9.1 on

page 413). Because it is not recommended to change the setting of `\baselineskip` directly, L<sup>A</sup>T<sub>E</sub>X provides the `\baselinestretch` command to allow for changing `\baselineskip` at all sizes globally.

Be aware that after the `\renewcommand{\baselinestretch}{1.5}` command is issued, the leading will not increase immediately. A font size changing command (e.g., `\small`, `\Large`) must be executed to make the new value take effect.

The package `setspace` (by Geoffrey Tobin and others) provides commands and environments for typesetting with variable spacing (primarily double and one-and-a-half). Three commands—`\singlespacing`, `\onehalfspacing`, and `\doublespacing`—are available for use in the preamble to set the overall spacing for the document. Alternatively, a different spacing value can be defined by placing a `\setstretch` command in the preamble. It takes the desired spacing factor as a mandatory argument. In the absence of any of the above commands, the default setting is single spacing.

To change the spacing inside a document three specific environments—`singlespace`, `onehalfspace`, and `doublespace`—are provided. They set the spacing to single, one-and-a-half, and double spacing, respectively. These environments cannot be nested.

In the beginning God created the heaven and the earth. Now the earth was unformed and void, and darkness was upon the face of the deep; and the spirit of God hovered over the face of the waters.

3-1-50

```
\usepackage{setspace}
\begin{doublespace}
In the beginning God created the heaven
and the earth. Now the earth was unformed
and void, and darkness was upon the face
of the deep; and the spirit of God
hovered over the face of the waters.
\end{doublespace}
```

For any other spacing values the generic environment `spacing` should be used. Its mandatory parameter is the value of `\baselinestretch` for the text enclosed by the environment.

In the beginning God created the heaven and the earth. Now the earth was unformed and void, and darkness was upon the face of the deep; and the spirit of God hovered over the face of the waters.

3-1-51

```
\usepackage{setspace}
\begin{spacing}{2.0}
In the beginning God created the heaven
and the earth. Now the earth was unformed
and void, and darkness was upon the face
of the deep; and the spirit of God
hovered over the face of the waters.
\end{spacing}
```

In the above example the coefficient “2.0” produces a larger leading than the “double spacing” (`doublespace` environment) required for some publications. With the `spacing` environment the leading is effectively increased twice—once by `\baselineskip` (which L<sup>A</sup>T<sub>E</sub>X already sets to about 20% above the font size) and a second time by setting `\baselinestretch`. “Double spacing” means that the vertical distance between baselines is about twice as large as the font size.

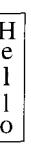
<i>spacing</i>	10pt	11pt	12pt
one and one-half	1.25	1.21	1.24
double	1.67	1.62	1.66

Table 3.3: Effective `\baselinestretch` values for different font sizes

Since `\baselinestretch` refers to the ratio between the desired distance and the `\baselineskip`, the values of `\baselinestretch` for different document base font sizes (and at two different optical spacings) can be calculated and are presented in Table 3.3.

### 3.1.14 `picinpar`—Making rectangular holes

The package `picinpar` (created by Friedhelm Sowa based on earlier work by Alan Hoenig) allows “windows” to be typeset inside paragraphs. The basic environment is `window`. It takes one mandatory argument specified in contrast to L<sup>A</sup>T<sub>E</sub>X conventions in square brackets, in the form of a comma-separated list of four elements. These elements are the number of lines before the window starts; the alignment of the window inside the paragraph (`l` for left, `c` for centered, and `r` for right); the material shown in the window; and explanatory text about the contents in the window (e.g., the caption).

In this case we center a word printed vertically inside the paragraph.  It is not difficult to understand that tables can also be easily included with the `tabwindow` environment. When a paragraph ends, like here, and the window is not yet finished, then it just continues past the paragraph boundary, right into the next one(s).

```
\usepackage{picinpar}
\begin{window}[1,c,%
  \fbox{\shortstack{H\\e\\l\\o}}]
In this case we center a word printed
vertically inside the paragraph. It is not
difficult to understand that tables can also
be easily included with the \texttt{tabwindow}
environment.\par When a paragraph ends, like
here, and the window is not yet finished,
then it just continues past the paragraph
boundary, right into the next one(s).
\end{window}
```

3-1-52

If you look at the above example you will notice that the second paragraph is not properly indented. You can fix this defect by requesting an explicit indentation using `\par\indent`, if necessary.

Centering a window as in the previous example works only if the remaining text width on either side is still suitably wide (where “suitably” means larger than one inch). Otherwise, the package will simply fill it with white space.

The package also provides two variant environments, `figwindow` and `tabwindow`. They can format the explanatory text as a caption, by adding a caption number. You should, however, be careful when mixing such “nonfloating”

floats with standard `figure` or `table` environments, because the latter might get deferred and this way mess up the numbering of floats.

The next example shows such an embedded figure—a map of Great Britain placed inside a paragraph. Unfortunately, the caption formatting is more or less hard-wired into the package; if you want to change it, you have to modify an internal command named `\@makewincaption`.

Is this a dagger which I see before me, The handle toward my hand? Come, let me clutch thee. I have thee not, and yet I see thee still. Art thou not, fatal vision, sensible To feeling as to sight? or art thou but A dagger of the mind, a false creation, Proceeding from the heat-oppressed brain? I see thee yet, in form as palpable As this which now I draw. Thou marshall'st me the way that I was going;

And such an instrument I was to use. Mine eyes are made the fools o' the other senses, Or else worth all the rest; I see thee still, And on thy blade and dudgeon gouts of blood, Which was not so before. (3-1-53) (*Macbeth*, Act II, Scene 1).



**Figure 1:** United Kingdom  
Is this a dagger which I see before me, The handle toward my hand? Come, let me clutch thee. I have thee not, and yet I see thee still. Art thou not, fatal vision, sensible To feeling as to sight? or art thou but A dagger of the mind, a false creation, Proceeding from the heat-oppressed brain? I see thee yet, in form as palpable As this which now I draw. Thou marshall'st me the way that I was going;

```
\usepackage{picinpar,graphicx}
\begin{figwindow}[3,1,%
 \fbox{\includegraphics[width=30mm]{ukmap}},%
 {United Kingdom}]
Is this a dagger which I see before me, The handle toward my hand? Come, let me clutch thee. I have thee not, and yet I see thee still. Art thou not, fatal vision, sensible To feeling as to sight? or art thou but A dagger of the mind, a false creation, Proceeding from the heat-oppressed brain? I see thee yet, in form as palpable As this which now I draw. Thou marshall'st me the way that I was going; And such an instrument I was to use. Mine eyes are made the fools o' the other senses, Or else worth all the rest; I see thee still, And on thy blade and dudgeon gouts of blood, Which was not so before. (\emph{Macbeth}, Act II, Scene 1).
\end{figwindow}
```

## 3.2 Footnotes, endnotes, and marginals

`LATEX` has facilities to typeset “inserted” text, such as marginal notes, footnotes, figures, and tables. The present section looks more closely at different kinds of notes, while Chapter 6 describes floats in more detail.

We start by discussing the possibilities offered through standard `LATEX`'s footnote commands and explain how (far) they can be customized. For two-column documents, a special layout for footnotes is provided by the `ftnright` package, which moves all footnotes to the bottom of the right column. This is followed by a presentation of the `footmisc` package, which overcomes most of the limitations of the standard commands and offers a wealth of additional features. The `manyfoot` package (which can be combined with `footmisc`) extends the footnote support for disciplines like linguistics by providing several independent footnote commands.

Support for endnotes is provided through the package `endnotes`, which allows for mixing footnotes and endnotes and can also be used to provide chapter

notes, as required by some publishers. The section concludes with a discussion of marginal notes, which are already provided by standard L<sup>A</sup>T<sub>E</sub>X.

### 3.2.1 Using standard footnotes

A sharp distinction is made between footnotes in the main text and footnotes inside a `minipage` environment. The former are numbered using the `footnote` counter, while inside a `minipage` the `\footnote` command is redefined to use the `mpfootnote` counter. Thus, the representation of the footnote mark is obtained by the `\thefootnote` or the `\thempfootnote` command depending on the context. By default, it typesets an Arabic number in text and a lowercase letter inside a `minipage` environment. You can redefine these commands to get a different representation by specifying, for example, footnote symbols, as shown in the next example.

text text text\* text text<sup>†</sup> text.

\*The first

<sup>†</sup>The second

```
\renewcommand{\thefootnote}
  {\fnsymbol{footnote}}
\text{ text text text}\footnote{The first}
\text{ text text}\footnote{The second} text.
```

3-2-1

*Peculiarities inside a `minipage`* Footnotes produced with the `\footnote` command inside a `minipage` environment use the `mpfootnote` counter and are typeset at the bottom of the `minipage` produced by the `minipage`. However, if you use the `\footnotemark` command in a `minipage`, it will produce a footnote mark in the same style and sequence as the main text footnotes—that is, stepping the footnote counter and using the `\thefootnote` command for the representation. This behavior allows you to produce a footnote inside your `minipage` that is typeset in sequence with the main text footnotes at the bottom of the page: you place a `\footnotemark` inside the `minipage` and the corresponding `\footnotetext` after it.

... main text ...

Footnotes in a `minipage` are numbered using lowercase letters.<sup>a</sup>

This text references a footnote at the bottom of the page.<sup>1</sup> And another<sup>b</sup> note.

<sup>a</sup>Inside `minipage`

<sup>b</sup>Inside again

<sup>1</sup>At bottom of page

```
\noindent\ldots{} main text \ldots
\begin{center}
\begin{minipage}{.7\linewidth}
Footnotes in a minipage are numbered using
lowercase letters.\footnote{Inside minipage}
\par This text references a footnote at the
bottom of the page.\footnotemark{}
And another\footnote{Inside again} note.
\end{minipage}\footnotetext{At bottom of page}
\end{center}
\ldots{} main text \ldots
```

3-2-2

As the previous example shows, if you need to reference a `minipage` footnote several times, you cannot use `\footnotemark` because it refers to footnotes type-

set at the bottom of the page. You can, however, load the package `footmisc` and then use `\mpfootnotemark` in place of `\footnotemark`. Just like `\footnotemark`, the `\mpfootnotemark` command first increments its counter and then displays its value. Thus, to refer to the previous value you typically have to decrement it first, as shown in the next example.

Main text ...

Footnotes in a minipage are numbered using lowercase letters.<sup>a</sup>  
This text references the previous footnote.<sup>a</sup> And another<sup>b</sup> note.

<sup>a</sup>Inside minipage

<sup>b</sup>Inside as well

3-2-3

```
\usepackage{footmisc}
\noindent Main text \ldots \begin{center}
\begin{minipage}{.7\linewidth}
Footnotes in a minipage are numbered using
lowercase letters.\footnote{Inside minipage}
\par This text references the previous
footnote.\addtocounter{\mpfootnote}{-1}%
` \mpfootnotemark{}\\
And another\footnote{Inside as well} note.
\end{minipage}
\end{center} \ldots main text \ldots
```

LATEX does not allow you to use a `\footnote` inside another `\footnote` command, as is common in some disciplines. You can, however, use the `\footnotemark` command inside the first footnote and then put the text of the footnote's footnote as the argument of a `\footnotetext` command. For other special footnote requirements consider using the `manyfoot` package (described below).

Some<sup>1</sup> text and some more text.

<sup>1</sup>A sample<sup>2</sup> footnote.

<sup>2</sup>A subfootnote.

3-2-4

```
Some\footnote{A sample\footnotemark{}}
footnote.}\footnotetext{A subfootnote.}
text and some more text.
```

What if you want to reference a given footnote? You can use LATEX's normal `\label` and `\ref` mechanism, although you may want to define your own command to typeset the reference in a special way. For instance:

This is some text.<sup>1</sup>  
... as shown in footnote (1) on page 6,...

3-2-5

<sup>1</sup>Text inside referenced footnote.

```
\newcommand\fnref[1]{\unskip~(\ref{#1})}
This is some text.\footnote{Text inside
referenced footnote\label{fn:myfoot}.}\par
\ldots as shown in footnote\fnref{fn:myfoot}
on page~\pageref{fn:myfoot},\ldots
```

Standard LATEX does not allow you to construct footnotes inside tabular material. Section 5.8 describes several ways of tackling that problem.

### 3.2.2 Customizing standard footnotes

Footnotes in L<sup>A</sup>T<sub>E</sub>X are generally simple to use and provide a quite powerful mechanism to typeset material at the bottom of a page.<sup>1</sup> This material can consist of several paragraphs and can include lists, inline or display mathematics, tabular material, and so on.

L<sup>A</sup>T<sub>E</sub>X offers several parameters to customize footnotes. They are shown schematically in Figure 3.1 on the next page and are described below:

`\footnotesize` The font size used inside footnotes (see also Table 7.1 on page 342).

`\footnotesep` The height of a strut placed at the beginning of every footnote. If it is greater than the `\baselineskip` used for `\footnotesize`, then additional vertical space will be inserted above each footnote. See Appendix A.2.3 for more information about struts.

`\skip\footins` A low-level T<sub>E</sub>X length parameter that defines the space between the main text and the start of the footnotes. You can change its value with the `\setlength` or `\addtolength` command by putting `\skip\footins` into the first argument:

```
\addtolength{\skip\footins}{10mm plus 2mm}
```

`\footnoterule` A macro to draw the rule separating footnotes from the main text that is executed right after the vertical space of `\skip\footins`. It should take zero vertical space; that is, it should use a negative skip to compensate for any positive space it occupies. The default definition is equivalent to the following:

```
\renewcommand\footnoterule{\vspace*{-3pt}%
\hrule width 2in height 0.4pt \vspace*{2.6pt}}
```

Note that T<sub>E</sub>X's `\hrule` command and not L<sup>A</sup>T<sub>E</sub>X's `\rule` command is used. Because the latter starts a paragraph, it would be difficult to calculate the spaces needed to achieve a net effect of zero height. For this reason producing a fancier "rule" is perhaps best done by using a zero-sized picture environment to position the rule object without actually adding vertical space.

In the `report` and `book` classes, footnotes are numbered inside chapters; in `article`, footnotes are numbered sequentially throughout the document. You can change the latter default by using the `\@addtoreset` command (see Appendix A.1.4). However, do not try to number your footnotes within pages with

<sup>1</sup>An interesting and complete discussion of this subject appeared in the French T<sub>E</sub>X Users' Group magazine *Cahiers GUTenberg* [10, 133].

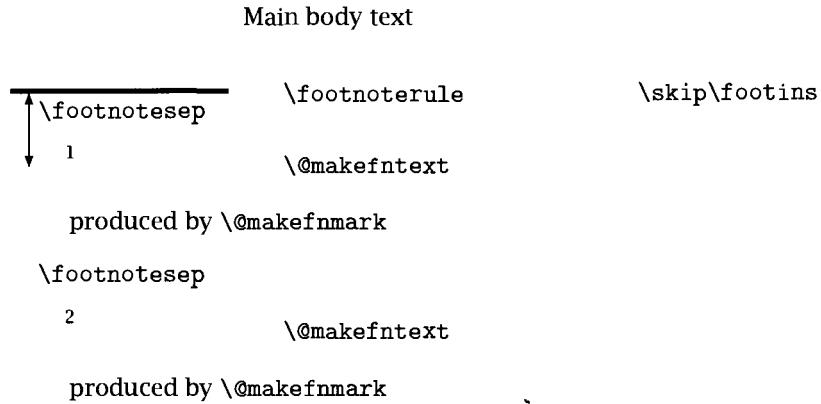


Figure 3.1: Schematic layout of footnotes

the help of this mechanism. L<sup>A</sup>T<sub>E</sub>X is looking ahead while producing the final pages, so your footnotes would most certainly be numbered incorrectly. To number footnotes on a per-page basis, use the `footmisc` or `perpage` package (described below).

The command `\@makefnmark` is normally used to generate the footnote mark. One would expect this command to take one argument (the current footnote number), but in fact it takes none. Instead, it uses the command `\@thefnmark` to indirectly refer to that number. The reason is that depending on the position (inside or outside of a `minipage`) a different counter needs to be accessed. The definition, which by default produces a superscript mark, looks roughly as follows:

```
\renewcommand{\@makefnmark}
  {\mbox{\textsuperscript{\normalfont\@thefnmark}}}
```

The `\footnote` command executes `\@makefntext` inside a `\parbox`, with a width of `\columnwidth`. The default version looks something like:

```
\newcommand{\@makefntext}[1]
  {\noindent\makebox[1.8em][r]{\@makefnmark}#1}
```

This will place the footnote mark right aligned into a box of width `1.8em` directly followed by the footnote text. Note that it reuses the `\@makefnmark` macro, so any change to it will, by default, modify the display of the mark in both places. If you want the text set flush left with the number placed into the margin, then you could use the redefinition shown in the next example. Here we do not use `\@makefnmark` to format the mark, but rather access the number via `\@thefnmark`. As a result,

the mark is placed onto the baseline instead of being raised. Thus, the marks in the text and at the bottom are formatted differently.

```
\makeatletter
\renewcommand\@makefntext[1]%
  {\noindent\makebox[0pt][r]{\@thefnmark.\,}\#1}
\makeatother
text text text1 text text2 text.
1. The first
2. The second
```

3-2-6

### 3.2.3 ftnright—Right footnotes in a two-column environment

It is sometimes desirable to group all footnotes in a two-column document at the bottom of the right column. This can be achieved by specifying the `ftnright` package written by Frank Mittelbach. The effect of this package is shown in Figure 3.2 on the facing page—the first page of the original documentation (including its spelling errors) of the `ftnright` implementation. It is clearly shown how the various footnotes collect in the lower part of the right-hand column.

The main idea for the `ftnright` package is to assemble the footnotes of all columns on a page and place them all together at the bottom of the right column. The layout produced allows for enough space between footnotes and text and, in addition, sets the footnotes in smaller type.<sup>1</sup> Furthermore, the footnote markers are placed at the baseline instead of raising them as superscripts.<sup>2</sup>

This package can be used together with most other class files for L<sup>A</sup>T<sub>E</sub>X. Of course, the `ftnright` package will take effect only with a document using a two-column layout specified with the `twocolumn` option on the `\documentclass` command. In most cases, it is best to use `ftnright` as the very last package to make sure that its settings are not overwritten by other options.

### 3.2.4 footmisc—Various footnotes styles

Since standard L<sup>A</sup>T<sub>E</sub>X offers only one type of footnotes and only limited (and somewhat low-level) support for customization, several people developed small packages that provided features otherwise not available. Many of these earlier efforts were captured by Robin Fairbairns in his `footmisc` package, which supports, among other things, page-wise numbering of footnotes and footnotes formatted as a single paragraph at the bottom of the page. In this section we describe the features provided by this package, showing which packages it supersedes whenever applicable.

<sup>1</sup>Some journals use the same size for footnotes and text, which sometimes makes it difficult to distinguish footnotes from the main text.

<sup>2</sup>Of course, this is done only for the mark preceding the footnote text and not the one used within the main text, where a raised number or symbol set in smaller type will help to keep the flow of thoughts uninterrupted.

## Footnotes in a multi-column layout\*

Frank Mittelbach

August 10, 1991

### 1 Introduction

The placement of footnotes in a multi-column layout always bothered me. The approach taken by *L<sup>A</sup>T<sub>E</sub>X* (i.e., placing the footnotes separately under each column) might be all right if nearly no footnotes are present. But it looks clumsy when both columns contain footnotes, especially when they occupy different amounts of space.

In the multi-column style option [5], I used page-wide footnotes at the bottom of the page, but again the result doesn't look very pleasant since short footnotes produce undesired gaps of white space. Of course, the main goal of this style option was a balancing algorithm for columns which would allow switching between different numbers of columns on the same page. With this feature, the natural place for footnotes seems to be the bottom of the page<sup>1</sup> but looking at some of the results it seems best to avoid footnotes in such a layout entirely.

Another possibility is to turn footnotes into endnotes, i.e., printing them at the end of every chapter or the end of the entire document. But I assume everyone who has ever read a book using such a layout will agree with me, that it is a pain to search back and forth, so that the reader is tempted to ignore the endnotes entirely.

When I wrote the article about "Future extensions of *TeX*" [6] I was again dissatisfied with the outcome of the footnotes, and since this article should show certain aspects of high quality typesetting, I decided to give the footnote problem a try and modified the *L<sup>A</sup>T<sub>E</sub>X* output routine for this purpose. The layout I used was inspired by the yearbook of the Gutenberg Gesellschaft Mainz [1]. Later on, I found that it is also recommended by Jan White [9]. On the layout of footnotes I also consulted books by Jan Tschichold [8] and Manfred Simoneit [7], books I would recommend to everyone being able to read German texts.

#### 1.1 Description of the new layout

The result of this effort is presented in this paper and the reader can judge for himself whether it was successful or not.<sup>2</sup> The main idea for this layout is to assemble the footnotes of all columns on a page and place them all

together at the bottom of the right column. Allowing for enough space between footnotes and text, and in addition, setting the footnotes in smaller type<sup>3</sup> I decided that one could omit the footnote separator rule which is used in most publications prepared with *TeX*.<sup>4</sup> Furthermore, I decided to place the footnote markers<sup>5</sup> at the baseline instead of raising them as superscripts.<sup>6</sup>

All in all, I think this generates a neat layout, and surprisingly enough, the necessary changes to the *L<sup>A</sup>T<sub>E</sub>X* output routine are nevertheless astonishingly simple.

#### 1.2 The use of the style option

This style option might be used together with any other style option for *L<sup>A</sup>T<sub>E</sub>X* which does not change the three internals changed by *ftnright.sty*.<sup>7</sup> In most cases, it is best to use this style option as the very last option in the \documentstyle command to make sure that its settings are not overwritten by other options.<sup>8</sup>

\*. The *L<sup>A</sup>T<sub>E</sub>X* style option *ftnright* which is described in this article has the version number v1.0d dated 92/06/19. The documentation was last revised on 92/06/19.

1. You can not use column footnotes at the bottom, since the number of columns can differ on one page.

2. Please note, that this option only changed the placement of footnotes. Since this article also makes use of the *doc* option [4], that assigns tiny numbers to code lines sprinkled throughout the text, the resulting design is not perfect.

3. The standard layout in *TUGboat* uses the same size for footnotes and text, giving the footnotes, in my opinion, much too much prominence.

4. People who prefer the rule can add it by redefining the command \footnoterule [2, p. 156]. Please, note that this command should occupy no space, so that a negative space should be used to compensate for the width of the rule used.

5. The tiny numbers or symbols, e.g., the '5' in front of this footnote.

6. Of course, this is only done for the mark preceding the footnote text and not the one used within the main text where a raised number or symbol set in smaller type will help to keep the flow of thoughts, uninterrupted.

7. These are the macros \startcolumn, \makecol and \outputdblcol as we will see below. Of course, the option will take only effect with a document style using a twocolumn layout (like *tugboat*) or when the user additionally specifies *twocolumn* as a document style option in the \documentstyle command.

8. The *l tugboat* option (which is currently set up as a style option instead of a document style option which it actually is) will overwrite

Figure 3.2: The placement of text and footnotes with the *ftnright* package

The interface for *footmisc* is quite simple: nearly everything is customized by specifying options when the package is loaded, though in some cases further control is possible via parameters.

In the *article* class, footnotes are numbered sequentially throughout the document; in *report* and *book*, footnotes are numbered inside chapters. Sometimes,

however, it is more appropriate to number footnotes on a per-page basis. This can be achieved by loading `footmisc` with the option `perpage`. The package `footnpag` (by Joachim Schrod) provides the same feature with a somewhat different implementation as a stand-alone package. A generalized implementation for resetting counters on a per-page basis is provided by the package `perpage` (see Section 3.2.5 on page 120). Since `TeX`'s page-building mechanism is asynchronous, it is always necessary to process the document at least twice to get the numbering correct. Fortunately, the package warns you via "Rerun to get cross-references right" if the footnote numbers are incorrect. The package stores information between runs in the `.aux` file, so after a lot of editing this information is sometimes not even close to reality. In such a case deleting the `.aux` file helps the package to find the correct numbering faster.<sup>1</sup>

Some text* with a footnote. More† text.  <small>*First. †Second.</small>	Even more text.* And even† more text. Some  <small>*Third. †Fourth.</small>	<pre>\usepackage[perpage,symbol]{footmisc} Some text\footnote{First.} with a footnote. More\footnote{Second.} text. Even more text.\footnote{Third.} And even\footnote {Fourth.} more text. Some final text.</pre>
---	--	--

*Counter too large  
errors*

For this special occasion our example shows two pages side by side, so you can observe the effects of the `perpage` option. The example also shows the effect of another option: `symbol` will use footnote symbols instead of numbers. As only a limited number of such symbols are available, you can use this option only if there are few footnotes in total or if footnote numbers restart on each page. There are six different footnote symbols and, by duplicating some, standard `TeX` supports nine footnotes. By triplicating some of them, `footmisc` supports up to 16 footnotes (per page or in total). If this number is exceeded you will get a `TeX` error message.

In particular with the `perpage` option, this behavior can be a nuisance because the error could be spurious, happening only while the package is still trying to determine which footnotes belong on which page. To avoid this problem, you can use the variant option `symbol*`, which also produces footnote symbols but numbers footnotes for which there are no symbols left with Arabic numerals. In that case you will get a warning at the end of the run that some footnotes were out of range and detailed information is placed in the transcript file.

`\setfnsymbol{name}`    `\DefineFNsymbols*[name][type]{symbol-list}`

If the `symbol` or `symbol*` option is selected, a default sequence of footnote symbols defined by Leslie Lamport is used. Other authorities suggest different se-

<sup>1</sup>In fact, during the preparation of this chapter we managed to confuse `footmisc` (by changing the `\textheight` in an example) so much that it was unable to find the correct numbering thereafter and kept asking for a rerun forever. Removing the `.aux` file resolved the problem.

```

lamport    * † ‡ § ¶ || ** †† ‡‡ $§ ¶¶ *** ††† ‡‡‡ $$$ ¶¶¶
bringhurst * † ‡ § ¶|| ¶
chicago    * † ‡ § ¶|| #
wiley      * ** † ‡ § ¶|| 

```

Table 3.4: Footnote symbol lists predefined by `footmisc`

quences, so `footmisc` offers three other sequences to chose from using the declaration `\setfnsymbol` (see Table 3.4).

In addition, you can define your own sequence using the `\DefineFNsymbols` declaration in the preamble. It takes two mandatory arguments: the *name* to access the list later via `\setfnsymbol` and the *symbol-list*. From this list symbols are taken one after another (with spaces ignored). If a symbol is built from more than one glyph, it has to be surrounded by braces. If the starred form of the declaration is used, `LATEX` issues an error message if it runs out of symbols. Without it, you will get Arabic numerals and a warning at the end of the `LATEX` run.

Due to an unfortunate design choice, footnote symbols (as well as some other text symbols) were originally added to the math fonts of `TEX`, rather than to the text fonts, with the result that they did not change when the text font was modified. In `LATEX` this flaw was partly corrected by adding these symbols to the text symbol encoding (TS1; see Section 7.5.4). However, for compatibility reasons the footnote symbols are still taken by default from the math fonts, even though this choice is not appropriate if one has changed the text font from Computer Modern to some other typeface. By using the optional *type* argument with the value `text`, you can tell `footmisc` that your list consists of text symbols. Note that all predefined symbol lists consist of math symbols and may need redeclaring if used with fonts other than Computer Modern.

Some text\* with a footnote. More\*\* text.  
 Even more text.\*\*\* And even\*\*\*\* more text.  
 Some more text to finish up.

\*First.  
 \*\*Second.  
 \*\*\*Third.  
 \*\*\*\*Fourth.

3-2-8

```

\usepackage[symbol]{footmisc}
\DefineFNsymbols{stars}{text}{* {**} {***} {****}}
\setfnsymbol{stars}
Some text\footnote{First.} with a footnote.
More\footnote{Second.} text. Even more
text.\footnote{Third.} And even\footnote{Fourth.}
more text. Some more text to finish up.

```

If you have many short footnotes then their default placement at the bottom of the page, stacked on top of each other, is perhaps not completely satisfactory. A typical example would be critical editions, which contain many short footnotes.<sup>1</sup> The layout of the footnotes can be changed using the `para` option, which formats

<sup>1</sup>See, for example, the `ledmac` package [171] for the kinds of footnotes and endnotes that are common in critical editions. This package is a reimplementation of the `EDMAC` system [112] for `LATEX` and was recently made available by Peter Wilson. See also the `bigfoot` package by David Kastrup.

/ them into a single paragraph. If this option is chosen then footnotes never split across pages. The code for this option is based on work by Chris Rowley and Dominik Wujastyk (available as the package `fnpa`), which in turn was inspired by an example in *The TeXbook* by Donald Knuth.

Some text with a footnote.<sup>1</sup> More text.<sup>2</sup> Even more text.<sup>3</sup> Some final text.

<sup>1</sup> A first. <sup>2</sup> A second. <sup>3</sup> A third.

```
\usepackage[para]{footmisc}
Some text with a footnote.\footnote{A first.}
More text.\footnote{A second.} Even more
text.\footnote{A third.} Some final text.
```

3-2-9

Another way to deal with footnotes is given by the option `side`. In this case footnotes are placed into the margin, if possible on the same line where they are referenced. What happens internally is that special `\marginpar` commands are used to place the footnote text, so everything said in Section 3.2.8 about the `\marginpar` commands is applicable. This option cannot be used together with the `para` option, described earlier, but can be combined with most others.

<sup>1</sup> A first.

<sup>2</sup> A second.

<sup>3</sup> A third.

<sup>4</sup> A fourth.

Some text with a footnote.<sup>1</sup> A lot of additional text here with a footnote.<sup>2</sup> Even more text and then another footnote.<sup>3</sup> Some more text.<sup>4</sup> A lot of additional lines of text here to fill up the space on the left.

```
\usepackage[side,flushmargin]{footmisc}
Some text with a footnote.\footnote{A first.}
A lot of additional text here with a
footnote.\footnote{A second.}
Even more text and then another
footnote.\footnote{A third.}
Some more text.\footnote{A fourth.} A lot of
additional lines of text here to fill up the
space on the left.
```

3-2-10

The option `flushmargin` used in the previous example makes the footnote text start at the left margin with the footnote marker protruding into the margin; by default, the footnote text is indented. For obvious reasons this option is incompatible with the `para` option. A variant form is called `marginal`. If this option is used then the marker sticks even farther into the margin, as shown in the example below.

Some text<sup>1</sup> with a footnote. More text.<sup>2</sup> Even more text.<sup>3</sup> Some final text.

<sup>1</sup> A first.

<sup>2</sup> A second.

<sup>3</sup> A third.

```
\usepackage[marginal]{footmisc}
Some text\footnote{A first.} with a
footnote. More text.\footnote{A second.}
Even more text.\footnote{A third.} Some
final text.
```

3-2-11

Instead of using one of the above options, the position of the footnote marker can be directly controlled using the parameter `\footnotemargin`. If set to a negative value the marker is positioned in the margin. A value of `0pt` is equivalent to using the option `flushmargin`. A positive value means that the footnote text

is indented by this amount and the marker is placed flush right in the space produced by the indentation.

Some text<sup>1</sup> with a footnote. More text.<sup>2</sup> Even more text.<sup>3</sup> Some final text.

<sup>1</sup>A first.

<sup>2</sup>A second.

<sup>3</sup>A third.

3-2-12

```
\usepackage{footmisc}
\setlength\footnotemargin{10pt}
Some text\footnote{A first.} with a
footnote. More text.\footnote{A second.}
Even more text.\footnote{A third.} Some
final text.
```

By default, the footnote text is justified but this does not always give satisfactory results, especially with the options `para` and `side`. In case of the `para` option nothing can be done, but for other layouts you can switch to ragged-right typesetting by using the option `ragged`. The next example does not specify `flushmargin`, so we get an indentation of width `\footnotemargin`—compare this to Example 3-2-10 on the preceding page.

<sup>1</sup>In the margin  
ragged right often  
looks better.

3-2-13

Some text<sup>1</sup> with a footnote  
A lot of additional text here to  
fill up the space in the example.  
A lot of additional text here to  
fill up the space in the example.

```
\usepackage[side,ragged]{footmisc}
Some text\footnote{In the margin ragged
right often looks better.} with a footnote
A lot of additional text here to fill
up the space in the example. A lot of
additional text here to fill up the space
in the example.
```

The two options `norule` and `splitlerule` (courtesy of Donald Arseneau) modify the rule normally placed between text and footnotes. If `norule` is specified, then the separation rule will be suppressed. As compensation the value of `\skip\footins` is slightly enlarged. If a footnote does not fit onto the current page it will be split and continued on the next page, unless the `para` option is used (as it does not support split footnotes). By default, the rule separating normal and split footnotes from preceding text is the same. If you specify the option `splitlerule`, however, it becomes customizable: the rule above split footnotes will run across the whole column while the one above normal footnotes will retain the default definition given by `\footnoterule`. More precisely, this option will introduce the commands `\mpfootnoterule` (for use in minipages), `\pagefootnoterule` (for use on regular pages), and `\splitfootnoterule` (for use on pages starting with a split footnote). By modifying their definitions, similar to the example given earlier for the `\footnoterule` command, you can customize the layout according to your needs.

Some text with a footnote.<sup>1</sup> More text.<sup>2</sup> Even more text.<sup>3</sup> Some final text.

<sup>1</sup>A first. <sup>2</sup>A second. <sup>3</sup>A third.

3-2-14

```
\usepackage[norule,para]{footmisc}
Some text with a footnote.\footnote{A first.}
More text.\footnote{A second.} Even more
text.\footnote{A third.} Some final text.
```

In classes such as `article` or `report` in which `\raggedbottom` is in effect, so that columns are allowed to be of different heights, the footnotes are attached at a distance of `\skip\footins` from the column text. If you prefer them aligned at the bottom, so that any excess space is put between the text and the footnotes, specify the option `bottom`. In classes for which `\flushbottom` is in force, such as `book`, this option does nothing.

In some documents, e.g., literary analysis, several footnotes may appear at a single point. Unfortunately, L<sup>A</sup>T<sub>E</sub>X's standard footnote commands are not able to handle this situation correctly: the footnote markers are simply clustered together so that you cannot tell whether you are to look for the footnotes 1 and 2, or for the footnote with the number 12.

Some text<sup>1,2</sup> with two footnotes. Even more text.<sup>3</sup>

`\usepackage[para]{footmisc}`  
`Some text.\footnote{A first.}\footnote{A second.} with`  
`two footnotes. Even more text.\footnote{A third.}`

<sup>1</sup> A first. <sup>2</sup> A second. <sup>3</sup> A third.

3-2-15

This problem will be resolved by specifying the option `multiple`, which ensures that footnotes in a sequence will display their markers separated by commas. The separator can be changed to something else, such as a small space, by changing the command `\multfootsep`.

Some text<sup>1,2</sup> with two footnotes. Even more text.<sup>3</sup>

`\usepackage[multiple,para]{footmisc}`  
`Some text.\footnote{A first.}\footnote{A second.} with`  
`two footnotes. Even more text.\footnote{A third.}`

<sup>1</sup> A first. <sup>2</sup> A second. <sup>3</sup> A third.

3-2-16

The `footmisc` package deals with one other potential problem: if you put a footnote into a sectional unit, then it might appear in the table of contents or the running header, causing havoc. Of course, you could prevent this dilemma (manually) by using the optional argument of the heading command; alternatively, you could specify the option `stable`, which prevents footnotes from appearing in such places.

### 3.2.5 `perpage`—Resetting counters on a “per-page” basis

As mentioned earlier, the ability to reset arbitrary counters on a per-page basis is implemented in the small package `perpage` written by David Kastrup.

`\MakePerPage[start]{counter}`

The declaration `\MakePerPage` defines `counter` to be reset on every page, optionally requesting that its initial starting value be `start` (default 1). For demonstration

we repeat Example 3-2-7 on page 116 but start each footnote marker sequence with the second symbol (i.e., “†” instead of “\*”).

<p>Some text<sup>†</sup> with a footnote. More<sup>‡</sup> text.</p> <hr/> <p><sup>†</sup>First. <sup>‡</sup>Second.</p>	<p>Even more text.<sup>†</sup> And even<sup>‡</sup> more text. Some</p> <hr/> <p><sup>†</sup>Third. <sup>‡</sup>Fourth.</p>	<pre>\usepackage[symbol]{footmisc} \usepackage{perpage} \MakePerPage[2]{footnote} Some text\footnote{First.} with a footnote. More\footnote{Second.} text. Even more text.\footnote{Third.} And even\footnote {Fourth.} more text. Some final text.</pre>
--	---	---

The package synchronizes the numbering via the .aux file of the document, thus requiring at least two runs to get the numbering correct. In addition, you may get spurious “Counter too large” error messages on the first run if \fnsymbol or \alph is used for numbering (see the discussion of the symbol\* option for the footmisc package on page 116).

Among L<sup>A</sup>T<sub>E</sub>X’s standard counters probably only footnote can be sensibly modified in this way. Nevertheless, one can easily imagine applications that provide, say, numbered marginal notes, which could be defined as follows:

```
\newcounter{mnote}
\newcommand\mnote[1]{{{\refstepcounter{mnote}}%
\marginpar[\itshape\small\raggedleft\themnote.\ #1]%
[\itshape\small\raggedright\themnote.\ #1]}}
\usepackage{perpage} \MakePerPage{mnote}
```

We step the new counter mnote outside the \marginpar so that it is executed only once;<sup>1</sup> we also need to limit the scope of the current redefinition of \label (through \refstepcounter) so we put braces around the whole definition. Notes on left-hand pages should be right aligned, so we use the optional argument of \marginpar to provide different formatting for this case.

<p>1. First. Some text with a footnote. More<sup>1</sup> text.</p> <p>2. Third! Even more text. And</p> <hr/> <p><sup>1</sup>Second as footnote.</p>	<p>even more text. Some 1. Fourth. final text.<sup>2</sup></p> <hr/> <p><sup>2</sup>Fifth!</p>	<pre>% code as above Some text\mnote{First.} with a footnote. More\footnote{Second as footnote.} text. Even more text.\mnote{Third!} And even more\mnote{Fourth.} text. Some final text.\footnote{Fifth!}</pre>
--	--	---

Another application for the package is given in Example 3-2-24 on page 125, where several independent footnote streams are all numbered on a per-page basis.

<sup>1</sup>If placed in both arguments of \marginpar it would be executed twice. It would work if placed in the optional argument only, but then we would make use of an implementation detail (that the optional argument is evaluated first) that may change.

### 3.2.6 manyfoot—Independent footnotes

Most documents have only a few footnotes, if any. For them L<sup>A</sup>T<sub>E</sub>X's standard commands plus the enhancements offered by `footmisc` are usually sufficient. However, certain applications, such as critical editions, require several independently numbered footnote streams. For these situations the package `manyfoot` by Alexander Rozhenko can provide valuable help.<sup>1</sup>

```
\DeclareNewFootnote[fn-style]{suffix}[enum-style]
```

This declaration can be used to introduce a new footnote level. In its simplest form you merely specify a *suffix* such as "B". This allocates a new counter `footnote<suffix>` that is used to automatically number the footnotes on the new level. The default is to use Arabic numerals; by providing the optional argument *enum-style*, some other counter style (e.g., `roman` or `alph`) can be selected.

The optional *fn-style* argument defines the general footnote style for the new level; the default is `plain`. If the package was loaded with the `para` or `para*` option, then `para` can also be selected as the footnote style.

The declaration will then automatically define six commands for you. The first three are described here:

```
\footnote<suffix>[number]{text} Same as \footnote but for the new level.  
Steps the footnote<suffix> counter unless the optional number argument is given. Generates footnote markers and puts text at the bottom of the page.  
\footnotemark<suffix>[number] Same as \footnotemark but for the new level.  
Steps the corresponding counter (if no optional argument is used) and prints a footnote marker corresponding to its value.  
\footnotetext<suffix>[number]{text} Same as \footnotetext but for the new level. Puts text at the bottom of the page using the current value of footnote<suffix> or the optional argument to generate a footnote marker in front of it.
```

In all three cases the style of the markers depends on the chosen *enum-style*.

The remaining three commands defined by `\DeclareNewFootnote` for use in the document are `\Footnote<suffix>`, `\Footnotemark<suffix>`, and `\Footnotetext<suffix>` (i.e., same names as above but starting with an uppercase F). The important difference to the previous set is the following: instead of the optional *number* argument, they require a mandatory *marker* argument allowing you to specify arbitrary markers if desired. Some examples are given below.

The layout of the footnotes can be influenced by loading the `footmisc` package in addition to `manyfoot`, except that the `para` option of `footmisc` cannot be used. In the next example we use the standard footnote layout for top-level footnotes and the run-in layout (option `para`) for the second level. Thus, if all footnote levels should produce run-in footnotes, the solution is to avoid top-level footnotes

<sup>1</sup>A more comprehensive package, `bigfoot`, is currently being developed by David Kastrup.

completely (e.g., `\footnote`) and provide all necessary levels through `manyfoot`. Note how `footmisc`'s `multiple` option properly acts on all footnotes.

Some text<sup>1,a</sup> with footnotes. Even more text.<sup>b</sup> Some text<sup>2,\*</sup> with footnotes. Even more text.<sup>c</sup>

<sup>1</sup>A first.  
<sup>2</sup>Another main note.  
<sup>a</sup>B-level. <sup>b</sup>A second. <sup>\*</sup>A manual marker.  
<sup>c</sup>Another B note.

```
\usepackage[multiple]{footmisc}
\usepackage[para]{manyfoot}
\DeclareNewFootnote[para]{B}[alph]
Some text\footnote{A first.}\footnoteB{B-level.}
with footnotes. Even more text.\footnoteB{A second.}
Some text\footnote{Another main note.}%
\FootnoteB{*}{A manual marker.} with footnotes.
Even more text.\footnoteB{Another B note.}
```

3-2-19

In the following example the top-level footnotes are moved into the margin by loading `footmisc` with a different set of options. This time `manyfoot` is loaded with the option `para*`, which differs from the `para` option used previously in that it suppresses any indentation for the run-in footnote block. In addition, the second-level notes are now numbered with Roman numerals. For comparison the example typesets the same input text as Example 3-2-19 but it uses a different measure, as we have to show marginal notes now.

<sup>1</sup>A first. Some text<sup>1,i</sup> with footnotes.  
<sup>2</sup>Another main note. Even more text.<sup>ii</sup> Some text<sup>2,\*</sup> with footnotes. Even more text.<sup>iii</sup>

<sup>i</sup>B-level. <sup>ii</sup>A second. <sup>\*</sup>A manual marker.  
<sup>iii</sup>Another B note.

```
\usepackage[side,flushmargin,ragged,multiple]
[footmisc]
\usepackage[para*]{manyfoot}
\DeclareNewFootnote[para]{B}[roman]
Some text\footnote{A first.}\footnoteB{B-level.}
with footnotes. Even more text.\footnoteB{A
second.} Some text\footnote{Another main note.}%
\FootnoteB{*}{A manual marker.} with footnotes.
Even more text.\footnoteB{Another B note.}
```

3-2-20

The use of run-in footnotes, with either the `para` or the `para*` option, is likely to produce one particular problem: very long footnotes near a page break will not be split. To resolve this problem the `manyfoot` package offers a (semi)manual solution: at the point where you wish to split your note you place a `\SplitNote` command and end the footnote. You then place the remaining text of the footnote one paragraph farther down in the document in a `\Footnotetext<suffix>` using an empty `marker` argument.

Some<sup>1</sup> text with two footnotes.<sup>i</sup> More text.<sup>ii</sup>  
Even more text.

<sup>1</sup>A first.<sup>1</sup>A second. <sup>ii</sup>This is a very very long footnote that

Some text here and<sup>2</sup> even more there. Some text for this block to fill the page.

<sup>2</sup>Another first.

is continued here.

```
\usepackage[para]{manyfoot}
\DeclareNewFootnote[para]{B}[roman]
Some\footnote{A first.} text with two
footnotes.\footnoteB{A second.} More
text.\footnoteB{This is a very very long
footnote that\SplitNote} Even more text.

Some\FootnotetextB{}{is continued here.}
text here and\footnote{Another first.}
even more there. \sample % as elsewhere
```

3-2-21

If both parts of the footnote fall onto the same page after reformatting the document, the footnote parts get correctly reassembled, as we prove in the next example, which uses the same example text but a different measure. However, if the reformatting requires breaking the footnote in a different place, then further manual intervention is unavoidable. Thus, such work is best left until the last stage of production.

Some<sup>i</sup> text with two footnotes.<sup>j</sup> More text.<sup>ii</sup> Even more text.

Some text here and<sup>2</sup> even more there. Some text for this block to fill the page.

<sup>i</sup>A first.

<sup>j</sup>Another first.

<sup>ii</sup>A second. <sup>ii</sup>"This is a very very long footnote that is continued here.

```
\usepackage[para]{manyfoot}
\DeclareNewFootnote[para]{B}[roman]
Some\footnote{A first.} text with two
footnotes.\footnoteB{A second.} More
text.\footnoteB{This is a very very long
footnote that\SplitNote} Even more text.
```

```
Some\FootnotetextB{}{is continued here.}
text here and\footnote{Another first.}
even more there. \sample % as elsewhere
```

3-2-22

The vertical separation between a footnote block and the previous one is specified by `\skip\footins<suffix>`. By default, it is equal to `\skip\footins` (i.e., the separation between main text and footnotes). Initially the extra blocks are only separated by such spaces, but if the option `ruled` is included a `\footnoterule` is used as well. In fact, arbitrary material can be placed in that position by redefining the command `\extrafootnoterule`—the only requirement being that the typeset result from that command does not take up any additional vertical space (see the discussion of `\footnoterule` on page 112 for further details). It is even possible to use different rules for different blocks of footnotes; consult the package documentation for details.

Some text<sup>i,\*</sup> with a footnote. Even more text.<sup>A</sup> Some text<sup>†</sup> with a footnote.<sup>B</sup> Some more text for the example.

<sup>i</sup> A first.

<sup>\*</sup> A second.

<sup>†</sup> A sample.

<sup>A</sup> A third.

<sup>B</sup> Another sample.

```
\usepackage[marginal,multiple]{footmisc}
\usepackage[ruled]{manyfoot}
\DeclareNewFootnote{B}[fnsymbol]
\DeclareNewFootnote{C}[Alph]
\setlength{\skip\footinsB}{5pt minus 1pt}
\setlength{\skip\footinsC}{5pt minus 1pt}
Some text\footnote{A first.}\footnoteB{A second.}
with a footnote. Even more text.\footnoteC{A third.}
Some text\footnoteB{A sample.} with a
footnote.\footnoteC{Another sample.} Some more
text for the example.
```

3-2-23

Number the  
footnotes per page

The previous example deployed two additional *enum-styles*, `Alph` and `fnsymbol`. However, as only a few footnote symbols are available in both styles, that choice is most likely not a good one, unless we ensure that these footnote streams are numbered on a per-page basis. The `perpage` option of `footmisc` will not help here, as it applies to only the top-level footnotes. We can achieve the

desired effect either by using `\MakePerPage` from the `perpage` package on the counters `footnoteB` and `footnoteC` (as done below), or by using the `perpage` option of `manyfoot` (which calls on the `perpage` package to do the job, which will number all new footnote levels defined on a per-page basis). Note that the top-level footnotes are still numbered sequentially the way the example was set up.

<p>Some text<sup>1</sup> with a footnote. Even more<sup>*,^A</sup> text. Some</p> <hr/> <p><sup>1</sup>A first.</p> <p><sup>*</sup>Second.</p> <p><sup>A</sup>Third.</p>	<p>text<sup>A</sup> with a footnote here.<sup>B</sup> Some more text. And<sup>2,*</sup> a</p> <hr/> <p><sup>2</sup>Again.</p> <p><sup>*</sup>A last.</p> <p><sup>A</sup>A sample.</p> <p><sup>B</sup>Another sample.</p>	<pre>\usepackage[multiple]{footmisc} \usepackage{manyfoot,perpage} \DeclareNewFootnote{B}[fnsymbol] \DeclareNewFootnote{C}[Alph] \MakePerPage{footnoteB}\MakePerPage{footnoteC} Some text\footnote{A first.} with a footnote. Even more\footnoteB{Second.}\footnoteC{Third.} text. Some\footnoteC{A sample.} with a footnote here.\footnoteC{Another sample.} Some more text. And\footnote{Again.}\footnoteB{A last.} a last note.</pre>
--	--	--

3-2-24

### 3.2.7 endnotes—An alternative to footnotes

Scholarly works usually group notes at the end of each chapter or at the end of the document. Such notes are called endnotes. Endnotes are not supported in standard L<sup>A</sup>T<sub>E</sub>X, but they can be created in several ways.

The package `endnotes` (by John Lavagnino) provides its own `\endnote` command, thus allowing footnotes and endnotes to coexist.

The document-level syntax is modeled after the footnote commands if you replace `foot` with `end`—for example, `\endnote` produces an endnote, `\endnotemark` produces just the mark, and `\endnotetext` produces just the text. The counter used to hold the current endnote number is called `endnote` and is stepped whenever `\endnote` or `\endnotemark` without an optional argument is used.

All endnotes are stored in an external file with the extension `.ent` and are made available when you issue the command `\theendnotes`.

<p>This is simple text.<sup>1</sup> This is simple text.<sup>2</sup> Some more text with a mark.<sup>1</sup></p> <p><b>Notes</b></p>	<pre>\usepackage{endnotes} This is simple text.\endnote{The first endnote.} This is simple text.\endnote{The second endnote.} Some more text with a mark.\endnotemark[1]</pre>
<p><sup>1</sup>The first endnote.</p> <p><sup>2</sup>The second endnote.</p>	<pre>\theendnotes % output endnotes here</pre>

3-2-25

This process is different from the way the table of contents is built; the endnotes are written directly to the file, so that you will see only those endnotes which are defined earlier in the document. The advantage of this approach is that you can have several calls to `\theendnotes`, for example, at the end of each chapter.

To additionally restart the numbering you have to set the `\endnote` counter to zero after calling `\theendnotes`.

The heading produced by `\theendnotes` can be controlled in several ways. The text can be changed by modifying `\notesname` (default is the string `Notes`). If that is not enough you can redefine `\enoteheading`, which is supposed to produce the sectioning command in front of the notes.

The layout for endnote numbers is controlled through `\theendnote`, which is the standard way L<sup>A</sup>T<sub>E</sub>X handles counter formatting. The format of the mark is produced from `\makeenmark` with `\theenmark`, holding the formatted number for the current mark.

This is simple text.<sup>a)</sup> This is simple text.<sup>b)</sup> Some more text with a mark.<sup>a)</sup>

## Chapter Notes

- <sup>a)</sup>The first endnote.
- <sup>b)</sup>The second endnote.

```
\usepackage{endnotes}
\renewcommand{\theendnote}{\alph{endnote}}
\renewcommand{\makeenmark}{\textsuperscript{\theenmark}}
\renewcommand{\notesname}{Chapter Notes}
This is simple text.\endnote{The first endnote.}
This is simple text.\endnote{The second endnote.}
Some more text with a mark.\endnotemark[1]
\theendnotes
```

3-2-26]

The font size for the list of endnotes is controlled through `\enotesize`, which defaults to `\footnotesize`. Also, by modifying `\enoteformat` you can change the display of the individual endnotes within their list. This command is supposed to set up the paragraph parameters for the endnotes and to typeset the note number stored in `\theenmark`. In the example we start with no indentation for the first paragraph and with the number placed into the margin.

This is simple text.<sup>1</sup> This is simple text.<sup>2</sup> Some more text with a mark.<sup>1</sup>

## Notes

1. The first endnote with a lot of text to produce two lines.  
And even a second paragraph.
2. The second endnote.

```
\usepackage{endnotes}
\renewcommand{\enoteformat}{\noindent\raggedright
\setlength{\parindent}{12pt}\makebox[0pt][r]{\theenmark.\,}}
\renewcommand{\enotesize}{\scriptsize}
This is simple text.\endnote{The first endnote with a lot
of text to produce two lines.\par And even a second
paragraph.}
This is simple text.\endnote{The second endnote.}
Some more text with a mark.\endnotemark[1]
\theendnotes
```

3-2-27]

### 3.2.8 Marginal notes

The standard L<sup>A</sup>T<sub>E</sub>X command `\marginpar` generates a marginal note. This command typesets the text given as its argument in the margin, with the first line being at the same height as the line in the main text where the `\marginpar` command occurs. When only the mandatory argument is specified, the text goes to the right margin for one-sided printing; to the outside margin for two-sided printing;

and to the nearest margin for two-column formatting. When you also specify an optional argument, its text is used if the left margin is chosen, while the second (mandatory) argument is used for the right margin.

This placement strategy can be reversed (except for two-column formatting) using `\reversemarginpar`, which acts on all marginal notes from there on. You can return to the default behavior with `\normalmarginpar`.

There are a few important things to understand when using marginal notes. First, the `\marginpar` command does not start a paragraph. Thus, if it is used before the first word of a paragraph, the vertical alignment will not match the beginning of the paragraph. Second, the first word of its argument is not automatically hyphenated. Thus, for a narrow margin and long words (as in German), you may have to precede the first word by a `\hspace{0pt}` command to allow hyphenation of that word. These two potential problems can be eased by defining a command like `\marginlabel`, which starts with an empty box `\mbox{}`, typesets a marginal note ragged right, and adds a `\hspace{0pt}` in front of the argument.

<p>Some text with a marginal note. Some more text. Another text with a marginal note. Some more text. A lot of additional text here to fill up the space in the example on the left.</p>	<pre> ASuperLongFirstWord with problems \newcommand{\marginlabel}[1]{\mbox{}\marginpar ASuperLong-   {\raggedright\hspace{0pt}#1}} Firstword     Some\marginpar{ASuperLongFirstWord with problems} without       text with a marginal note. Some more text. problems      Another\marginlabel{ASuperLongFirstword without                   problems} text with a marginal note. Some more                   text. A lot of additional text here to fill                   up the space in the example on the left. </pre>
--	--

Of course, the above definition can no longer produce different texts depending on the chosen margin. With a little more finesse this problem could be solved, using, for example, the `\ifthenelse` constructs from the `ifthen` package.

The `LATEX` kernel tries hard (without producing too much processing overhead) to ensure that the contents of `\marginpar` commands always show up in the correct margin and in most circumstances will make the right decisions. In some cases, however, it will fail. If you are unlucky enough to stumble across one of them, a one-off solution is to add an explicit `\pagebreak` to stop the page generation from looking too far ahead. Of course, this has the disadvantage that the correction means visual formatting and has to be undone if the document changes. A better solution is to load the package `mparhack` written by Tom Sgouros and Stefan Ulrich. Once this package is loaded all `\marginpar` positions are tracked (internally using a label mechanism and writing the information to the `.aux` file). You may then get a warning “Marginpars may have changed. Rerun to get them right”, indicating that the positions have changed in comparison to the previous `LATEX` run and that a further run is necessary to stabilize the document.

As explained in Table 4.2 on page 196, there are three length parameters to customize the style of marginal notes: `\marginparwidth`, `\marginparsep`, and `\marginparpush`.

*Incorrectly placed  
\\marginpars*

	<i>Command</i>	<i>Default Definition</i>	<i>Representation</i>
<i>First Level</i>	<code>\labelitemi</code>	<code>\textbullet</code>	•
<i>Second Level</i>	<code>\labelitemii</code>	<code>\normalfont\bfseries \textendash</code>	–
<i>Third Level</i>	<code>\labelitemiii</code>	<code>\textasteriskcentered</code>	*
<i>Fourth Level</i>	<code>\labelitemiv</code>	<code>\textperiodcentered</code>	.

Table 3.5: Commands controlling an `itemize` list environment

### 3.3 List structures

Lists are very important  $\text{\LaTeX}$  constructs and are used to build many of  $\text{\LaTeX}$ 's display-like environments.  $\text{\LaTeX}$ 's three standard list environments are discussed in Section 3.3.1, where we also show how they can be customized. Section 3.3.2 starting on page 132 provides an in-depth discussion of the `paralist` package, which introduces a number of new list structures and offers comprehensive methods to customize them, as well as the standard lists. It is followed by a discussion of “headed lists”, such as theorems and exercises. Finally, Section 3.3.4 on page 144 discusses  $\text{\LaTeX}$ 's general list environment.

#### 3.3.1 Modifying the standard lists

It is relatively easy to customize the three standard  $\text{\LaTeX}$  list environments `itemize`, `enumerate`, and `description`, and the next three sections will look at each of these environments in turn. Changes to the default definitions of these environments can either be made globally by redefining certain list-defining parameters in the document preamble or can be kept local.

##### Customizing the `itemize` list environment

For a simple unnumbered `itemize` list, the labels are defined by the commands shown in Table 3.5. To create a list with different-looking labels, you can redefine the label-generating command(s). You can make that change local for one list, as in the example below, or you can make it global by putting the redefinition in the document preamble. The following simple list is a standard `itemize` list with a marker from the PostScript Zapf Dingbats font (see Section 7.6.4 on page 378) for the first-level label:

```
\usepackage{pifont}
\newenvironment{MYitemize}{\renewcommand\labelitemi
  {\ding{43}}\begin{itemize}}{\end{itemize}}
\begin{MYitemize}
\item Text of the first item in the list.
\item Text of the first sentence in the second item of the list. And the second sentence.
\end{MYitemize}
```

Text of the first item in the list.

Text of the first sentence in the second item of the list. And the second sentence.

### Customizing the enumerate list environment

$\text{\LaTeX}$ 's enumerated (numbered) list environment `enumerate` is characterized by the commands and representation forms shown in Table 3.6 on the next page. The first row shows the names of the counter used for numbering the four possible levels of the list. The second and third rows are the commands giving the representation of the counters and their default definition in the standard  $\text{\LaTeX}$  class files. Rows four, five, and six contain the commands, the default definition, and an example of the actual enumeration string printed by the list.

A reference to a numbered list element is constructed using the `\theenumi`, `\theenumii`, and similar commands, prefixed by the commands `\p@enumi`, `\p@enumii`, etc., respectively. The last three rows in Table 3.6 on the following page show these commands, their default definition, and an example of the representation of such references. It is important to consider the definitions of both the representation and reference-building commands to get the references correct.

We can now create several kinds of numbered description lists simply by applying what we have just learned.

Our first example redefines the first- and second-level counters to use capital Roman digits and Latin characters. The visual representation should be the value of the counter followed by a dot, so we can use the default value from Table 3.6 on the next page for `\labelenumi`.

#### I. Introduction

##### A. Applications

Motivation for research and applications related to the subject.

##### B. Organization

Explain organization of the report, what is included, and what is not.

#### II. Literature Survey

3-3-2 q1=I q2=IA q3=IB q4=II

```
\renewcommand\theenumi {\Roman{enumi}}
\renewcommand\theenumii {\Alph{enumi}}
\renewcommand\labelenumii{\theenumii.}
\begin{enumerate}
\item \textbf{Introduction} \label{q1}
\begin{enumerate}
\item \textbf{Applications} \\
Motivation for research and applications
related to the subject. \label{q2}
\item \textbf{Organization} \\
Explain organization of the report, what
is included, and what is not. \label{q3}
\end{enumerate}
\item \textbf{Literature Survey} \label{q4}
\end{enumerate}
\end{document}
```

After these redefinitions we get funny-looking references; to correct this we have to adjust the definition of the prefix command `\p@enumii`. For example, to get a reference like “I-A” instead of “IA” as in the previous example, we need

```
\makeatletter \renewcommand\p@enumii{\theenumi--} \makeatother
```

because the reference is typeset by executing `\p@enumii` followed by `\theenumii`.

	<i>First Level</i>	<i>Second Level</i>	<i>Third Level</i>	<i>Fourth Level</i>
<i>Counter Representation</i>	<code>enumi</code>	<code>enumii</code>	<code>enumiii</code>	<code>enumiv</code>
<i>Default Definition</i>	<code>\arabic{enumi}</code>	<code>\alph{enumii}</code>	<code>\roman{enumiii}</code>	<code>\Alph{enumiv}</code>
<i>Label Field</i>	<code>\labelenumi</code>	<code>\labelenumii</code>	<code>\labelenumiii</code>	<code>\labelenumiv</code>
<i>Default Form</i>	<code>\theenumi.</code>	<code>(\theenumii)</code>	<code>\theenumiii.</code>	<code>\theenumiv.</code>
<i>Numbering Example</i>	1., 2.	(a), (b)	i., ii.	A., B.
<i>Reference representation</i>				
<i>Prefix</i>	<code>\p@enumi</code>	<code>\p@enumii</code>	<code>\p@enumiii</code>	<code>\p@enumiv</code>
<i>Default Definition</i>	<code>{}</code>	<code>\theenumi</code>	<code>\theenumi(\theenumii)</code>	<code>\p@enumii\theenumiii</code>
<i>Reference Example</i>	1, 2	1a, 2b	1(a)i, 2(b)ii	1(a)iA, 2(b)iiB

Table 3.6: Commands controlling an `enumerate` list environment

Note that we need `\makeatletter` and `\makeatother` because the command name to redefine contains an @ sign. Instead of this low-level method, consider using `\labelformat` from the `varioref` package described in Section 2.4.2.

You can also decorate an `enumerate` field by adding something to the label field. In the example below, we have chosen for the first-level list elements the paragraph sign (§) as a prefix and a period as a suffix (omitted in references).

§1. text inside list, more text inside list	<code>\renewcommand{\labelenumi}{\\$\\theenumi.}</code>
§2. text inside list, more text inside list	<code>\usepackage{varioref} \labelformat{enumi}{\\$#1}</code> <code>\begin{enumerate}</code>
§3. text inside list, more text inside list	<code>\item \label{w1} text inside list, more text inside list</code> <code>\item \label{w2} text inside list, more text inside list</code> <code>\item \label{w3} text inside list, more text inside list</code> <code>\end{enumerate}</code>
w1=§1 w2=§2 w3=§3	<code>w1=\ref{w1} w2=\ref{w2} w3=\ref{w3}</code>

3-3-3

You might even want to select different markers for consecutive labels. For instance, in the following example, characters from the PostScript font ZapfDingbats are used. In this case there is no straightforward way to automatically make the `\ref` commands produce the correct references. Instead of `\theenumi` simply producing the representation of the `enumi` counter, we define it to calculate from the counter value which symbol to select. The difficulty here is to create this definition in a way such that it survives the label-generating process. The trick is to add the `\protect` commands so that `\setcounter` and `\ding` are not executed when the label is written to the `.aux` file, yet to ensure that the current value of the counter is stored therein. The latter goal is achieved by prefixing `\value` by the (internal)

\TeX command `\the` within `\setcounter` (but not within `\ding!`); without it the references would all show the same values.<sup>1</sup>

- ```
\usepackage{calc,pifont} \newcounter{local}
\renewcommand{\theenumi}{\protect\setcounter{local}{%
  \arabic{enumi}}\protect\ding{\value{local}}}
\renewcommand{\labelenumi}{\theenumi}
\begin{enumerate}
\item text inside list, text inside list, text
  inside list, more text inside list;
\item text inside list, text inside list, text
  inside list, more text inside list;
\item text inside list, text inside list, text
  inside list, more text inside list.
\end{enumerate}
```

3-3-4] l1=① l2=② l3=③

The same effect is obtained with the `dingautolist` environment defined in the `pifont` package, which is part of the `PSNFSS` system (see Section 7.6.4 on page 378).

#### Customizing the `description` list environment

With the `description` environment you can change the `\descriptionlabel` command that generates the label. In the following example the font for typesetting the labels is changed from boldface (default) to sans serif.

- ```
\renewcommand{\descriptionlabel}[1]{%
  \hspace{\labelsep}\textsf{\#1}}
\begin{description}
\item[A.] text inside list, text inside list,
  text inside list, more text inside list;
\item[B.] text inside list, text inside list,
  text inside list, more text inside list;
\end{description}
```

The standard \TeX class files set the starting point of the label box in a `description` environment at a distance of `\labelsep` to the left of the left margin of the enclosing environment. Thus, the `\descriptionlabel` command in the example above first adds a value of `\labelsep` to start the label aligned with the left margin (see page 147 for detailed explanations).

<sup>1</sup>For the \TeXnically interested: \TeX's `\value` command, despite its name, does not produce the “value” of a \TeX counter but only its internal \TeX register name. In most circumstances this can be used as the value but unfortunately not inside `\edef` or `\write`, where the internal name rather than the “value” will survive. By prefixing the internal register name with the command `\the`, we get the “value” even in such situations.

### 3.3.2 paralist—Extended list environments

The `paralist` package created by Bernd Schandl provides a number of new list environments and offers extensions to L<sup>A</sup>T<sub>E</sub>X's standard ones that make their customization much easier. Standard and new list environments can be nested within each other and the enumeration environments support the `\label/\ref` mechanism.

#### Enumerations

All standard L<sup>A</sup>T<sub>E</sub>X lists are display lists; that is, they leave some space at their top and bottom as well as between each item. Sometimes, however, one wishes to enumerate something within a paragraph without such visual interruption. The `inparaenum` environment was developed for this purpose. It supports an optional argument that you can use to customize the generated labels, the exact syntax of which is discussed later in this section.

```
\usepackage{paralist}
We may want to enumerate items within a paragraph to
\begin{inparaenum}[(a)]
  \item save space
  \item make a less prominent statement, or
  \item for some other reason.
\end{inparaenum}
```

We may want to enumerate items within a paragraph to (a) save space (b) make a less prominent statement, or (c) for some other reason.

3-3-6 |

But perhaps this is not precisely what you are looking for. A lot of people like to have display lists but prefer them without much white space surrounding them. In that case `compactenum` might be your choice, as it typesets the list like `enumerate` but with all vertical spaces set to 0pt.

```
\usepackage{paralist}
On the other hand we may want to enumerate like this:
\begin{compactenum}[i)]
  \item still make a display list
  \item format items as usual but with less
        vertical space, that is
  \item similar to normal \texttt{enumerate}.
\end{compactenum}
```

On the other hand we may want to enumerate like this:

- i) still make a display list
- ii) format items as usual but with less vertical space, that is
- iii) similar to normal `enumerate`.

3-3-7 |

Actually, our previous statement was not true—you can customize the vertical spaces used by `compactenum`. Here are the parameters: `\pltopsep` is the space above and below the environment, `\plpartopsep` is the extra space added to the previous space when the environment starts a paragraph on its own, `\plitemsep` is the space between items, and `\plparsep` is the space between paragraphs within an item.

A final enumeration alternative is offered with the `asparaenum` environment, which formats the items as individual paragraphs. That is, their first line is indented by `\parindent` and following lines are aligned with the left margin.

Or perhaps we may want to enumerate like this:

- 1) still make a display list
- 2) format items as paragraphs with turnover lines not indented, that is
- 3) similar to normal `enumerate`.

3-3-8

```
\usepackage{paralist}
Or perhaps we may want to enumerate like this:
\begin{asparaenum}[1)]
\item still make a display list \item format items
as paragraphs with turnover lines not indented,
that is \item similar to normal \texttt{\{enumerate\}}.
\end{asparaenum}
```

As seen in the previous examples all enumeration environments support one optional argument that describes how to format the item labels. Within the argument the tokens A, a, I, i, and 1 have a special meaning: they are replaced by the enumeration counter displayed in style `\Alph`, `\alph`, `\Roman`, `\roman`, or `\arabic`, respectively. All other characters retain their normal meanings. Thus, the argument `[(a)]` will result in labels like (a), (b), (c), and so on, while `[\S i:]` will produce §i, §ii, §iii, and so on.

You have to be a bit careful if your label contains text strings, such as labels like Example 1, Example 2, ... In this case you have to hide the “a” inside a brace group—that is, use an argument like `[{Example} 1]`. Otherwise, you will get strange results, as shown in the next example.

Item b shows what can go wrong:  
 Example a: On the first item we will not notice it but  
 Example b: the second item then shows what happens if a special character is mistakenly matched.

3-3-9

```
\usepackage{paralist}
Item~\ref{bad} shows what can go wrong:
\begin{asparaenum}[Example a:]
\item On the first item we will not notice it
but \item the second item then shows what
happens if a special character is mistakenly
matched. \label{bad}
\end{asparaenum}
```

Fortunately, the package usually detects such incorrect input and will issue a warning message. A consequence of hiding special characters by surrounding them with braces is that an argument like `[\textbf{a}]` will not work either, because the “a” will not be considered as special any more. A workaround for this case is to use something that does not require braces, such as `\bfseries`.

As can be seen above, referencing a `\label` will produce only the counter value in the chosen representation but not any frills added in the optional argument. This is the case for all enumeration environments.

It is not possible with this syntax to specify that a label should show the outer as well as the inner enumeration counter, because the special characters always refer to the current enumeration counter. There is one exception: if you load the

package with the option `pointedenum` or with the option `pointlessenum`, you will get labels like those shown in the next example.

```
\usepackage[pointedenum]{paralist}
\begin{compactenum}
\item First level.
    \begin{compactenum}
        \item Second level.
            \begin{compactenum}
                \item Third level.
                    \begin{compactenum}
                        \item Second level again.
                    \end{compactenum}
                \end{compactenum}
            \end{compactenum}
        \item Second level again.
    \end{compactenum}

```

3-3-10

The difference between the two options is the presence or absence of the trailing period. As an alternative to the options you can use the commands `\pointedenum` and `\pointlessenum`. They enable you to define your own environments that format labels in this way while other list environments show labels in different formats. If you need more complicated labels, such as those involving several enumeration counters from different levels, then you have to construct them manually using the methods described in Section 3.3.1 on page 129.

The optional argument syntax for specifying the typesetting of enumeration labels was first implemented in the `enumerate` package by David Carlisle, who extended the standard `enumerate` environment to support such an optional argument. With `paralist` the optional argument is supported for all enumeration environments, including the standard `enumerate` environment (for which it is an upward-compatible extension).

If an optional argument is used on any of the enumeration environments, then by default the left margin will be made only as wide as necessary to hold the labels. More exactly, the indentation is adjusted to the width of the label as it would be if the counter value is currently seven. This produces a fairly wide number (vii) if the numbering style is “Roman” and does not matter otherwise. This behavior is shown in the next example. For some documents this might be the right behavior, but if you prefer a more uniform indentation use the option `neverdecrease`, which will ensure that the left margin is always at least as wide as the default setting.

The left margin may vary if we are not careful.

1. An item in a normal `enumerate`.
1. Same left margin in
  2. this case.
  - i) But a different one
  - ii) here.

```
\usepackage{paralist}
The left margin may vary if we are not careful.
\begin{enumerate}
\item An item in a normal \texttt{enumerate}.
\end{enumerate}
\begin{compactenum}
\item Same left margin in \item this case.
\end{compactenum}
\begin{compactenum}[i]
\item But a different one \item here.
\end{compactenum}
```

3-3-11

On the other hand, you can always force that kind of adjustment, even for environments without an optional argument, by specifying the option `alwaysadjust`.

<p>Here we force the shortest possible indentation always:</p> <ol style="list-style-type: none"> <li>1. An item in a normal <code>enumerate</code>.</li> <li>i) But a different indentation</li> <li>ii) here.</li> <li>1. Same left margin as</li> <li>2. in the first case.</li> </ol>	<pre>\usepackage[alwaysadjust]{paralist} Here we force the shortest possible indentation always: \begin{enumerate} \item An item in a normal \texttt{enumerate}. \end{enumerate} \begin{compactenum}[i] \item But a different \item indentation \item here. \end{compactenum} \begin{compactenum}[1.] \item Same left margin as \item in the first case. \end{compactenum}</pre>
---	--

Finally, with the option `neveradjust` the standard indentation is used in all cases. Thus, labels that are too wide will extend into the left margin.

<p>With this option the label is pushed into the margin.</p> <ol style="list-style-type: none"> <li>1. An item in a normal <code>enumerate</code>.</li> </ol> <p>Task A) Same left indentation in Task B) this case.</p> <ol style="list-style-type: none"> <li>1) And the same indentation</li> <li>2) here.</li> </ol>	<pre>\usepackage[neveradjust]{paralist} With this option the label is pushed into the margin. \begin{enumerate} \item An item in a normal\\ \texttt{enumerate}. \end{enumerate} \begin{compactenum}[{Task} A)] \item Same left indentation in \item this case. \end{compactenum} \begin{compactenum}[1] \item And the same indentation \item here. \end{compactenum}</pre>
--	--

### Itemizations

For itemized lists the `paralist` package offers the environments `compactitem`, which is a compact version of the standard `itemize` environment; `asparaitem` which formats the items as paragraphs; and `inparaitem`, which produces an inline itemization. The last environment was added mainly for symmetry reasons. All three environments accept an optional argument, that specifies the label to be used for each item.

<p>Producing itemized lists with special labels is easy.</p> <ul style="list-style-type: none"> <li>★ This example uses the package option <code>neverdecrease</code>.</li> <li>★ Without it the left margin would be smaller.</li> </ul>
---

<pre>\usepackage[neverdecrease]{paralist} Producing itemized lists with special labels is easy. \begin{compactitem}[\$\star\$] \item This example uses the package option       \texttt{neverdecrease}. \item Without it the left margin would be smaller. \end{compactitem}</pre>
--

3-3-14

The three label justification options `neverdecrease`, `alwaysadjust`, and `neveradjust` are also valid for the itemized lists, as can be seen in the previous example. When the `paralist` package is loaded, L<sup>A</sup>T<sub>E</sub>X's `itemize` environment is extended to also support that type of optional argument.

### Descriptions

For descriptions the `paralist` package introduces three additional environments: `compactdesc`, which is like the standard L<sup>A</sup>T<sub>E</sub>X description environment but with all vertical spaces reduced to zero (or whatever you specify as a customization); `asparadesc`, which formats each item as a paragraph; and `inparadesc`, which allows description lists within running text.

Because description-type environments specify each label at the `\item` command, these environments have no need for an optional argument.

Do you like inline description lists?  
Try them out!

**paralist** A useful package as it supports `compact...` environments that have zero vertical space, `aspara...` environments formatted as paragraphs, and `inpara...` environments as inline lists.

**enumerate** A package that is superseded now.

```
\usepackage{paralist}
Do you like inline description lists? Try them out!
\begin{compactdesc}
\item[paralist] A useful package as it supports
  \begin{inparadesc} \item[compact\ldots] environments
    that have zero vertical space, \item[aspara\ldots]
    environments formatted as paragraphs, and
    \item[inpara\ldots] environments as inline lists.
  \end{inparadesc}
\item[enumerate] A package that is superseded now.
\end{compactdesc}
```

3-3-15

### Adjusting defaults

Besides providing these useful new environments the `paralist` package lets you customize the default settings of enumerated and itemized lists.

You can specify the default labels for different levels of itemized lists with the help of the `\setdefaultitem` declaration. It takes four arguments (as four levels of nesting are possible). In each argument you specify the desired label (just as you do with the optional argument on the environment itself) or, if you are satisfied with the default for the given level, you specify an empty argument.

- Outer level is using the default label.
  - On the second level we use again a bullet.
    - \* And on the third level a star.

```
\usepackage{paralist} \setdefaultitem{}{\textbullet}{*\star}{}
\begin{compactitem}
\item Outer level is using the default label.
  \begin{compactitem}
  \item On the second level we use again a bullet.
    \begin{compactitem}
    \item And on the third level a star.
    \end{compactitem}
  \end{compactitem}
\end{compactitem}
```

3-3-16

The changed defaults apply to all subsequent itemized environments. Normally, such a declaration is placed into the preamble, but you can also use it to change the defaults mid-document. In particular, you can define environments that contain a `\setdefaultitem` declaration which would then apply only to that particular environment—and to lists nested within its body.

You will probably not be surprised to learn that a similar declaration exists for enumerations. By using `\setdefaultenum` you can control the default look and feel of such environments. Again, there are four arguments corresponding to the four levels. In each you either specify your label definition (using the syntax explained earlier) or you leave it empty to indicate that the default for this level should be used.

- 3-3-17
- 1) All levels get a closing parenthesis in this example.
    - a) Lowercase letters here.
      - i) Roman numerals here.
      - ii) Really!
- ```
\usepackage{paralist} \setdefaultenum{1}{a}{i}{A}
\begin{compactenum}
\item All levels get a closing parenthesis in this example.
  \begin{compactenum}
    \item Lowercase letters here.
      \begin{compactenum}
        \item Roman numerals here. \item Really!
      \end{compactenum}
    \end{compactenum}
  \end{compactenum}
```

There is also the possibility of adjusting the indentation for the various list levels using the declaration `\setdefaultleftmargin`. However, this command has six arguments (there are a total of six list levels in the standard L<sup>A</sup>T<sub>E</sub>X classes), each of which takes either a dimension denoting the increase of the indentation at that level or an empty argument indicating to use the current value as specified by the class or elsewhere. Another difference from the previous declarations is that in this case we are talking about the absolute list levels and not about relative levels related to either enumerations or itemizations (which can be mixed). Compare the next example with the previous one to see the difference.

- 3-3-18
- 1) All levels get a closing parenthesis in this example.
    - a) Lowercase letters here.
      - i) Roman numerals here.
      - ii) Really!
- ```
\usepackage{paralist}
\setdefaultenum{1}{a}{i}{A}
\setdefaultleftmargin{\parindent}{\parindent}{\parindent}{\parindent}{\parindent}{\parindent}
\begin{compactenum}
\item All levels get a closing parenthesis in this example.
  \begin{compactenum}
    \item Lowercase letters here.
      \begin{compactenum}
        \item Roman numerals here. \item Really!
      \end{compactenum>
    \end{compactenum}
  \end{compactenum}
```

By default, enumeration and itemized lists set their labels flush right. This behavior can be changed with the help of the option `flushleft`.

As described earlier, the label of the standard `description` list can be adjusted by modifying `\descriptionlabel`, which is also responsible for formatting the label in a `compactdesc` environment. With `inparadesc` and `asparadesc`, however, a different command, `\paradescriptionlabel`, is used for this purpose. As these environments handle their labels in slightly different ways, they do not need adjustments involving `\labelsep` (see page 147). Thus, its default definition is simply:

```
\newcommand*\paradescriptionlabel[1]{\normalfont\bfseries #1}
```

Finally, the `paralist` package supports the use of a configuration file named `paralist.cfg`, which by default is loaded if it exists. You can prevent this by specifying the option `nocfg`.

### 3.3.3 amsthm—Providing headed lists

The term “headed lists” describes typographic structures that, like other lists such as quotations, form a discrete part of a section or chapter and whose start and finish, at least, must be clearly distinguished. This is typically done by adjusting the vertical space at the start or adding a rule, and in this case also by including some kind of heading, similar to a sectioning head. The end may also be distinguished by a rule or other symbol, maybe within the last paragraph, and by extra vertical space.

Another property that distinguishes such lists is that they are often numbered, using either an independent system or in conjunction with the sectional numbering.

Perhaps one of the more fruitful sources of such “headed lists” is found in the so-called “theorem-like” environments. These had their origins in mathematical papers and books but are equally applicable to a wide range of expository material, as examples and exercises may take this form whether or not they contain mathematical material.

Because their historical origins lie in the mathematical world, we choose to describe the `amsthm` package [7] by Michael Downes from the American Mathematical Society (AMS) as a representative of this kind of extension.<sup>1</sup> This package provides an enhanced version of standard L<sup>A</sup>T<sub>E</sub>X’s `\newtheorem` declaration for specifying theorem-like environments (headed lists).

As in standard L<sup>A</sup>T<sub>E</sub>X, environments declared in this way take an optional argument in which extra text, known as “notes”, can be added to the head of the environment. See the example below for an illustration.

<sup>1</sup>When the `amsthm` package is used with a non-AMS document class and with the `amsmath` package, `amsthm` must be loaded *after* `amsmath`. The AMS document classes incorporate both packages.

```
\newtheorem*{name}{heading}
```

The `\newtheorem` declaration has two mandatory arguments. The first is the environment name that the author would like to use for this element. The second is the heading text.

If `\newtheorem*` is used instead of `\newtheorem`, no automatic numbers will be generated for the environments. This form of the command can be useful if you have only one lemma or exercise and do not want it to be numbered; it is also used to produce a special named variant of one of the common theorem types.

```
\usepackage{amsthm}
\newtheorem{lem}{Lemma}
\newtheorem*[ML]{Mittelbach's Lemma}
\begin{lem}[Main] The \LaTeX{} Companion complements any \LaTeX{} introduction.
\end{lem}
\begin{ML} The \LaTeX{} Companion contains packages from all application areas.
\end{ML}
```

**Lemma 1 (Main).** *The  $\text{\LaTeX}$  Companion complements any  $\text{\LaTeX}$  introduction.*

**Mittelbach's Lemma.** *The  $\text{\LaTeX}$  Companion contains packages from all application areas.*

3-3-19

In addition to the two mandatory arguments, `\newtheorem` has two mutually exclusive optional arguments. They affect the sequencing and hierarchy of the numbering.

```
\newtheorem{name}[use-counter]{heading}
\newtheorem{name}{heading}[number-within]
```

By default, each kind of theorem-like environment is numbered independently. Thus, if you have lemmas, theorems, and some examples interspersed, they will be numbered something like this: Example 1, Lemma 1, Lemma 2, Theorem 1, Example 2, Lemma 3, Theorem 2. If, for example, you want the lemmas and theorems to share the same numbering sequence—Example 1, Lemma 1, Lemma 2, Theorem 3, Example 2, Lemma 4, Theorem 5—then you should indicate the desired relationship as follows:

```
\newtheorem{thm}{Theorem}      \newtheorem{lem}[thm]{Lemma}
```

The optional *use-counter* argument (value `thm`) in the second statement means that the `lem` environment should share the `thm` numbering sequence instead of having its own independent sequence.

To have a theorem environment numbered subordinately within a sectional unit—for example, to get exercises numbered Exercise 2.1, Exercise 2.2, and so on, in Section 2—put the name of the parent counter in square brackets in the final position:

```
\newtheorem{exa}{Exercise}[section]
```

With the optional argument `[section]`, the `exa` counter will be reset to 0 whenever the parent counter `section` is incremented.

### Defining the style of headed lists

The specification part of the `amsthm` package supports the notion of a current theorem style, which determines the formatting that will be set up by a collection of `\newtheorem` commands.<sup>1</sup>

```
\theoremstyle{style}
```

The three theorem styles provided by the package are `plain`, `definition`, and `remark`; they specify different typographical treatments that give the environments a visual emphasis corresponding to their relative importance. The details of this typographical treatment may vary depending on the document class, but typically the `plain` style produces italic body text and the other two styles produce Roman body text.

To create new theorem-like environments in these styles, divide your `\newtheorem` declarations into groups and preface each group with the appropriate `\theoremstyle`. If no `\theoremstyle` command is given, the style used will be `plain`. Some examples follow:

**Definition 1.** A typographical challenge is a problem that cannot be solved with the help of *The L<sup>A</sup>T<sub>E</sub>X Companion*.

**Theorem 2.** *There are no typographical challenges.*

**Remark.** The proof is left to the reader.

```
\usepackage{amsthm}
\theoremstyle{plain} \newtheorem{thm}{Theorem}
\theoremstyle{definition} \newtheorem{defn}[thm]{Definition}
\theoremstyle{remark} \newtheorem*{rem}{Remark}
\begin{defn}
A typographical challenge is a problem that cannot be solved with the help of \emph{The LATEX Companion}.
\end{defn}
\begin{thm}There are no typographical challenges.\end{thm}
\begin{rem}The proof is left to the reader.\end{rem}
```

3-3-20

Note that the fairly obvious choice of “def” for the name of a “Definition” environment does not work, because it conflicts with the existing low-level T<sub>E</sub>X command `\def`.

A fairly common style variation for theorem heads is to have the theorem number on the left, at the beginning of the heading, instead of on the right. As this variation is usually applied across the board regardless of individual `\theoremstyle` changes, swapping numbers is done by placing a `\swapnumbers` declaration at the beginning of the list of `\newtheorem` statements that should be affected.

*Number swapping*

<sup>1</sup>This was first introduced in the now-superseded `theorem` package by Frank Mittelbach.

### Advanced customization

More extensive customization capabilities are provided by the package through the `\newtheoremstyle` declaration and through a mechanism for using package options to load custom theorem style definitions.

```
\newtheoremstyle{name}{space-above}{space-below}{body-style}{indent}{head-style}{head-after-punct}{head-after-space}{head-full-spec}
```

To set up a new style of “theorem-like” headed list, use this declaration with the nine mandatory arguments described below. For many of these arguments, if they are left empty, a default is used as listed here.

*name* The name used to refer to the new style.

*space-above* The vertical space above the headed list, a rubber length (default `\topsep`).

*space-below* The vertical space below the headed list, a rubber length (default `\topsep`).

*body-style* A declaration of the font and other aspects of the style to use for the text in the body of the list (default `\normalfont`).

*indent* The extra indentation of the first line of the list, a non-rubber length (default is no extra indent).

*head-style* A declaration of the font and other aspects of the style to use for the text in the head of the list (default `\normalfont`).

*head-after-punct* The text (typically punctuation) to be inserted after the head text, including any note text.

*head-after-space* The horizontal space to be inserted after the head text and “punctuation”, a rubber length. It cannot be completely empty. As two very special cases it can contain either a single space character to indicate that just a normal interword space is required or, more surprisingly, just the command `\newline` to indicate that a new line should be started for the body of the list.

*head-full-spec* A non-empty value for this argument enables a complete specification of the setting of the head itself to be supplied; an empty value means that the layout of the “plain” theorem style is used. See below for further details.

Any extra set-up code for the whole environment is best put into the *body-style* argument, although care needs to be taken over how it will interact with what is set up automatically. Anything that applies only to the head can be put in *head-style*.

In the example below we define a *break* theorem style, which starts a new line after the heading. The heading text is set in bold sans serif, followed by a colon and outdented into the margin by 12pt. Since the book examples are typeset in a very small measure, we added \RaggedRight to the *body-style* argument.

```
\usepackage{ragged2e,amsthm}
\newtheoremstyle{break}%
  {9pt}{9pt}% Space above and below
  {\itshape\RaggedRight}% Body style
  {-12pt}% Heading indent amount
  {\sffamily\bfseries}{:}% Heading font and punctuation after it
  {\newline}% Space after heading (\newline = linebreak)
  {}% Head spec (empty = same as 'plain' style)

\theoremstyle{break}
```

**Exercise 1 (Active author):**

*Find the author responsible for the largest number of packages described in The L<sup>A</sup>T<sub>E</sub>X Companion.*

```
\newtheorem{exa}[Exercise]
```

```
\begin{exa}[Active author]
```

*Find the author responsible for the largest number of packages described in The L<sup>A</sup>T<sub>E</sub>X Companion.*

```
\end{exa}
```

[3-3-21]

The *head-full-spec* argument, if non-empty, becomes the definition part of an internal command that is used to typeset the (up to) three bits of information contained in the head of a theorem-like environment: its number (if any), its name, and any extra notes supplied by the author when using the environment. Thus, it should contain references to three arguments that will then be replaced as follows:

- #1 The fixed text that is to be used in the head (for example, "Exercises"), It comes from the \newtheorem used to declare an environment.
- #2 A representation of the number of the element, if it should be numbered. It is conventionally left empty if the environment should not be numbered.
- #3 The text for the optional note, from the environment's optional argument.

Assuming all three parts are present, the contents of the *head-full-spec* argument could look as follows:

```
#1 #2 \textup{(#3)}
```

Although you are free to make such a declaration, it is normally best not to use these arguments directly as this might lead to unwanted extra spaces if, for example, the environment is unnumbered.

To account for this extra complexity, the package offers three additional commands, each of which takes one argument: \thmname, \thmnumber, and \thmnote. These three commands are redefined at each use of the environment so as to process their arguments in the correct way. The default for each of them is simply to "typeset the argument". Nevertheless, if, for example, the particular occurrence is

unnumbered, then `\thmnumber` gets redefined to do no typesetting. Thus, a better definition for the `head-full-spec` argument would be

```
\thmname{\#1}\thmnumber{ #2}\thmnote{ \textup{(#3)}}
```

which corresponds to the set-up used by the default plain style. Note the spaces within the last two arguments: they provide the interword spaces needed to separate the parts of the typeset head but, because they are inside the arguments, they are present only if that part of the head is typeset.

In the following example we provide a “Theorem” variation in which the whole theorem heading has to be supplied as an optional note, such as for citing theorems from other sources.

```
\usepackage{amsthm}
\newtheoremstyle{citing}%
  {}% Name
  {3pt}{3pt}% Space above and below
  {\itshape}% Body font
  {\parindent}{\bfseries}% Heading indent and font
  {.}% Punctuation after heading
  { }% Space after head (" " = normal interword space)
  {\thmnote{#3}}% Typeset note only, if present
\theoremstyle{citing} \newtheorem*{varthm}{}%
\begin{varthm}[Theorem 3.16 in \cite{Knuth90}]
By focusing on small details, it is possible to understand the deeper significance of a passage.
\end{varthm}
```

**Theorem 3.16 in [87].** *By focusing on small details, it is possible to understand the deeper significance of a passage.*

3-3-22

### Proofs and the QED symbol

Of more specifically mathematical interest, the package defines a proof environment that automatically adds a “QED symbol” at the end. This environment produces the heading “Proof” with appropriate spacing and punctuation.<sup>1</sup>

An optional argument of the proof environment allows you to substitute a different name for the standard “Proof”. If you want the proof heading to be, for example, “Proof of the Main Theorem”, then put this in your document:

```
\begin{proof}[Proof of the Main Theorem]
...
\end{proof}
```

A “QED symbol” (default  $\square$ ) is automatically appended at the end of a proof environment. To substitute a different end-of-proof symbol, use `\renewcommand` to redefine the command `\qedsymbol`. For a long proof done as a subsection or

<sup>1</sup>The proof environment is primarily intended for short proofs, no more than a page or two in length. Longer proofs are usually better done as a separate `\section` or `\subsection` in your document.

section, you can obtain the symbol and the usual amount of preceding space by using the command `\qed` where you want the symbol to appear.

Automatic placement of the QED symbol can be problematic if the last part of a proof environment is, for example, tabular or a displayed equation or list. In that case put a `\qedhere` command at the somewhat earlier place where the QED symbol should appear; it will then be suppressed from appearing at the logical end of the proof environment. If `\qedhere` produces an error message in an equation, try using `\mbox{\qedhere}` instead.

```
\usepackage{amsthm}
\begin{proof}[Proof (sufficiency)]
This proof involves a list:
\begin{enumerate}
\item because the proof comes in two parts --- we need to use \verb|\qedhere|. \qedhere
\end{enumerate}
\end{proof}
```

*Proof (sufficiency).* This proof involves a list:

1. because the proof comes in two parts ---
2. — we need to use `\qedhere`.  $\square$

3-3-23

### 3.3.4 Making your own lists

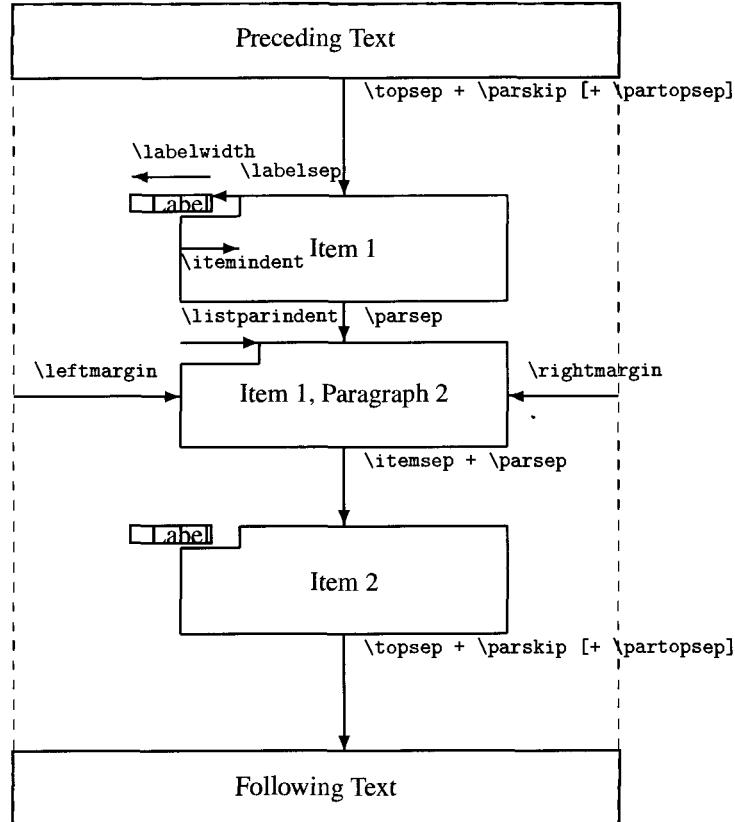
Most lists in L<sup>A</sup>T<sub>E</sub>X, including those we have seen previously, are internally built using the generic list environment. It has the following syntax:

```
\begin{list}{default-label}{{decls}} item-list \end{list}
```

The argument *default-label* is the text to be used as a label when an `\item` command is found without an optional argument. The second argument, *decls*, can be used to modify the different geometrical parameters of the *list* environment, which are shown schematically in Figure 3.3 on the next page.

The default values of these parameters typically depend on the type size and the level of the list. Those being vertically oriented are rubber lengths, meaning that they can stretch or shrink. They are set by the *list* environment as follows: upon entering the environment the internal command `\@list{level}` is executed, where *level* is the list nesting level represented as a Roman numeral (e.g., `\@listi` for the first level, `\@listii` for the second, `\@listiii` for the third, and so on). Each of these commands, defined by the document class, holds appropriate settings for the given level. Typically, the class contains separate definitions for each major document size available via options. For example, if you select the option `11pt`, one of its actions is to change the list defaults. In the standard classes this is done by loading the file `size11.clo`, which contains the definitions for the `11pt` document size.

In addition, most classes contain redefinitions of `\@listi` (i.e., first-level list defaults) within the size-changing commands `\normalsize`, `\small`, and `\footnotesize`, the assumption being that one might have lists within “small”



`\topsep` rubber space between first item and preceding paragraph.

`\partopsep` extra rubber space added to `\topsep` when environment starts a new paragraph.

`\itemsep` rubber space between successive items.

`\parsep` rubber space between paragraphs within an item.

`\leftmargin` space between left margin of enclosing environment (or of page if top-level list) and left margin of this list. Must be non-negative. Its value depends on the list level.

`\rightmargin` similar to `\leftmargin` but for the right margin. Its value is usually 0pt.

`\listparindent` extra indentation at beginning of every paragraph of a list except the one started by `\item`. Can be negative, but is usually 0pt.

`\itemindent` extra indentation added to the horizontal indentation of the text part of the first line of an item. The starting position of the label is calculated with respect to this reference point by subtracting the values of `\labelsep` and `\labelwidth`. Its value is usually 0pt.

`\labelwidth` the nominal width of the box containing the label. If the natural width of the label is  $\leq \labelwidth$ , then by default the label is typeset flush right inside a box of width `\labelwidth`. Otherwise, a box of the natural width is employed, which causes an indentation of the text on that line. It is possible to modify the way the label is typeset by providing a definition for the `\makelabel` command.

`\labelsep` the space between the end of the label box and the text of the first item. Its default value is 0.5em.

Figure 3.3: Parameters used by the list environment

or “footnote-sized” text. However, since this is a somewhat incomplete set-up, strange effects are possible if you

- Use nested lists in such small sizes (the nested lists get the standard defaults intended for `\normalsize`),
- Jump from `\small` or `\footnotesize` directly to a large size, such as `\huge` (a first-level list now inherits the defaults from the small size, since in this set-up `\huge` does not reset the list defaults).

With a more complex set-up these defects could be mended. However, since the simpler set-up works well in most practical circumstances, most classes provide only this restricted support.

*Global changes are difficult*

Because of this size- and nesting-dependent set-up for the list parameters, it is not possible to change any of them globally in the preamble of your document. For global changes you have to provide redefinitions for the various `\@list...` commands discussed above or select a different document class.

*Page breaking around lists*

Page breaking around and within a list structure is controlled by three TEX counters: `\@beginparpenalty` (for breaking before the list), `\@itempenalty` (for breaking before an item within the list), and `\@endparpenalty` (for breaking the page after a list). By default, all three are set to a slightly negative value, meaning that it is permissible (and even preferable) to break a page in these places compared to other break points. However, this outcome may not be appropriate. You may prefer to discourage or even prevent page breaks directly before a list. To achieve this, assign a high value to `\@beginparpenalty` (10000 or more prohibits the break in all circumstances), for example:

```
\makeatletter \@beginparpenalty=9999 \makeatother
```

*Many environments are implemented as lists*

TEX counters need this unusual assignment form and since all three contain an `@` sign in their name, you have to surround them with `\makeatletter` and `\makeatother` if the assignment is done in the preamble.

It is important to realize that such a setting is global to all environments based on the generic `list` environment (unless it is made in the `decls` argument) and that several LATEX environments are defined with the help of this environment (for example, `quote`, `quotation`, `center`, `flushleft`, and `flushright`). These environments are “lists” with a single item, and the `\item[]` command is specified in the environment definition. The main reason for them to be internally defined as lists is that they then share the vertical spacing with other display objects and thus help achieve a uniform layout.

As an example, we can consider the `quote` environment, whose definition gives the same left and right margins. The simple variant `Quote`, shown below, is identical to `quote` apart from the double quote symbols added around the text. Note the special precautions, which must be taken to eliminate undesirable white space in front of (`\ignorespaces`) and following (`\unskip`) the text. We also placed the quote characters into boxes of zero width to make the quotes hang into

the margin. (This trick is worth remembering: if you have a zero-width box and align the contents with the right edge, they will stick out to the left.)

```
\newenvironment{Quote}%
  {\begin{list}{}%
   \setlength\rightmargin{\leftmargin}%
   \item[]\makebox[0pt][r]{'}\ignorespaces}%
  {\unskip\makebox[0pt][l]{'}}\end{list}%
\ldots\text before.
\begin{Quote}
  "Some quoted text, followed
  by more quoted text."
\end{Quote}
Text following ...
```

3-3-25

Text following \ldots

In the remainder of this section we will construct a number of different “description” lists, thereby explaining the various possibilities offered by the generic `list` environment. We start by looking at the default definition of the `description` environment as it can be found in L<sup>A</sup>T<sub>E</sub>X’s standard classes such as `article` or `report`.<sup>1</sup>

```
\newenvironment{description}
  {\begin{list}{}%
   \setlength\labelwidth{0pt}%
   \setlength\itemindent{-\leftmargin}%
   \let\makelabel\descriptionlabel}%
\end{list}}
```

To understand the reasoning behind this definition recall Figure 3.3 on page 145, which explains the relationship between the various list parameters. The parameter settings start by setting `\labelwidth` to zero, which means that we do not reserve any space for the label. Thus, if the label is being typeset, it will move the text of the first line to the right to get the space it needs. Then the `\itemindent` parameter is set to the negation of `\leftmargin`. As a result, the starting point for the first text line is moved to the enclosing margin but all turnover lines are still indented by `\leftmargin`. The last declaration makes `\makelabel` identical to the command `\descriptionlabel`. The command `\makelabel` is called by the `list` environment whenever it has to format an item label. It takes one argument (the label) and is supposed to produce a typeset version of that argument. So the final task to finish the definition of the `description` environment is to provide a suitable definition for `\descriptionlabel`. This indirection is useful because it allows us to change the label formatting without modifying the rest of the environment definition.

How should `\descriptionlabel` be defined? It has to provide the formatting for the label. With the standard `description` environment this label is supposed

<sup>1</sup>If you look into `article.cls` or `report.cls` you will find a slightly optimized coding that uses, for example, low-level assignments instead of `\setlength`. However, conceptually, the definitions are identical.

to be typeset in boldface. But recall that the label is separated from the following text by a space of width `\labelsep`. Due to the parameter settings given above this text starts at the outer margin. Thus, without correction our label would end up starting in the margin (by the width of `\labelsep`). To prevent this outcome the standard definition for the `\descriptionlabel` command has the following curious definition, in that it first moves to the right and then typesets the label:

```
\newcommand*\descriptionlabel[1]
  {\hspace{\labelsep}\normalfont\bfseries #1}
```

To remove this dependency, one would need to change the setting of `\itemindent` to already take the `\labelsep` into account, which in itself would not be difficult. You may call this behavior an historical artifact, but many documents rely on this somewhat obscure feature. Thus, it is difficult to change the setting in the L<sup>A</sup>T<sub>E</sub>X kernel without breaking those documents.

With the parameter settings of the standard `description` environment, in case of short labels the text of the first line starts earlier than the text of remaining lines. If we always want a minimal indentation we can try a definition similar to the one in the following example, where we set `\labelwidth` to 40pt and `\leftmargin` to `\labelwidth` plus `\labelsep`. This means that `\makelabel` has to concern itself only with formatting the label. However, given that we now have a positive nominal label width, we need to define what should happen if the label is small. By using `\hfil` we specify where extra white space should be inserted.

```
\usepackage{calc}
\newenvironment{Description}
  {\begin{list}{}{\let\makelabel\Descriptionlabel
    \setlength\labelwidth{40pt}%
    \setlength\leftmargin{\labelwidth+\labelsep}}}{%
  \end{list}}
\newcommand*\Descriptionlabel[1]{\textsf{#1}\hfil}
\begin{Description}
\item[Description]
  Returns from a function. If issued at top level,
  the interpreter simply terminates, just as if
  end of input had been reached.
\item[Errors] None.
\item[Return values]
  \mbox{}\\
  Any arguments in effect are passed back to the
  caller.
\end{Description}
```

**Description:** Returns from a function.  
If issued at top level, the interpreter simply terminates, just as if end of input had been reached.

**Errors:** None.

**Return values:**

Any arguments in effect are passed back to the caller.

This example shows a typical problem with `description`-like lists when the text in the label (*term*) is wider than the width of the label. Our definition lets the text of the term continue into the text of the *description* part. This is often not

desired, and to improve the visual appearance of the list we have started one of the description parts on the next line. A new line was forced by putting an empty box on the same line, followed by the ‘\\’ command.

In the remaining part of this section various possibilities for controlling the width and mutual positioning of the term and description parts will be investigated. The first method changes the width of the label. The environment is declared with an argument specifying the desired width of the label field (normally chosen to be the widest term entry). Note the redefinition of the `\makelabel` command where you specify how the label will be typeset. As this redefinition is placed inside the definition<sup>1</sup> of the `altDescription` environment, the argument placeholder character # must be escaped to ## to signal L<sup>A</sup>T<sub>E</sub>X that you are referring to the argument of the `\makelabel` command, and not to the argument of the outer environment. In such a case, `\labelwidth` is set to the width of the environment’s argument after it is processed by `\makelabel`. This way formatting directives for the label that might change its width are taken into account.

<b>Description:</b> Returns from a function. If issued at top level, the interpreter simply terminates, just as if end of input had been reached.	<pre>\usepackage{calc} \newenvironment{altDescription}[1]   {\begin{list}{}%    \renewcommand{\makelabel}[1]{\textsf{\#\#1}\hfil}%    \setwidth{\labelwidth}{\makelabel{\#1}}%    \setlength{\leftmargin}{\labelwidth+\labelsep}}%   {\end{list}} \begin{altDescription}{Return values} \item[Description]   Returns from a function. If issued at top level,   the interpreter simply terminates, just as if end   of input had been reached. \item[Errors]   None. \item[Return values]   Any arguments in effect are passed back to the   caller. \end{altDescription}</pre>
<b>Errors:</b> None.	
<b>Return values:</b> Any arguments in effect are passed back to the caller.	

3-3-27

A similar environment (but using an optional argument) is shown in Example A-1-9 on page 850. However, having several lists with varying widths for the label field on the same page might look typographically unacceptable. Evaluating the width of the term is another possibility that avoids this problem. If the width is wider than `\labelwidth`, an additional empty box is appended with the effect that the description part starts on a new line. This matches the conventional method for displaying options in UN\*X manuals.

To illustrate this method we reuse the `Description` environment defined

<sup>1</sup>This is done for illustration purposes. Usually the solution involving an external name is preferable, as with `\Descriptionlabel` in Example 3-3-26 on the preceding page.

in Example 3-3-26 but provide a different definition for the `\Descriptionlabel` command as follows:

```
\usepackage{calc,ifthen} \newlength{\Mylen}
% definition of Description environment as before
\newcommand*\Descriptionlabel[1]{%
  \settowidth{\Mylen}{\textsf{\#1:}}% determine width
  \ifthenelse{\lengthtest{\Mylen > \labelwidth}}{%
    \parbox[b]{\labelwidth}{\textsf{\#1:}}% term > labelwidth
    {\makebox[0pt][l]{\textsf{\#1:}}}\,\,\,\mbox{}% term <= labelwidth
    \hfill}%
\begin{Description}
\item[Description] Returns from a function.
If issued at top level, the interpreter simply
terminates, just as if end of input had been reached.
\item[Errors] None.
\item[Return values]
Any arguments in effect are passed back to the caller.
\end{Description}
```

3-3-28

**Description:**

Returns from a function. If issued at top level, the interpreter simply terminates, just as if end of input had been reached.

**Errors:** None.

**Return values:**

Any arguments in effect are passed back to the caller.

The definition of `\Descriptionlabel` sets the length variable `\Mylen` equal to the width of the label. It then compares that length with `\labelwidth`. If the label is not wider than `\labelwidth`, then it is typeset on the same line as the description term. Otherwise, it is typeset in a zero-width box with the material sticking out to the right as far as needed. It is placed into a bottom-aligned `\parbox` followed by a forced line break so that the description term starts one line lower. This somewhat complicated maneuver is necessary because `\makelabel`, and thus `\Descriptionlabel`, are executed in a strictly horizontal context in which vertical spaces or `\`` commands have no effect.

Yet another possibility is to allow multiple-line labels.

```
\usepackage{calc}
% definition of Description environment as before
\newcommand*\Descriptionlabel[1]{%
  \raisebox{0pt}[1ex][0pt]{\makebox[\labelwidth][l]{\textsf{\#1:}}}\,\,\,\hbox{\hspace{0pt}\textsf{\#1:}}%
\begin{Description}
\item[Description] Returns from a function.
If issued at top level, the interpreter simply
terminates, just as if end of input had been reached.
\item[Errors] None.
\item[Return values]
Any arguments in effect are passed back to the caller.
\end{Description}}
```

3-3-29

**Description:** Returns from a function. If issued at top level, the interpreter simply terminates, just as if end of input had been reached.

**Errors:** None.

**Return values:** Any arguments in effect are passed back to the caller.

In the previous example, we once again used the `Description` environment as a basis, with yet another redefinition of the `\Descriptionlabel` command. The idea here is that large labels may be split over several lines. Certain precautions have to be taken to allow hyphenation of the first word in a paragraph, and therefore the `\hspace{0pt}` command is introduced in the definition. The material gets typeset inside a paragraph box of the correct width `\labelwidth`, which is then top and left aligned into a box that is itself placed inside a box with a height of `1ex` and no depth. In this way, L<sup>A</sup>T<sub>E</sub>X does not realize that the material extends below the first line.

The final example deals with the definition of enumeration lists. An environment with an automatically incremented counter can be created by including a `\usecounter` command in the declaration of the `list` environment. This function is demonstrated with the `Notes` environment, which produces a sequence of notes. In this case, the first parameter of the `list` environment is used to provide the automatically generated text for the term part.

After declaring the `notes` counter, the default label of the `Notes` environment is defined to consist of the word `NOTE` in small caps, followed by the value of the `notes` counter, using as its representation an Arabic numeral followed by a dot. Next `\labelsep` is set to a relatively large value and `\itemindent`, `\leftmargin`, and `\labelwidth` are adjusted in a way such that the label nevertheless starts out at the left margin. Finally, the already-mentioned `\usecounter` declaration ensures that the `notes` counter is incremented for each `\item` command.

```

\newcounter{notes}
\newenvironment{Notes}
  {\begin{list}{\textsc{Note} \arabic{notes}.}%
   \setlength{\labelsep{10pt}}%
   \setlength{\itemindent{10pt}}%
   \setlength{\leftmargin{0pt}}%
   \setlength{\labelwidth{0pt}}%
   \usecounter{notes}}{%
  \end{list}}
\begin{Notes}
\item This is the text of the first note item. Some more text for the first note item.
\item This is the text of the second note item. Some more text for the second note item.
\end{Notes}

```

NOTE 1. This is the text of the first note item. Some more text for the first note item.

NOTE 2. This is the text of the second note item. Some more text for the second note item.

## 3.4 Simulating typed text

It is often necessary to display information verbatim—that is, “as entered at the terminal”. This ability is provided by the standard L<sup>A</sup>T<sub>E</sub>X environment `verbatim`. However, to guide the reader it might be useful to highlight certain textual strings

in a particular way, such as by numbering the lines. Over time a number of packages have appeared that addressed one or the other extra feature—unfortunately, each with its own syntax.

In this section we will review a few such packages. Since they have been used extensively in the past, you may come across them in document sources on the Internet or perhaps have used them yourself in the past. But we then concentrate on the package `fancyvrb` written by Timothy Van Zandt, which combines all such features and many more under the roof of a single, highly customizable package.

This coverage is followed by a discussion of the `listings` package, which provides a versatile environment in which to pretty print computer listings for a large number of computer languages.

### 3.4.1 Simple verbatim extensions

The package `alltt` (by Leslie Lamport) defines the `alltt` environment. It acts like a `verbatim` environment except that the backslash “\” and braces “{” and “}” retain their usual meanings. Thus, other commands and environments can appear inside an `alltt` environment. A similar functionality is provided by the `fancyvrb` environment keyword `commandchars` (see page 161).

```
\usepackage{alltt}
\begin{alltt}
One can have font changes, like
emphasized text.
Some special characters: # $ % ^ & ~ _
\end{alltt}
```

3-4-1

In documents where a lot of `\verb` commands are needed the source soon becomes difficult to read. For this reason the `doc` package, described in Chapter 14, introduces a shortcut mechanism that lets you use a special character to denote the start and stop of verbatim text, without having to repeatedly write `\verb` in front of it. This feature is also available in a stand-alone package called `shortvrb`. With `fancyvrb` the same functionality is provided, unfortunately using a slightly different syntax (see page 168).

```
\usepackage{shortvrb}
\MakeShortVerb{\|} 
The use of |\MakeShortVerb| can make sources
much more readable.
\DeleteShortVerb{\|}\MakeShortVerb{\+}
And with the declaration +\DeleteShortVerb{\|}+
we can return the +|+ character back to normal.
```

3-4-2

The use of `\MakeShortVerb` can make sources much more readable. And with the declaration `\DeleteShortVerb{\|}` we can return the `|` character back to normal.

The variant form, `\MakeShortVerb*`, implements the same shorthand mechanism for the `\verb*` command. This is shown in the next example.

3-4-3

Instead of `\verb` we can now write `\verb+`.

```
\usepackage{shortvrb} \MakeShortVerb*{\+}
Instead of \verb*/ / we can now write + +.
```

The package `verbatim` (by Rainer Schöpf) reimplements the L<sup>A</sup>T<sub>E</sub>X environments `verbatim` and `verbatim*`. One of its major advantages is that it allows arbitrarily long `verbatim` texts, something not possible with the basic L<sup>A</sup>T<sub>E</sub>X versions of the environments. It also defines a `comment` environment that skips all text between the commands `\begin{comment}` and `\end{comment}`. In addition, the package provides hooks to implement user extensions for defining customized `verbatim`-like environments.

A few such extensions are realized in the package `moreverb` (by Angus Duggan). It offers some interesting `verbatim`-like commands for writing to and reading from files as well as several environments for the production of listings and dealing with tab characters. All of these extensions are also available in a consistent manner with the `fancyvrb` package, so here we only give a single example to show the flavor of the syntax used by the `moreverb` package.

3-4-4

Text before listing environment.

The `listing` environment numbers the lines in it. It takes an optional argument, which is the step between numbered lines (line 1 is always numbered if present), and a required argument, which is the starting line. The star form makes blanks visible.

Text between listing environments.

This `listingcont` environment continues where the previous listing environment left off. Both the listing and `listingcont` environments expand tabs with a default tab width of 8.

Text following listing environments.

```
\usepackage{verbatim,moreverb}
Text before listing environment.
\begin{listing*}[2]{3}
The listing environment numbers the
lines in it. It takes an optional
argument, which is the step between
numbered lines (line 1 is always
numbered if present), and a required
argument, which is the starting line.
The star form makes blanks visible.
\end{listing*}
Text between listing environments.
\begin{listingcont}
This listingcont environment continues
where the previous listing environment
left off. Both the listing and
listingcont environments expand tabs
with a default tab width of 8.
\end{listingcont}
Text following listing environments.
```

### 3.4.2 upquote—Computer program style quoting

The Computer Modern Typewriter font that is used by default for typesetting “`verbatim`” is a very readable monospaced typeface. Due to its small running length it is very well suited for typesetting computer programs and similar material. See Section 7.7.4 for a comparison of this font with other monospaced typefaces.

There is, however, one potential problem when using this font to render computer program listings and similar material: most people expect to see a (right) quote in a computer listing represented with a straight quote character (i.e., ') and a left or back quote as a kind of grave accent on its own (i.e., `). The Computer Modern Typewriter font, however, displays real left and right curly quote characters (as one would expect in a normal text font). In fact, most other typewriter fonts when set up for use with L<sup>A</sup>T<sub>E</sub>X follow this pattern. This produces somewhat unconventional results that many people find difficult to understand. Consider the following example, which shows the standard behavior for three major typewriter fonts: LuxiMono, Courier, and Computer Modern Typewriter.

```
\usepackage[scaled=0.85]{luximono}
\raggedright
\verb+TEST='ls -l |awk '{print $3}'+ +
\par \renewcommand\ttdefault{pcr} .
\verb+TEST='ls -l |awk '{print $3}'+ +
\par \renewcommand\ttdefault{cmtt} .
\verb+TEST='ls -l |awk '{print $3}'+ +
\par \texttt{but 'text' is unaffected!} 3-4-5
```

This behavior can be changed by loading the package `upquote` (written by Michael Covington), which uses the glyphs `\textasciigrave` and `\textquotesingle` from the `textcomp` package instead of the usual left and right curly quote characters within `\verb` or the `verbatim` environment. Normal typewriter text still uses the curly quotes, as shown in the last line of the example.

```
\usepackage[scaled=0.85]{luximono}
\usepackage{upquote}
\raggedright
\verb+TEST='ls -l |awk '{print $3}'+ +
\par \renewcommand\ttdefault{pcr} .
\verb+TEST='ls -l |awk '{print $3}'+ +
\par \renewcommand\ttdefault{cmtt} .
\verb+TEST='ls -l |awk '{print $3}'+ +
\par \texttt{but 'text' is unaffected!} 3-4-6
```

The package works well together with “verbatim” extensions as described in this chapter, except for the `listings` package; it conflicts with the scanning mechanism of that package. If you want this type of quoting with `listings` simply use the `\lstset` keyword `upquote`.

```
\usepackage{textcomp}
\usepackage{listings} \lstset{upquote}
\begin{lstlisting}[language=ksh]
TEST='ls -l |awk '{print $3}'+ +
\end{lstlisting} 3-4-7
```

### 3.4.3 fancyvrb—Highly customizable verbatim environments

The `fancyvrb` package by Timothy Van Zandt (these days maintained by Denis Girou and Sebastian Rahtz) offers a highly customizable set of environments and commands to typeset and manipulate verbatim text.

It works by parsing one line at a time from an environment or a file (a concept pioneered by the `verbatim` package), thereby allowing you to preprocess lines in various ways. By incorporating features found in various other packages it provides a truly universal production environment under a common set of syntax rules.

The main environment provided by the package is the `Verbatim` environment, which, if used without customization, is much like standard L<sup>A</sup>T<sub>E</sub>X's `verbatim` environment. The main difference is that it accepts an optional argument in which you can specify customization information using a key/value syntax. However, there is one restriction to bear in mind: the left bracket of the optional argument must appear on the same line as `\begin{Verbatim}`. Otherwise, the optional argument will not be recognized but instead typeset as verbatim text.

More than 30 keywords are available, and we will discuss their use and possible values in some detail.

A number of variant environments and commands will be discussed near the end of this section as well. They also accept customization via the key/value method. Finally, we cover possibilities for defining your own variants in a straightforward way.

#### Customization keywords for typesetting

To manipulate the fonts used by the verbatim environments of the `fancyvrb` package, four environment keywords, corresponding to the four axes of NFSS, are available. The keyword `fontfamily` specifies the font family to use. Its default is `Computer Modern Typewriter`, so that when used without keywords the environments behave in a fashion similar to standard L<sup>A</sup>T<sub>E</sub>X's `verbatim`. However, the value of this keyword can be any font family name in NFSS notation, such as `pcr` for `Courier` or `cmss` for `Computer Modern Sans`, even though the latter is not a monospaced font as would normally be used in a verbatim context. The keyword also recognizes the special values `tt`, `courier`, and `helvetica` and translates them internally into NFSS nomenclature.

Because typesetting of verbatim text can include special characters like “\” you must be careful to ensure that such characters are present in the font. This should be no problem when a font encoding such as `T1` is active, which could be loaded using the `fontenc` package. It is, however, not the case for L<sup>A</sup>T<sub>E</sub>X's default font encoding `OT1`, in which only some monospaced fonts, such as the default typewriter font, contain all such special characters. The type of incorrect output you might encounter is shown in the second line of the next example.

```

\usepackage{fancyvrb}
\usepackage[OT1,T1]{fontenc}
\fontencoding{OT1}\selectfont
\begin{Verbatim}[fontfamily=tt]
Family 'tt' is fine in OT1: \sum_{i=1}^n
\end{Verbatim}
\begin{Verbatim}[fontfamily=helvetica]
But 'helvetica' fails in OT1: \sum_{i=1}^n
\end{Verbatim}
\fontencoding{T1}\selectfont
\begin{Verbatim}[fontfamily=helvetica]
... while it works in T1: \sum_{i=1}^n
\end{Verbatim}

```

3-4-8

Family 'tt' is fine in OT1:  $\sum_{i=1}^n$   
 But 'helvetica' fails in OT1: "sum'-i=1"n  
 ... while it works in T1:  $\sum_{i=1}^n$

Since all examples in this book are typeset using the T1 encoding this kind of problem will not show up elsewhere in the book. Nevertheless, you should be aware of this danger. It represents another good reason to use T1 in preference to TeX's original font encoding; for a more in-depth discussion see Section 7.2.4 on page 336.

The other three environment keywords related to the font set-up are `fontseries`, `fontshape`, and `fontsize`. They inherit the current NFSS settings from the surrounding text if not specified. While the first two expect values that can be fed into `\fontseries` and `\fontshape`, respectively (e.g., `bx` for a bold extended series or `it` for an *italic* shape), the `fontsize` is special. It expects one of the higher-level NFSS commands for specifying the font size—for example, `\small`. If the `relsize` package is available then you could alternatively specify a change of font size relative to the current text font by using something like `\relsize{-2}`.

```

\usepackage{relsize,fancyvrb}
\begin{Verbatim}[fontsize=\relsize{-2}]
\sum_{i=1}^n
\end{Verbatim}
A line of text to show the body size.
\begin{Verbatim}[fontshape=s1,fontsize=\Large]
\sum_{i=1}^n
\end{Verbatim}

```

3-4-9

$\sum_{i=1}^n$   
 A line of text to show the body size.  
 $\sum_{i=1}^n$

A more general form for customizing the formatting is available through the environment keyword `formatcom`, which accepts any TeX code and executes it at the start of the environment. For example, to color the verbatim text you could pass it something like `\color{blue}`. It is also possible to operate on each line of text by providing a suitable redefinition for the command `\FancyVerbFormatLine`. This command is executed for every line, receiving the text from the line as its argument. In the next example every second line is

colored in blue, a result achieved by testing the current value of the counter `FancyVerbLine`. This counter is provided automatically by the environment and holds the current line number.

This line should become blue while  
this one will be black. And here  
you can observe that gobble removes  
not only blanks but any character.

3-4-10

```
\usepackage{ifthen,color,fancyvrb}
\renewcommand{\FancyVerbFormatLine}[1]
  {\ifthenelse{\isodd{\value{FancyVerbLine}}}{%
    \textcolor{blue}{#1}}{\begin{Verbatim}[gobble=2]
This line should become blue while
      this one will be black. And here
      you can observe that gobble removes
      not only blanks but any character.
\end{Verbatim}}
```

As shown in the previous example the keyword `gobble` can be used to remove a number of characters or spaces (up to nine) from the beginning of each line. This is mainly useful if all lines in your environments are indented and you wish to get rid of the extra space produced by the indentation. Sometimes the opposite goal is desired: every line should be indented by a certain space. For example, in this book all verbatim environments are indented by 24pt. This indentation is controlled by the keyword `xleftmargin`. There also exists a keyword `xrightmargin` to specify the right indentation, but its usefulness is rather limited, since verbatim text is not broken across lines. Thus, its only visible effect (unless you use frames, as discussed below) is potentially more overfull box messages<sup>1</sup> that indicate that your text overflows into the right margin. Perhaps more useful is the Boolean keyword `resetmargins`, which controls whether preset indentations by surrounding environments are ignored.

- Normal indentation left:

A verbatim line of text!

- No indentation at either side:

A verbatim line of text!

```
\usepackage{fancyvrb}
\begin{itemize} \item Normal indentation left:
  \begin{Verbatim}[frame=single,xrightmargin=2pc]
A verbatim line of text!
\end{Verbatim}
\item No indentation at either side:
  \begin{Verbatim}[resetmargins=true,
    frame=single]
A verbatim line of text!
\end{Verbatim}
\end{itemize}
```

The previous example demonstrates one use of the `frame` keyword: to draw a frame around verbatim text. By providing other values for this keyword, different

<sup>1</sup>Whether overfull boxes inside a verbatim environment are shown is controlled the `hfuzz` keyword, which has a default value of 2pt. A warning is issued only if boxes protrude by more than the keyword's value into the margin.

looking frames can be produced. The default is `none`, that is, no frame. With `topline`, `bottomline`, or `leftline` you get a single line at the side indicated;<sup>1</sup> `lines` produces a line at top and bottom; and `single`, as we saw in Example 3-4-11, draws the full frame. In each case, the thickness of the rules can be customized by specifying a value via the `framerule` keyword (default is 0.4pt). The separation between the lines and the text can be controlled with `framesep` (default is the current value of `\fboxsep`).

If the `color` package is available, you can color the rules using the environment keyword `rulecolor` (default is black). If you use a full frame, you can also color the separation between the frame and the text via `fillcolor`.

```
\usepackage{color,fancyvrb}
\begin{Verbatim}[frame=single,rulecolor=\color{blue},
    framerule=3pt,framesep=1pc,fillcolor=\color{yellow}]
A framed verbatim line!
\end{Verbatim}
```

3-4-12

Unfortunately, there is no direct way to fill the entire background. The closest you can get is by using `\colorbox` inside `\FancyVerbFormatLine`. But this approach will leave tiny white rules between the lines and—without forcing the lines to be of equal length, such as via `\makebox`—will also result in colored blocks of different widths.

```
\usepackage{color,fancyvrb}
\renewcommand{\FancyVerbFormatLine}[1]
  {\colorbox{green}{#1}}
\begin{Verbatim}
Some verbatim lines with a
background color.
\end{Verbatim}
\renewcommand{\FancyVerbFormatLine}[1]
  {\colorbox{yellow}{\makebox[\ linewidth][l]{#1}}}
\begin{Verbatim}
Some verbatim lines with a
background color.
\end{Verbatim}
```

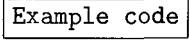
3-4-13

It is possible to typeset text as part of a frame by supplying it as the value of the `label` keyword. If this text contains special characters, such as brackets, equals sign, or comma, you have to hide them by surrounding them with a brace group. Otherwise, they will be mistaken for part of the syntax. The text appears by default at the top, but is printed only if the frame set-up would produce a line in that position. Alternate positions can be specified by using `labelposition`, which accepts `none`, `topline`, `bottomline`, or `all` as values. In the last case the text is printed above and below. If the label text is unusually large you may need

<sup>1</sup>There is no value to indicate a line at the right side.

to increase the separation between the frame and the verbatim text by using the keyword `framesep`. If you want to cancel a previously set label string, use the value `none`—if you really need “`none`” as a label string, enclose it in braces.

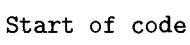
**3-4-14**

Some verbatim text framed		Some verbatim text framed
<b>Example code</b>		\end{Verbatim}

```
\usepackage{fancyvrb}
\begin{Verbatim}[frame=single,label=\fbox{Example code},
  framesep=5mm,labelposition=bottomline]
Some verbatim text framed
\end{Verbatim}
```

You can, in fact, provide different texts to be placed at top and bottom by surrounding the text for the top position with brackets, as shown in the next example. For this scheme to work `frame` needs to be set to either `single` or `lines`.

**3-4-15**

Start of code		A line of code
A line of code		\end{Verbatim}
End of code		\end{Verbatim}

```
\usepackage{fancyvrb}
\begin{Verbatim}[frame=lines,framesep=5mm,
  label={[Start of code]End of code}]
Start of code
A line of code
End of code
\end{Verbatim}
```

By default, the typeset output of the verbatim environments can be broken across pages by L<sup>A</sup>T<sub>E</sub>X if it does not fully fit on a single page. This is even true in cases where a frame surrounds the text. If you want to ensure that this cannot happen, set the Boolean keyword `samemode` to `true`.

The vertical spacing between lines in a verbatim environment is the same as in normal text, but if desired you can enlarge it by a factor using the keyword `baselinestretch`. Shrinking so that lines overlap is not possible. If you want to revert to the default line separation, use the string `auto` as a value.

**3-4-16**

This text is more or less double-spaced. See also the discussion about the setspace package elsewhere.	\begin{Verbatim}[baselinestretch=1.6] This text is more or less double-spaced. See also the discussion about the setspace package elsewhere. \end{Verbatim}
--	---

```
\usepackage{fancyvrb}
\begin{Verbatim}[baselinestretch=1.6]
This text is more or less double-spaced.
See also the discussion about the
setspace package elsewhere.
\end{Verbatim}
```

When presenting computer listings, it is often helpful to number some or all of the lines. This can be achieved by using the keyword `numbers`, which accepts `none`, `left`, or `right` as a value to control the position of the numbers. The distance between the number and the verbatim text is 12pt by default but it can be adjusted by specifying a different value via the keyword `numbersep`. Usually, numbering restarts at 1 with each environment, but by providing an explicit number with the keyword `firstnumber` you can start with any integer value, even a negative one. Alternatively, this keyword accepts the word `last` to indicate that numbering should resume where it had stopped in the previous Verbatim instance.

```
\usepackage{fancyvrb}
\begin{Verbatim}[numbers=left, numbersep=6pt]
Verbatim lines can be numbered
at either left or right.
\end{Verbatim}
Some intermediate text...
\begin{Verbatim}[numbers=left, firstnumber=last]
Continuation is possible too
as we can see here.
\end{Verbatim}
```

3-4-17

- 1 Verbatim lines can be numbered
- 2 at either left or right.
- 3 Some intermediate text...
- 4 Continuation is possible too
- 5 as we can see here.

Some people prefer to number only some lines, and the package caters to this possibility by providing the keyword `stepnumber`. If this keyword is assigned a positive integer number, then only line numbers being an integer multiple of that number will get printed. We already learned that the counter that is used internally to count the lines is called `FancyVerbLine`, so it comes as no surprise that the appearance of the numbers is controlled by the command `\theFancyVerbLine`. By modifying this command, special effects can be obtained; a possibility where the current chapter number is prepended is shown in the next example. It also shows the use of the Boolean keyword `numberblanklines`, which controls whether blank lines are numbered (default is `false`, i.e., to not number them).

```
\usepackage{fancyvrb}
\renewcommand{\theFancyVerbLine}{\footnotesize
    \thechapter.\arabic{FancyVerbLine}}
\begin{Verbatim}[numbers=left, stepnumber=2,
    numberblanklines=true]
Normally empty lines in
in a verbatim will not receive
numbers---here they do!

```

- 3.2 Normally empty lines in  
in a verbatim will not receive  
numbers---here they do!
- 3.4 Admittedly using stepnumber
- 3.6 with such a redefinition of  
`FancyVerbLine` looks a bit odd.

```
\end{Verbatim}
```

3-4-18

In some situations it helps to clearly identify white space characters by displaying all blanks as `\_`. This can be achieved with the Boolean keyword `showspaces` or, alternatively, the `Verbatim*` variant of the environment.

Another white space character, the tab, plays an important rôle in some programming languages, so there may be a need to identify it in your source. This is achieved with the Boolean keyword `showtabs`. The tab character displayed is defined by the command `\FancyVerbTab` and can be redefined, as seen below. By default, tab characters simply equal eight spaces, a value that can be changed with the keyword `tabsize`. However, if you set the Boolean keyword `obeytabs` to `true`, then each tab character produces as many spaces as necessary to move to the next

integer multiple of `tabsize`. The example input contains tabs in each line that are displayed on the right as spaces with the default `tabsize` of 8. Note in particular the difference between the last input and output line.

```
\usepackage{fancyvrb}
\begin{Verbatim}[showtabs=true]
123456789012345678901234567890
Two      default tabs
\end{Verbatim}
\begin{Verbatim}[obeytabs=true,showtabs=true]
Two      real    tabs
\end{Verbatim}
123456789012345678901234567890
Two      >default   >tabs
Two      >real     >tabs
Two      >new      >tabs
3-4-19 Using>a >special tab>size
\begin{Verbatim}[obeytabs=true,tabsize=3,showtabs=true]
Using a special tab size
\end{Verbatim}
```

If you wish to execute commands within the verbatim text, then you need one character to act as an escape character (i.e., to denote the beginning of a command name) and two characters to serve as argument delimiters (i.e., to play the rôle that braces normally play within L<sup>A</sup>T<sub>E</sub>X). Such special characters can be specified with the `commandchars` keyword as shown below; of course, these characters then cannot appear as part of the verbatim text. The characters are specified by putting a backslash in front of each one so as to mask any special meaning they might normally have in L<sup>A</sup>T<sub>E</sub>X. The keyword `commentchar` allows you to define a comment character, which will result in ignoring everything following it until and including the next new line. Thus, if this character is used in the middle of a line, this line and the next will be joined together. If you wish to cancel a previous setting for `commandchars` or `commentchar`, use the string value “none”.

```
\usepackage{fancyvrb}
\begin{Verbatim}[commandchars=\|\|[],commentchar=!]
We can \emph{emphasize} text
! see above (this line is invisible)
Line with label\label[linea] ! removes new line
is shown here.
\end{Verbatim}
We can emphasize text
Line with label is shown here.
3-4-20 On line 2 we see...
On line~\ref{linea} we see\ldots
```

If you use `\label` within the verbatim environment, as was done in the previous example, it will refer to the internal line number whether or not that number is displayed. This requires the use of the `commandchars` keyword, a price you might consider too high because it deprives you of the use of the chosen characters in your verbatim text.

Two other keywords let you change the parsing and manipulation of verbatim data: `codes` and `defineactive`. They allow you to play some devious tricks but their use is not so easy to explain: one needs a good understanding of TeX's inner workings. If you are interested, please check the documentation provided with the `fancyvrb` package.

### Limiting the displayed data

Normally, all lines within the verbatim environment are typeset. But if you want to display only a subset of lines, you have a number of choices. With the keywords `firstline` and `lastline`, you can specify the start line and (if necessary) the final line to typeset. Alternatively, you can specify a start and stop string to search for within the environment body, with the result that all lines between (but this time *not* including the special lines) will be typeset. The strings are specified in the macros `\FancyVerbStartString` and `\FancyVerbStopString`. To make this work you have to be a bit careful: the macros need to be defined with `\newcommand*` and redefined with `\renewcommand*`. Using `\newcommand` will *not* work! To cancel such a declaration is even more complicated: you have to `\let` the command to `\relax`, for example,

```
\let\FancyVerbStartString\relax
```

or ensure that your definition is confined to a group—everything else fails.

```
\usepackage{fancyvrb}
\newcommand*\FancyVerbStartString{START}
\newcommand*\FancyVerbStopString{STOP}
\begin{Verbatim}
    A verbatim line not shown.
START
    Only the third line is shown.
STOP
    But the remainder is left out.
\end{Verbatim}
```

Only the third line is shown.

3-4-21

You may wonder why one would want to have such functionality available, given that one could simply leave out the lines that are not being typeset. With an environment like `Verbatim` they are indeed of only limited use. However, when used together with other functions of the package that write data to files and read it back again, they offer powerful solutions to otherwise unsolvable problems.

*How the book  
examples have been  
produced*

For instance, all examples in this book use this method. The example body is written to a file together with a document preamble and other material, so that the resulting file will become a processable L<sup>A</sup>T<sub>E</sub>X document. This document is then externally processed and included as an EPS graphic image into the book. Beside it, the sample code is displayed by reading this external file back in but displaying only those lines that lie between the strings `\begin{document}`

and `\end{document}`. This accounts for the example lines you see being typeset in black. The preamble part, which is shown in blue, is produced in a similar fashion: for this the start and stop strings are redefined to include only those lines lying between the strings `\StartShownPreambleCommands` and `\StopShownPreambleCommands`. When processing the example externally, these two commands are simply no-ops; that is, they are defined by the “example” class (which is otherwise close to the `article` document class) to do nothing. As a consequence, the example code will always (for better or worse) correspond to the displayed result.<sup>1</sup>

To write data verbatim to a file the environment `VerbatimOut` is available. It takes one mandatory argument: the file name into which to write the data. There is, however, a logical problem if you try to use such an environment inside your own environments: the moment you start the `VerbatimOut` environment, everything is swallowed without processing and so the end of your environment is not recognized. As a solution the `fancyvrb` package offers the command `\VerbatimEnvironment`, which, if executed within the `\begin` code of your environment, ensures that the end tag of your environment will be recognized in verbatim mode and the corresponding code executed.

To read data verbatim from a file, the command `\VerbatimInput` can be used. It takes an optional argument similar to the one of the `Verbatim` environment (i.e., it accepts all the keywords discussed previously) and a mandatory argument to specify the file from which to read. The variant `\BVerbatimInput` puts the typeset result in a box without space above and below. The next example demonstrates some of the possibilities: it defines an environment `example` that first writes its body verbatim to a file, reads the first line back in and displays it in blue, reads the file once more, this time starting with the second line, and numbers the lines starting with the number 1. As explained above, a similar, albeit more complex definition was used to produce the examples in this book.

```
\usepackage{fancyvrb,color}
\newenvironment{example}
  {\VerbatimEnvironment\begin{VerbatimOut}{test.out}}
  {\end{VerbatimOut}\noindent
   \BVerbatimInput[lastline=1,formatcom=\color{blue}]{test.out}%
   \VerbatimInput[numbers=left,firstnumber=1,firstline=2]{test.out}}
\begin{example}
A blue line. A blue line.
Two lines
1 Two lines with numbers.
2 with numbers. \end{example}
```

3.4-22

An interesting set of sample environments can be found in the package `fverb-ex` written by Denis Girou, which builds on the features provided by `fancyvrb`.

<sup>1</sup>In the first edition we unfortunately introduced a number of mistakes when showing code in text that was not directly used.

### Variant environments and commands

So far, all examples have used the `\Verbatim` environment, but there also exist a number of variants that are useful in certain circumstances. `\BVerbatim` is similar to `\Verbatim` but puts the verbatim lines into a box. Some keywords discussed above (notably those dealing with frames) are not supported, but two additional ones are available. The first, `baseline`, denotes the alignment point for the box; it can take the values `t` (for top), `c` (for center), or `b` (for bottom—the default). The second, `boxwidth`, specifies the desired width of the box; if it is missing or given the value `auto`, the box will be as wide as the widest line present in the environment. We already encountered `\BVerbatimInput`; it too, supports these additional keywords.

```
\usepackage{fancyvrb}
\begin{BVerbatim}[boxwidth=4pc,baseline=t]
first line
second line
\end{BVerbatim}
\begin{BVerbatim}[baseline=c]
first line
second line
\end{BVerbatim}
```

first line  
second line

3-4-23

All environments and commands for typesetting verbatim text also have star variants, which, as in the standard L<sup>A</sup>T<sub>E</sub>X environments, display blanks as `\` . In other words, they internally set the keyword `showspaces` to true.

### Defining your own variants

Defining customized variants of verbatim commands and environments is quite simple. For starters, the default settings built into the package can be changed with the help of the `\fvset` command. It takes one argument, a comma-separated list of key/value pairs. It applies them to every verbatim environment or command. Of course, you can still overwrite the new defaults with the optional argument on the command or environment. For example, if nearly all of your verbatim environments are indented by two spaces, you might want to remove them without having to deploy `gobble` on each occasion.

```
\usepackage{fancyvrb} \fvset{gobble=2}
\noindent A line of text to show the left margin.
\begin{Verbatim}
The new ‘normal’ case.
\end{Verbatim}
\begin{Verbatim}[gobble=0]
We now need to explicitly
cancel gobble occasionally!
\end{Verbatim}
```

A line of text to show the left margin.

The new ‘normal’ case.

We now need to explicitly  
cancel gobble occasionally!

3-4-24

However, `\fvset` applies to all environments and commands, which may not be what you need. So the package offers commands to define your own verbatim environments and commands or to modify the behavior of the predefined ones.

```
\CustomVerbatimEnvironment  {new-env}{base-env}{key/val-list}
\RecustomVerbatimEnvironment{change-env}{base-env}{key/val-list}
\CustomVerbatimCommand    {new-cmd}{base-cmd}{key/val-list}
\RecustomVerbatimCommand {change-cmd}{base-cmd}{key/val-list}
```

These declarations take three arguments: the name of the new environment or command being defined, the name of the environment or command (without a leading backslash) on which it is based, and a comma-separated list of key/value pairs that define the new behavior. To define new structures, you use `\CustomVerbatimEnvironment` or `\CustomVerbatimCommand` and to change the behavior of existing environments or commands (predefined ones as well as those defined by you), you use `\RecustomVerbatimEnvironment` or `\RecustomVerbatimCommand`. As shown in the following example, the default values, set in the third argument, can be overwritten as usual with the optional argument when the environment or command is instantiated.

**1** The normal case with thick  
**2** rules and numbers on the left.

The exception without numbers  
and thinner rules.

**1** And from here on the environment  
**2** behaves differently again.

```
\usepackage{fancyvrb}
\CustomVerbatimEnvironment{myverbatim}{Verbatim}
  {numbers=left,frame=lines,framerule=2pt}
\begin{myverbatim}
The normal case with thick
rules and numbers on the left.
\end{myverbatim}
\begin{myverbatim}[numbers=none,framerule=.6pt]
The exception without numbers
and thinner rules.
\end{myverbatim}
\RecustomVerbatimEnvironment{myverbatim}{Verbatim}
  {numbers=left,frame=none,showspaces=true}
\begin{myverbatim}
And from here on the environment
behaves differently again.
\end{myverbatim}
```

### Miscellaneous features

`\verb`'s standard `\verb` command normally cannot be used inside arguments, because in such places the parsing mechanism would go astray, producing incorrect results or error messages. A solution to this problem is to process the verbatim data outside the argument, save it, and later use the already parsed data in such dangerous places. For this purpose the `fancyvrb` package offers the commands `\SaveVerb` and `\UseVerb`.

```
\SaveVerb[key/val-list]{label}=data= \UseVerb*[key/val-list]{label}
```

The command `\SaveVerb` takes one mandatory argument, a *label* denoting the storage bin in which to save the parsed data. It is followed by the verbatim *data* surrounded by two identical characters (= in the syntax example above), in the same way that `\verb` delimits its argument. To use this data you call `\UseVerb` with the *label* as the mandatory argument. Because the data is only parsed but not typeset by `\SaveVerb`, it is possible to influence the typesetting by applying a list of key/value pairs or a star as with the other verbatim commands and environments. Clearly, only a subset of keywords make sense, irrelevant ones being silently ignored. The `\UseVerb` command is unnecessarily fragile, so you have to `\protect` it in moving arguments.

## Contents

### 1 Real \danger

`Real\danger` is no longer dangerous and can be reused as often as desired.

```
\usepackage{fancyvrb}
\SaveVerb{danger}=Real \danger=
\tableofcontents
\section{\protect\UseVerb{danger}}
\UseVerb*{danger} is no longer dangerous
and can\marginpar{\UseVerb[fontsize=\tiny]
{danger}}
be reused as often as desired.
```

3-4-26

It is possible to reuse such a storage bin when it is no longer needed, but if you use `\UseVerb` inside commands that distribute their arguments over a large distance you have to be careful to ensure that the storage bin still contains the desired contents when the command finally typesets it. In the previous example we placed `\SaveVerb` into the preamble because the use of its storage bin inside the `\section` command eventually results in an execution of `\UseVerb` inside the `\tableofcontents` command.

`\SaveVerb` also accepts an optional argument in which you can put key/value pairs, though again only a few are relevant (e.g., those dealing with parsing). There is one additional keyword `aftersave`, which takes code to execute immediately after saving the verbatim text into the storage bin. The next example shows an application of this keyword: the definition of a special variant of the `\item` command that accepts verbatim text for display in a `description` environment. It also supports an optional argument in which you can put a key/value list to influence the formatting. The definition is worth studying, even though the amount of mixed braces and brackets seems distressingly complex at first. They are necessary to ensure that the right brackets are matched by `\SaveVerb`, `\item`, and `\UseVerb`—the usual problem, since brackets do not nest like braces do in TeX.<sup>1</sup> Also note the use of `\textnormal`, which is needed to cancel the `\bfseries` implicitly issued

<sup>1</sup>The author confesses that it took him three trials (close to midnight) to make this example work.

by the `\item` command. Otherwise, the `\emph` command in the example would not show any effect since no Computer Modern bold italic face exists.

```
\usepackage{fancyvrb}
\ddanger Dangerous beast; \newcommand{\item[1]}{\SaveVerb{commandchars=\|\<\>,%
  aftersave={\item[\textnormal{\UseVerb[#1]{vsave}}]}\{vsave\}}}
\begin{description}
\item+\ddanger+ Dangerous beast;\\ found in \TeX books.
\item[fontsize=tiny]+\danger+ Its small brother,
  still dangerous.
\item+\dddanger{arg}+ The ultimate horror.
\end{description}
```

3-4-27

In the same way you can save whole verbatim environments using the environment `\SaveVerbatim`, which takes the name of a storage bin as the mandatory argument. To typeset them, `\UseVerbatim` or `\BUseVerbatim` (boxed version) with the usual key/value machinery can be used.

Even though verbatim commands or environments are normally not allowed inside footnotes, you do not need to deploy `\SaveVerb` and the like to get verbatim text into such places. Instead, place the command `\VerbatimFootnotes` at the beginning of your document (following the `\preamble!`) and from that point onward, you can use verbatim commands directly in footnotes. However, this was only implemented for footnotes—for other commands, such as `\section`, you still need the more complicated storage bin method described above.

```
A bit of text to give us a reason to \usepackage{fancyvrb}
use a footnote.1 Was this good enough? \VerbatimFootnotes
A bit of text to give us a reason to use a
footnote.\footnote{Here is proof: \verb=\danger{%^}=}
Was this good enough?
```

3-4-28

<sup>1</sup>Here is proof: `\danger{%^}`

The `fancyvrb` version of `\verb` is called `\Verb`, and it supports all applicable keywords, which can be passed to it via an optional argument as usual. The example below creates `\verbx` as a variant of `\Verb` with a special setting of `commandchars` so that we can execute commands within its argument. We have to use `\CustomVerbatimCommand` for this purpose, since `\verbx` is a new command not available in standard `LATEX`.

```
\usepackage{fancyvrb}
\CustomVerbatimCommand{\verbx}{Verb}{commandchars=\|\<\>}
\realdanger{| emph<arg>} \Verb[fontfamily=courier]+\realdanger{| emph<arg>}+ \\
\realdanger{arg} \verbx[fontfamily=courier]+\realdanger{| emph<arg>}+
```

3-4-29

As already mentioned, `fancyvrb` offers a way to make a certain character denote the start and stop of verbatim text without the need to put `\verb` in front. The command to declare such a delimiting character is `\DefineShortVerb`.

Like other `fancyvrb` commands it accepts an optional argument that allows you to set key/value pairs. These influence the formatting and parsing, though this time you cannot overwrite your choices on the individual instance. Alternatively, `\fvset` can be used, since it works on all verbatim commands and environments within its scope. To remove the special meaning from a character declared with `\DefineShortVerb`, use `\UndefineShortVerb`.

```
\usepackage{fancyvrb}
\DefineShortVerb[\fontsize=\tiny]{\|}%
The use of |\DefineShortVerb| can make sources
much more readable---or unreadable! \par
\UndefineShortVerb{\|}\DefineShortVerb{\+}%
\fvset{fontfamily=courier}%
And with +\UndefineShortVerb{\|}+
we can return the +|+ character back to normal.
```

3-4-30

The use of `\DefineShortVerb` can make sources much more readable—or unreadable!

And with `\UndefineShortVerb{\|}` we can return the | character back to normal.

Your favorite extensions or customizations can be grouped in a file with the name `fancyvrb.cfg`. After `fancyvrb` finishes loading, the package will automatically search for this file. The advantage of using such a file, when installed in a central place, is that you do not have to put your extensions into all your documents. The downside is that your documents will no longer be portable unless you distribute this file in tandem with them.

### 3.4.4 listings—Pretty-printing program code

A common application of verbatim typesetting is presenting program code. While one can successfully deploy a package like `fancyvrb` to handle this job, it is often preferable to enhance the display by typesetting certain program components (such as keywords, identifiers, and comments) in a special way.

Two major approaches are possible: one can provide commands to identify the logical aspects of algorithms or the programming language, or the application can (try to) analyze the program code behind the scenes. The advantage of the first approach is that you have potentially more control over the presentation; however, your program code is intermixed with `TEX` commands and thus may be difficult to maintain, unusable for direct processing, and often rather complicated to read in the source. Examples of packages classified into this category are `alg` and `algorithmic`. Here is an example:

```
if i ≤ 0 then
    i ← 1
else
    if i ≥ 0 then
        i ← 0
    end if
end if
```

```
\usepackage{algorithmic}
\begin{algorithmic}
\IF { $i \leq 0$ } \STATE $i \gets 1$ \ELSE
\IF { $i \geq 0$ } \STATE $i \gets 0$ \ENDIF
\ENDIF
\end{algorithmic}
```

3-4-31

ABAP (R/2 4.3, R/2 5.0, R/3 3.1, R/3 4.6C, R/3 6.10)	Haskell	PHP
ACSL	HTML	PL/I
Ada (83, 95)	IDL (empty, CORBA)	POV
Algol (60, 68)	Java (empty, AspectJ)	Prolog
Assembler (x86masm)	ksh	Python
Awk (gnu, POSIX)	Lisp (empty, Auto)	R
Basic (Visual)	Logo	Reduce
C (ANSI, Objective, Sharp)	Make (empty, gnu)	S (empty, PLUS)
C++ (ANSI, GNU, ISO, Visual)	Mathematica (1.0, 3.0)	SAS
Caml (light, Objective)	Matlab	Scilab
Clean	Mercury	SHELXL
Cobol (1974, 1985, ibm)	MetaPost	Simula (67, CII, DEC, IBM)
Comal 80	Miranda	SQL
csh	Mizar	tcl (empty, tk)
Delphi	ML	TeX (AlLaTeX, common, LaTeX, plain, primitive)
Eiffel	Modula-2	VBScript
Elan	MuPAD	Verilog
erlang	NASTRAN	VHDL (empty, AMS)
Euphoria	Oberon-2	VRML (97)
Fortran (77, 90, 95)	OCL (decorative, OMG)	XML
GCL	Octave	
Gnuplot	Pascal (Borland6, Standard, XSC)	
	Perl	

*blue indicates default dialect*

Table 3.7: Languages supported by `listings` (Winter 2003)

The second approach is exemplified in the package `listings`<sup>1</sup> written by Carsten Heinz. This package first analyzes the code, decomposes it into its components, and then formats those components according to customizable rules. The package parser is quite general and can be tuned to recognize the syntax of many different languages (see Table 3.7). New languages are regularly added, so if your target language is not listed it might be worth checking the latest release of the package on CTAN. You may even consider contributing the necessary declarations yourself, which involves some work but is not very difficult.

The user commands and environments in this package share many similarities with those in `fancyvrb`. Aspects of parsing and formatting are controlled via key/value pairs specified in an optional argument, and settings for the whole document or larger parts of it can be specified using `\lstset` (the corresponding `fancyvrb` command is `\fvset`). Whenever appropriate, both packages use the same keywords so that users of one package should find it easy to make the transition to the other.

<sup>1</sup>The package version described here is 1.0. Earlier releases used a somewhat different syntax in some cases, so please upgrade if you find that certain features do not work as advertised.

After loading the package it is helpful to specify all program languages needed in the document (as a comma-separated list) using `\lstloadlanguages`. Such a declaration does not select a language, but merely loads the necessary support information and speeds up processing.

Program fragments are included inside a `lstlisting` environment. The language of the fragment is specified with the `language` keyword. In the following example we set this keyword via `\lstset` to C and then overwrite it later in the optional argument to the second `lstlisting` environment.

```
\usepackage{listings}
\lstloadlanguages{C,Ada}
\lstset{language=C,commentstyle=\scriptsize}
A ``for'' loop in C:
\begin{lstlisting}[keywordstyle=\underline]
int sum;
int i; /* for loop variable */
sum=0;
for (i=0;i<n;i++) {
    sum += a[i];
}
\end{lstlisting}
Now the same loop in Ada:
Sum: Integer;
-- no decl for I necessary
Sum := 0;
for I in 1..N loop
    Sum := Sum + A(I);
end loop;
\end{lstlisting}
```

3-4-32

This example also uses the keyword `commentstyle`, which controls the layout of comments in the language. The package properly identifies the different syntax styles for comments. Several other such keywords are available as well—`basicstyle` to set the overall appearance of the listing, `stringstyle` to format strings in the language, and `directivestyle` to format compiler directives, among others.

To format the language keywords, `keywordstyle` and `ndkeywordstyle` (second order) are used. Other identifiers are formatted according to the setting of `identifierstyle`. The values for the “style” keywords (except `basicstyle`) accept a one-argument L<sup>A</sup>T<sub>E</sub>X command such as `\textbf` as their last token. This scheme works because the “identifier text” is internally surrounded by braces and can thus be picked up by a command with an argument.

Thus, highlighting of keywords, identifiers, and other elements is done automatically in a customizable way. Nevertheless, you might want to additionally emphasize the use of a certain variable, function, or interface. For this purpose

you can use the keywords `emph` and `emphstyle`. The first gets a list of names you want to emphasize; the second specifies how you want them typeset.

```
\usepackage{listings,color}
\lstset{emph={Sum,N},emphstyle=\color{blue},
       emph=[2]I,emphstyle=[2]\underbar}
\begin{lstlisting}[language=Ada]
Sum: Integer;   Sum := 0;
for I in 1..N loop
    Sum := Sum + A(I);
end loop;
\end{lstlisting}
```

If you want to typeset a code fragment within normal text you can use the command `\lstinline`. The code is delimited in the same way as with the `\verb` command, meaning that you can choose any character (other than the open bracket) that is not used within the code fragment and use it as delimiter. An open bracket cannot be used because the command also accepts an optional argument in which you can specify a list of key/value pairs.

**3-4-34** The `for` loop is specified as `i=0;i<n;i++`.

```
\usepackage{listings} \lstset{language=C}
The \lstinline[keywordstyle=\underbar]!for!
loop is specified as \lstinline!i=0;i<n;i++!.
```

Of course, it is also possible to format the contents of whole files; for this purpose you use the command `\lstinputlisting`. It takes an optional argument in which you can specify key/value pairs and a mandatory argument in which you specify the file name to process. In the following example, the package identifies keywords of case-insensitive languages, even if they are written in an unusual mixed-case (`WRITe`) manner.

```
\usepackage{listings}
\begin{filecontents*}{pascal.src}
for i:=1 to maxint do
begin
    WRITe('This is stupid');
end.
\end{filecontents*}
\lstinputlisting[language=Pascal]{pascal.src}
```

**3-4-35**

Spaces in strings are shown as `_` by default. This behavior can be turned off by setting the keyword `showstringspaces` to `false`, as seen in the next example. It is also possible to request that all spaces be displayed in this way by setting the keyword `showspaces` to `true`. Similarly, tab characters can be made visible by using the Boolean keyword `showtabs`.

Line numbering is possible, too, using the same keywords as employed with `fancyvrb`: `numbers` accepts either `left`, `right`, or `none` (which turns numbering on or off), `numberblanklines` decides whether blank lines count with respect to numbering (default `false`), `numberstyle` defines the overall look and feel of the numbers, `stepnumber` defines which line numbers will appear (`0` means no numbering), and `numbersep` defines the separation between numbers and the start of the line. By default, line numbering starts with `1` on each `\lstinputlisting` but this can be changed using the `firstnumber` keyword. If you specify `last` as a special value to `firstnumber`, numbering is continued.

```
\usepackage{listings}
% pascal.src as defined before
\lstset{numberstyle=\tiny, numbers=left,
        stepnumber=2, numbersep=5pt, firstnumber=10,
        xleftmargin=12pt, showstringspaces=false}
\noindent Some text before \ldots
\lstinputlisting[language=Pascal]{pascal.src}
```

3-4-36

An overall indentation can be set using the `xleftmargin` keyword, as shown in the previous example, and `gobble` can be used to remove a certain number of characters (hopefully only spaces) from the left of each line displayed. Normally, indentations of surrounding environments like `itemize` will be honored. This feature can be turned off using the Boolean keyword `resetmargin`. Of course, all such keywords can be used together. To format only a subrange of the code lines you can specify the first and/or last line via `firstline` and `lastline`; for example, `lastline=10` would typeset a maximum of 10 code lines.

Another way to provide continued numbering is via the `name` keyword. If you define “named” environments using this keyword, numbering is automatically continued with respect to the previous environment with the same name. This allows independent numbering if the need arises.

```
\usepackage{listings} \lstset{language=Ada, numbers=right,
    numberstyle=\tiny, stepnumber=1, numbersep=5pt}
\begin{lstlisting}[name=Test]
Sum: Integer;           1 Sum: Integer;
The second fragment continues the numbering.          \end{lstlisting}
\begin{lstlisting}[name=Test]
Sum := 0;               2 for I in 1..N loop
for I in 1..N loop     3   Sum := Sum + A(I);
  Sum := Sum + A(I);  4   end loop;
end loop;              5 \end{lstlisting}
```

3-4-37

If a listing contains very long lines they may not fit into the available measure. In that case `listings` will produce overfull lines sticking out to the right, just

like a `verbatim` environment would do. However, you can direct it to break long lines at spaces or punctuation characters by specifying the keyword `breaklines`. Wrapped lines are indented by 20pt, a value that can be adjusted through the keyword `breakindent`.

If desired, you can add something before (keyword `prebreak`) and after (keyword `postbreak`) the break to indicate that the line was artificially broken in the listing. We used this ability below to experiment with small arrows and later on with the string “(cont.)” in tiny letters. Both keywords are internally implemented as a TeX \discretionary, which means that they accept only certain input (characters, boxes, and kerns). For more complicated material it would be best to wrap everything in an `\mbox`, as we did in the example. In case of color changes, even that is not enough: you need an extra level of braces to prevent the color `\special` from escaping from the box (see the discussion in Appendix A.2.5).

The example exhibits another feature of the breaking mechanism—namely, if spaces or tabs appear in front of the material being broken, then these spaces are by default repeated on continuation lines. If this behavior is not desired, set the keyword `breakautoindent` to `false` as we did in the second part of the example.

```
3-4-38
\usepackage{color, listings}
Text at left margin
/*A long \
→ string is \
→ broken \
→ across the \
→ line !*/
\begin{lstlisting}
Text at left margin
    /*A long string is broken across the line!*/
\end{lstlisting}
\begin{lstlisting}[breakautoindent=false,
    postbreak=\tiny (cont.)\.,]
    /*A long string is broken across the line!*/
\end{lstlisting}
```

You can put frames or rules around listings using the `frame` keyword, which takes the same values as it does in `fancyvrb` (e.g., `single`, `lines`). In addition, it accepts a subset of the string `trblTRBL` as its value. The uppercase letters stand for double rules the lowercase ones for single rules. There are half a dozen more keywords: to influence rule widths, create separation from the text, make round corners, and so on—all of them are compatible with `fancyvrb` if the same functionality is provided.

```
for _i:=1 _to _maxint _do
begin
  _Write( ' This _is _stupid ' );
end .
```

```
3-4-39
\usepackage{listings}
% pascal.src as defined before
\lstset{frame=trBL,framerule=2pt,framesep=4pt,
        rulesep=1pt,showspaces=true}
\lstinputlisting[language=Pascal]{pascal.src}
```

You can specify a caption for individual listings using the keyword `caption`. The captions are, by default, numbered and prefixed with the string `Listing` stored in `\lstlistingname`. The counter used is `lstlisting`; thus, to change its appearance you could modify `\thelstlisting`. The caption is positioned either above (default) or below the listing, and this choice can be adjusted using the keyword `captionpos`.

To get a list of all captions, put the command `\lstlistoflistings` at an appropriate place in your document. It produces a heading containing the words stored in `\lstlistlistingname` (default is `Listings`). If you want the caption text in the document to differ from the caption text in the list of listings, use an optional argument as shown in the following example. Note that in this case you need braces around the value to hide the right bracket. To prevent the caption from appearing in the list of listings, use the keyword `nolol` with a value of `true`. By using the keyword `label` you can specify a label for referencing the listing number via `\ref`, provided you have not suppressed the number.

## Listings

1 Pascal listing . . . . . 6

The Pascal code in listing 1 shows...

```
for i:=1 to maxint do
begin
  Writeln('This_is_stupid');
end.
```

Listing 1: Pascal

```
\usepackage{listings}
% pascal.src as defined before
\lstset{frame=single,frameround=tftt,
        language=Pascal,captionpos=b}
\lstlistoflistings
%
\bigskip % normally the above is in the
\noindent % front matter section, but here ...
%
The Pascal code in listing~\ref{foo} shows\ldots
\lstinputlisting
[caption={[Pascal listing]Pascal},label=foo]
[pascal.src]
```

[3-4-40]

The keyword `frameround` used in the previous example allows you to specify round corners by giving `t` for true and `f` for false, starting with the upper-right corner and moving clockwise. This feature is not available with `fancyvrb` frames.

Instead of formatting your listings within the text, you can turn them into floats by using the keyword `float`, typically together with the `caption` keyword. Its value is a subset of `htbp` specifying where the float is allowed to go (using it without a value is equivalent to `tbp`). You should, however, avoid mixing floating and nonfloating listings as this could sometimes result in captions being numbered out of order, as in Example 6-3-5 on page 296.

By default, `listings` only deals with input characters in the ASCII range; unexpected 8-bit input can produce very strange results, like the misordered letters in the following example. By setting `extendedchars` to `true` you can enable the use of 8-bit characters, which makes the package work harder, but (usually) produces

the right results. Of course, if you use an extended character set you would normally add the keyword to the `\lstset` declaration instead of specifying it every time on the environment. It is also possible to specify an input encoding for the code fragments (if different from the input encoding used for the remainder of the document) by using the keyword `inputencoding`. This keyword can be used only if the `inputenc` package is loaded.

```
\usepackage[latin1]{inputenc}
\usepackage{listings}
\lstset{language=C,commentstyle=\scriptsize}
\begin{lstlisting}
int i; /*für die äußere Schleife*/
\end{lstlisting}
\begin{lstlisting}[extendedchars=true]
int i; /*für die äußere Schleife*/
\end{lstlisting}
```

3-4-41

The package offers many more keys to influence the presentation. For instance, you can escape to L<sup>A</sup>T<sub>E</sub>X for special formatting tricks, display tab or form-feed characters, index certain identifiers, or interface to `hyperref` so that clicking on some identifier will jump to the previous occurrence. Some of the features are still considered experimental and you have to request them using an optional argument during package loading. These are all documented in great detail in the manual (roughly 50 pages) accompanying the package.

As a final example of the kind of treasures you can find in that manual, look at the following example. It shows code typesetting as known from Donald Knuth's literate programming conventions.

```
\usepackage{listings}
\lstset{literate={:=}{{\$\gets\$}}1
{<=}{{\$\leq\$}}1 {>}{{\$\geq\$}}1 {<>}{{\$\neq\$}}1
\begin{lstlisting}[gobble=2]
var i:integer;
if (i<0) i ← 1;
if (i>0) i ← 0;
if (i≠0) i ← 0;
\end{lstlisting}
```

3-4-42

## 3.5 Lines and columns

In the last part of this chapter we present a few packages that help in manipulating the text stream in its entirety. The first package deals with attaching line numbers to paragraphs, supporting automatic references to them. This can be useful in critical editions and other scholarly works.

The second package deals with the problem of presenting two text streams side by side—for example, some original and its translation. We will show how both packages can be combined in standard cases.

The third package deals with layouts having multiple columns. It allows switching between different numbers of columns on the same page and supports balancing textual data. Standard L<sup>A</sup>T<sub>E</sub>X already offers the possibility of typesetting text in one- or two-column mode, but one- and two-column output cannot be mixed on the same page.

We conclude by introducing a package that allows you to mark the modifications in your source with vertical bars in the margin.

### 3.5.1 lineno—Numbering lines of text

In certain applications it is useful or even necessary to number the lines of paragraphs to be able to refer to them. As T<sub>E</sub>X optimizes the line breaking over the whole paragraph, it is ill equipped to provide such a facility, since technically line breaking happens at a very late stage during the processing, just before the final pages are constructed. At that point macro processing, which could add the right line number or handle automatic references, has already taken place. Hence, the only way to achieve line numbering is by deconstructing the completed page line by line in the “output routine” (i.e., the part of L<sup>A</sup>T<sub>E</sub>X, that normally breaks the paragraph galley into pages and adds running headers and footers) and attaching the appropriate line numbers at that stage.

This approach was taken by Stephan Böttcher in his `lineno` package. Although one would expect such an undertaking to work only in a restricted environment, his package is surprisingly robust and works seamlessly with many other packages—even those that modify the L<sup>A</sup>T<sub>E</sub>X output routine, such as `ftnright`, `multicol`, and `wrapfig`. It also supports layouts produced with the `twocolumn` option of the standard L<sup>A</sup>T<sub>E</sub>X classes.

```
\linenumbers*[start-number]      \nolinenumbers
```

Loading the `lineno` package has no direct effect: to activate line numbering, a `\linenumbers` command must be specified in the preamble or at some point in the document. The command `\nolinenumbers` deactivates line numbering again. Line numbering works on a per-paragraph basis. Thus, when L<sup>A</sup>T<sub>E</sub>X sees the end of a paragraph, it checks whether line numbering is currently requested and, if so, attaches numbers to *all* lines of that paragraph. It is therefore best to put these commands between paragraphs rather than within them.

The `\linenumbers` command can take an optional argument that denotes the number to use for the first line. If used without such an argument, it continues from where it stopped numbering previously. You can also use a star form, which

is a shorthand for `\linenumbers[1]`.

<p>No line numbers here. Some text to experiment with line numbering.</p> <p>1 But here we get line numbers. Some text to experiment with line numbering.</p> <p>2 And here too. Some text to experiment with line numbering.</p> <p>-10 Restart with a negative number. Some text to experiment with line numbering.</p>	<pre>\usepackage{lineno} \newcommand\para{ Some text to experiment with line numbering.\par}  No line numbers here.\para \linenumbers But here we get line numbers.\para And here too.\para \linenumbers[-10] Restart with a negative number.\para</pre>
<p><b>3-5-1</b></p>	

Rather than starting or stopping line numbering with the above commands, you can use the environment `linenumbers` to define the region that should get line numbers. This environment will automatically issue a `\par` command at the end to terminate the current paragraph. If line numbers are needed only for short passages, the environment `form` (or one of the special environments `numquote` and `numquotation` described later) is preferable.

As the production of line numbers involves the output routine, numbering will take place only for paragraphs being built and put on the “main vertical list” but not for those built inside boxes (e.g., not inside a `\marginpar` or within the body of a float). However, the package offers some limited support for numbering lines in such places via the `\internallinenumbers` command. Restrictions are that the baselines within such paragraphs need to be a fixed distance apart (otherwise, the numbers will not get positioned correctly) and that you may have to end such paragraphs with explicit `\par` commands. The `\internallinenumbers` command accepts a star and an optional argument just as `\linenumbers` does. However, the starred form not only ensures that line numbering is (re)started with 1, but also that the line numbers do not affect line numbering in the main vertical list; compare the results in the two `\marginpars` below.

<p>1 Some text to experiment with line numbering.</p> <p>6 Some text to experiment with line numbering.</p>	<p>1 Some text on the main vertical list! Some text to experiment with line numbering.</p> <p>4 Some text to experiment with line numbering.</p> <p>9 In this paragraph we use a second marginal note affecting the line numbers this time. Some text to experiment with line numbering.</p>	<pre>\usepackage{lineno} % \para defined as before \linenumbers Some text on the main vertical list! \marginpar{\footnotesize \internallinenumbers* \para} \para \para In this paragraph we use a second marginal note affecting the \marginpar{\footnotesize \internallinenumbers \para} line numbers this time.\para</pre>
<p><b>3-5-2</b></p>		

The line numbers in the second `\marginpar` continue the numbering on the main vertical list (the last line of the preceding paragraph was 5) and the third

*Numbering boxed paragraphs*

paragraph then continues with line number 9. Such `\marginpar` commands are processed before the paragraph containing them is broken into lines, which explains the ordering of the numbers.

*Handling display math* As `lineno` needs `\par` to attach line numbers when the output routine is invoked, a *TeX*nical problem arises when certain display math constructs are used: the partial paragraph above such a display is broken into lines by *TeX* without issuing a `\par`. As a consequence, without further help such a partial paragraph will not get any line numbers attached. The package's solution, as illustrated in the next example, is to offer the environment `linenomath`, which, if it surrounds such a display, will take care of the line numbering problem. It also has a starred form that also numbers the display lines.

No line number before the display:

$$x \neq y$$

- 1 Some text to experiment with line numbering.
- 2 But line numbers in this case:

$$x \neq y$$

- 3 Some text to experiment with line numbering.

```
\usepackage{lineno} \linenumbers
\newcommand\sample{ Some text to
                  experiment with line numbering.}
No line number before the display:
\[\ x \neq y \] \sample \par
But line numbers in this case:
\begin{linenomath}
  \[\ x \neq y \]
\end{linenomath}
\sample\par
```

3-5-3

If there are many such displays the need for surrounding each of them with a `linenomath` environment is cumbersome. For this reason the package offers the option `displaymath`, which redefines the basic *LATEX* math display environments so that they internally use `linenomath` environments. The option `mathlines` will make `linenomath` behave like its starred form so that the displayed mathematical formulas get line numbers as well.

- 1 Some text to experiment with line numbering.
- 2  $x \neq y$
- 3 Some text to experiment with line numbering.
- 4 Some text to experiment with line numbering.
- 5  $x \neq y$
- 6 Some text to experiment with line numbering.

```
\usepackage[displaymath,mathlines]
           {lineno}
\linenumbers
% \sample as defined before
\sample \[\ x \neq y \] \sample\par
\sample
\begin{displaymath}
  x \neq y
\end{displaymath}
\sample
```

3-5-4

*Cross-references to line numbers* To reference line numbers put a `\linelabel` into the line and then refer to it via `\ref` or `\pageref`, just as with other references defined using `\label`. The exception is that `\linelabel` can only be used on the main vertical list and should only be used within paragraphs that actually carry numbers. If it is used elsewhere,

you get either a bogus reference (if the current line does not have a line number) or an error message (in places where `\linelabel` is not allowed).

1 Some text to experiment with line numbering.  
 2 In the text on lines 2, 3, up to and including line 5 we see references to individual  
 3 lines ...  
 4  
 5  
 6  
 7  
 8  
 9

3-5-5

```
\usepackage{lineno}
\linenumbers
% \sample as defined before

\sample\linelabel{first}\sample\sample
\sample\linelabel{second}\sample

In the text on lines~\ref{first},
\lineref[1]{first}, up to and including
line~\ref{second} we see references to
individual lines \ldots
```

It is also possible to refer to a line that carries no `\linelabel`, by using the `\lineref` command with an optional argument specifying the offset. This ability can be useful if you need to refer to a line that cannot be easily labeled, such as a math display, or if you wish to refer to a sequence of lines, as in the previous example.

There are several ways to customize the visual appearance of line numbers. Specifying the option `modulo` means that line numbers will only appear on some lines (default is every fifth). This effect can also be achieved by using the command `\modulolinenumbers`. Calling this command with an optional argument attaches numbers to lines that are multiples of the specified number (in particular, a value of 1 corresponds to normal numbering). Neither command nor option initiates line numbering mode, for that a `\linenumbers` command is still necessary.

*Labeling only some lines*

1 Some text to experiment with line numbering. Some text to experiment with line numbering. Some text to experiment with line numbering.  
 2 And now a paragraph with numbers on  
 3 every second line. Some text to experiment  
 4 with line numbering. Some text to experiment  
 5 with line numbering. Some text to experiment  
 6 with line numbering.  
 7  
 8  
 9

3-5-6

```
\usepackage{lineno}
\linenumbers
% \sample defined as before

\sample \sample \sample \par
\modulolinenumbers[2]
And now a paragraph with numbers on every
second line.\sample \sample \sample \par
```

The font for line numbers is controlled by the hook `\linenumberfont`. Its default definition is to use tiny sans serif digits. The numbers are put flush right in a box of width `\linenumberwidth`. This box is separated from the line by the value stored in `\linenumbersep`. To set the number flush left you have to dig deeper, but even for this case you will find hooks like `\makeLineNumberRight` in the package. Although changing the settings in the middle of a document is usually not a

good idea, it was done in the next example for demonstration purposes.

The option “right” changes the line number position. Some text to experiment with line numbering. Some text to experiment with line numbering.

Now we use a different font and a bigger separation. Some text to experiment with line numbering. Some text to experiment with line numbering.

```
\usepackage[right]{lineno}
\linenumbers
1 % \sample defined as before
2 The option ‘‘right’’ changes the line
3 number position. \sample \sample \par
4 \renewcommand\linenumberfont
5 {\normalfont\footnotesize\ttfamily}
6 \setlength{\linenumbersep}{20pt}
7 Now we use a different font and a bigger
8 separation. \sample \sample \par
```

3-5-7

For special applications the package offers two environments that provide line numbers automatically: `numquote` and `numquotation`. They are like their L<sup>A</sup>T<sub>E</sub>X cousins `quote` and `quotation`, except that their lines are numbered. They accept an optional argument denoting the line number with which to start (if the argument is omitted, they restart with 1) and they have starred forms that will suppress resetting the line numbers.

The main difference from their L<sup>A</sup>T<sub>E</sub>X counterparts (when used together with the `\linenumbers` command) is the positioning of the numbers, which are indented inward. Thus, their intended use is for cases when only the quoted text should receive line numbers that can be referenced separately.

```
1 Some text to experiment with line
2 numbering.
3 Some text to experiment with line number-
4 ing. Some text to experiment with line num-
5 bering.
1 Some text to experiment with line
2 numbering.
3 Some more text.
```

```
\usepackage{lineno}
\linenumbers
% \sample defined as before
\begin{quote}
\sample
\end{quote}
\sample \sample
\begin{numquote}
\sample
\end{numquote}
Some more text.
```

3-5-8

*Providing your own extensions* Using the machinery provided by the package material, it is fairly easy to develop your own environments that attach special items to each line. The main macro to customize is `\makeLineNumber`, which gets executed inside a box of zero width at the left edge of each line (when line numbering mode is turned on). The net effect of your code should take up no space, so it is best to operate with `\llap` or `\rlap`. Apart from that you can use basically anything. You should only remember that the material is processed and attached after the paragraph has been broken into lines and normal macro-processing has finished, so, you should not expect it to interact with data in mid-paragraph. You can produce the current line number with the `\LineNumber` command, which will supply the number or nothing, depending on whether line numbering mode is on.

The following example shows the definition and use of two new environments that (albeit somewhat crudely, as they do not care about setting fonts and the like) demonstrate some of the possibilities. Note that even though the second environment does not print any line numbers, the lines are internally counted, so that line numbering resumes afterwards with the correct value.

```

1→ Some text to experiment
2→ with line numbering.
    Some text to experiment ←
        with line numbering. Some text ←
            to experiment with line number- ←
                ing.                                ←
7→ Some text to experiment
8→ with line numbering. Some text
9→ to experiment with line number-
10→ ing.

```

```

\usepackage{lineno} \linenumbers
% \sample defined as before
\newenvironment{numarrows}
  {\renewcommand{\makeLineNumber}
   {\llap{\LineNumber$\rightarrow$}}}
  {\par}
\newenvironment{arrows}{\renewcommand{\makeLineNumber}
  {\rlap{\hspace{\textwidth}\leftarrow}}\par}
\begin{numarrows} \sample \end{numarrows}
\begin{arrows} \sample \sample \end{arrows}
\sample
\begin{numarrows} \sample \end{numarrows}

```

The appearance and behavior of the line numbers can be further controlled by a set of options or, alternatively, by a set of commands equivalent to the options (see the package documentation for details on the command forms). With the options `left` (default) and `right`, you specify in which margin the line numbers should appear. Using the option `switch` or `switch*`, you get them in the outer and inner margins, respectively.

At least two  $\text{\LaTeX}$  runs of the document are required before the line numbers will appear in the appropriate place. Unfortunately, there is no warning about the need to rerun the document, so you have to watch out for this issue yourself.

You can also request that numbers restart on each page by specifying the option `pagewise`. This option needs to come last.

### 3.5.2 parallel—Two text streams aligned

Sometimes it is necessary to typeset something in parallel columns, such as when presenting some text and its translation. Parallel in this context means that at certain synchronization points the two text streams are vertically (re)aligned. This type of layout is normally not supported by  $\text{\LaTeX}$  (which by default only works with a single text stream), but it can be achieved by using Matthias Eckermann's `parallel` package.

This package provides the `Parallel` environment, which surrounds the material to be typeset in parallel. It takes two mandatory arguments: the widths of the left and right columns. Their sum should be less than `\textwidth`; otherwise, the text in the two columns will touch or even overlap. To ease usage, one or both arguments can be left empty, in which case the appropriate width for the column(s) will be calculated automatically, using the current value of `\ParallelUserMidSkip` as the column separation. To mark up the left and the right text streams, you use

*\verb is allowed* \ParallelText and \ParallelRText, respectively. Although both commands expect the text as an argument, it is nevertheless possible to use \verb or a verbatim environment inside, as the following example shows.

This is text in  
the English lan-  
guage explaining  
the command \foo.  
Dies ist Text in  
deutscher Sprache,  
der das Kommando  
\foo erläutert.

```
\usepackage{parallel}
\begin{Parallel}{}{}
\ParallelText{This is text in the English
language explaining the command \verb=\foo=.}
\ParallelRText{Dies ist Text in deutscher Sprache,
der das Kommando \verb=\foo= erläutert.}
\end{Parallel}
```

3-5-10

To align certain lines of text you split the two text streams at appropriate points by using pairs of \ParallelText and \ParallelRText commands and separating each pair with \ParallelPar. If you forget one of the \ParallelPar commands, some of your text will get lost without warning. Moreover, as its name suggests, the \ParallelPar command introduces a paragraph break, so that alignment is possible only at paragraph boundaries. Additional paragraph breaks inside the argument of a \Parallel..Text command are also possible but in that case no alignment is attempted.

In the next example, displaying a few “direct” translations of computer jargon into German (taken from [54] with kind permission by Eichborn Verlag), we define a shorthand command \LR to make it easier to input the text. If such a shorthand is used, \verb can no longer be used in the argument. Thus, if you need \verb, use the package commands directly. We also use the lineno package since line numbers can be useful when talking about a text and its translation.

I just go online 2 and download an update.	Ich geh mal eben auf den Strich und lade mir ein Auffrisch herunter.
4	
6 This laptop is missing	Dieser Schoßspitze
8 several interfaces.	fehlt so manches Zwi- schengesicht.
10 Microsoft Office 12 on floppy disks.	Kleinweich Büro auf Schlabber- scheiben.

```
\usepackage{parallel,lineno}
\linenumbers \modulolinenumbers[2]
\setlength\linenumbersep{1pt}
\newcommand\LR[2]{\ParallelText{\#1}%
\ParallelRText{\#2}\ParallelPar}
\begin{Parallel}{.45\linewidth}{}%
\raggedright \setlength\leftskip{10pt}
\setlength\parindent{-\leftskip}
\LR{I just go online and download an update.}{Ich
geh mal eben auf den Strich und lade mir ein
Auffrisch herunter.} \LR{This laptop is missing
several interfaces.}{Dieser Schoßspitze
fehlt so manches Zwischen-sicht.}
\LR{Microsoft Office on floppy disks.}{Kleinweich
Büro auf Schlabberscheiben.}
\end{Parallel}
```

3-5-11

As you can see, it is possible to adjust paragraph parameters within the scope of the Parallel environment. The negative \parindent cancels the pos-

itive `\leftskip` so that each paragraph starts flush left but following lines are indented by `\leftskip` (and both must be changed *after* calling `\raggedright`, as the latter also sets these registers).

The `Parallel` environment works by aligning line by line, which has a surprising consequence when one block contains unusually large objects, such as a display. Thus, the method is suitable only for normal text lines.

This is text that contains:

$$\sum_{n=1}^x 2a_n$$

And here is the explanation showing some surprising effect.

```
\usepackage{parallel}
\begin{Parallel}{}{}
  \ParallelLText{This is text that contains:
    \[ \sum_{n=1}^x a_n \]}
  \ParallelRText{And here is the explanation
    showing some surprising effect.}
\end{Parallel}
```

3-5-12

Footnotes within the parallel text are not placed at the bottom of the current page, but rather are typeset directly after the end of the current `Parallel` environment and separated from it by the result of executing `\ParallelAtEnd`, which is a command defined to do nothing. You can, however, redefine it to place something between footnotes and preceding text. If the redefinition should apply only to a single `Parallel` environment, place it within the scope of the environment.

*Footnotes in parallel text*

The presentation of the footnotes is controlled by four package options: `OldStyleNums` sets footnote numbers using old-style numerals, `RaiseNums` generates raised footnote numbers, and `ItalicNums` produces italic numbers. If none of these options is given, then Arabic numerals at the baseline position are used. The options affect only the numbers in front of the footnote text; the markers within the parallel text are always raised Arabic numerals. The fourth option, `SeparatedFootnotes`, can be combined with one of the three other options and indicates that footnotes in each column should be independently numbered. The numbers from the right column are then postfixed with `\ParallelDot`, which by default produces a centered dot. In the next example its definition is slightly modified so that the dot itself does not take up any space.

This is text in the English language<sup>1</sup> explaining the command `\foo`.

Dies ist Text<sup>1</sup> in deutscher Sprache<sup>2</sup>, der das Kommando `\foo` erläutert.

---

<sup>1</sup> We hope!  
<sup>2</sup> Ein Satz.  
<sup>2</sup> Schlechter Stil!

```
\usepackage[OldStyleNums, SeparatedFootnotes]{parallel}
\renewcommand\ParallelAtEnd{\vspace{7pt}\footnoterule}
\renewcommand\ParallelDot
  {\makebox[0pt][l]{\textperiodcentered}}
\begin{Parallel}[v]{}
  \raggedright
  \ParallelLText{Dies ist Text\footnote{Ein Satz.} in
    deutscher Sprache\footnote{Schlechter Stil!}, der
    das Kommando \verb=\foo= erl\"autert.}
\end{Parallel}
```

3-5-13

The `\Parallel` environment can sport an optional argument before the mandatory ones, whose value can be `c` (make two columns—the default), `v` (separate columns with a vertical rule as shown in the previous example), or `p` (put left text on left-hand pages and right text on right-hand pages). If the “page” variant is chosen it is possible that you get empty pages. For example, if you are on a verso page the environment has to skip to the next recto page in order to display the texts on facing pages.

### 3.5.3 `multicol`—A flexible way to handle multiple columns

With standard L<sup>A</sup>T<sub>E</sub>X it is possible to produce documents with one or two columns (using the class option `twocolumn`). However, it is impossible to produce only parts of a page in two-column format as the commands `\twocolumn` and `\onecolumn` always start a fresh page. Additionally, the columns are never balanced, which sometimes results in a slightly weird distribution of the material.

The `multicol` package<sup>1</sup> by Frank Mittelbach solves these problems by defining an environment, `multicols`, with the following properties:

- Support is provided for 2–10 columns, which can run for several pages.
- When the environment ends, the columns on the last page are balanced so that they are all of nearly equal length.
- The environment can be used inside other environments, such as `figure` or `minipage`, where it will produce a box containing the text distributed into the requested number of columns. Thus, you no longer need to hand-format your layout in such cases.
- Between individual columns, vertical rules of user-defined widths can be inserted.
- The formatting can be customized globally or for individual environments.

```
\begin{multicols}{columns} [preface] [skip]
```

Normally, you can start the environment simply by specifying the number of desired columns. By default paragraphs will be justified, but with narrow measures—as in the examples—they would be better set `unjustified` as we show later on.

Here is some text to be distributed over several columns. If the columns are very narrow try type-

```
\usepackage{multicol}
\begin{multicols}{3}
Here is some text to be distributed over
several columns. If the columns are very
narrow try typesetting ragged right.
\end{multicols}
```

3-5-14

<sup>1</sup>Although the `multicol` package is distributed under LPPL (L<sup>A</sup>T<sub>E</sub>X Project Public License) [111], for historical reasons its copyright contains an additional “moral obligation” clause that asks commercial users to consider paying a license fee to the author or the L<sup>A</sup>T<sub>E</sub>X3 fund for their use of the package. For details see the head of the package file itself.

```
\premulticols 50.0pt      \postmulticols 20.0pt
\columnsep    10.0pt       \columnseprule 0.0pt
\multicolssep 12.0pt plus 4.0pt minus 3.0pt
```

Table 3.8: Length parameters used by `multicols`

You may be interested in prefixing the multicolumn text with a bit of single-column material. This can be achieved by using the optional *preface* argument. L<sup>A</sup>T<sub>E</sub>X will then try to keep the text from this argument and the start of the multi-column text on the same page.

## Some useful advice

Here is some text to be distributed over several columns. If the

```
\usepackage{multicol}
\begin{multicols}{2}
[\section*{Some useful advice}]
Here is some text to be distributed over
several columns. If the columns are very
narrow try typesetting ragged right.
\end{multicols}
```

The `multicols` environment starts a new page if there is not enough free space left on the current page. The amount of free space is controlled by a global parameter. However, when using the optional argument the default setting for this parameter may be too small. In this case you can either change the *global* default (see below) or adjust the value for the *current* environment by using a second optional *skip* argument as follows:

```
\begin{multicols}{3}[\section*{Index}] [7cm]
Text Text Text Text ...
\end{multicols}
```

This would start a new page if less than 7 cm free vertical space was available.

The `multicols` environment balances the columns on the last page (it was originally developed for exactly this purpose). If this effect is not desired you can use the `multicols*` variant instead. Of course, this environment works only in the main vertical galley, since inside a box one has to balance the columns to determine a column height.

*Preventing  
balancing*

The `multicols` environment recognizes several formatting parameters. Their meanings are described in the following sections. The default values can be found in Table 3.8 (dimensions) and Table 3.9 (counters). If not stated otherwise, all changes to the parameters have to be placed before the start of the environment to which they should apply.

The `multicols` environment first checks whether the amount of free space left on the page is at least equal to `\premulticols` or to the value of the second optional argument, when specified. If the requested space is not available, a

*The required free  
space*

\multicolpretolerance	-1	\multicoltolerance	9999
columnbadness	10000	finalcolumnbadness	9999
collectmore	0	unbalance	0
tracingmulticols	0		

Table 3.9: Counters used by `multicols`

\newpage is issued. A new page is also started at the end of the environment if the remaining space is less than \postmulticols. Before and after the environment, a vertical space of length \multicolsep is placed.

*Column width and separation* The column width inside the `multicols` environment will automatically be calculated based on the number of requested columns and the current value of \linewidth. It will then be stored in \columnwidth. Between columns a space of \columnsep is left.

*Adding vertical lines* Between any two columns, a rule of width \columnseprule is placed. If this parameter is set to 0pt (the default), the rule is suppressed. If you choose a rule width larger than the column separation, the rule will overprint the column text.

```
\usepackage{multicol,ragged2e}
\setlength\columnseprule{0.4pt}
\addtolength\columnsep{2pt}
\begin{multicols}{3}
\RaggedRight
Here is some text to be distributed over
several columns. In this example ragged-right
typesetting is used.
\end{multicols}
```

Here is some text to be distributed over several columns. In this example ragged-right typesetting is used.

3-5-16

### Column formatting

By default (the \flushcolumns setting), the `multicols` environment tries to typeset all columns with the same length by stretching the available vertical space inside the columns. If you specify \raggedcolumns the surplus space will instead be placed at the bottom of each column.

Paragraphs are formatted using the default parameter settings (as described in Sections 3.1.11 and 3.1.12) with the exception of \pretolerance and \tolerance, for which the current values of \multicolpretolerance and \multicoltolerance are used, respectively. The defaults are -1 and 9999, so that the paragraph-breaking trial without hyphenation is skipped and relatively bad paragraphs are allowed (accounting for the fact that the columns are typically very narrow). If the columns are wide enough, you might wish to change these defaults to something more restrictive, such as

```
\multicoltolerance=3000
```

Note the somewhat uncommon assignment form: `\multicoltolerance` is an internal TeX counter and is controlled in exactly the same way as `\tolerance`.

### Balancing control

At the end of the `multicols` environment, remaining text will be balanced to produce columns of roughly equal length. If you wish to place more text in the left columns you can advance the counter `unbalance`. This counter determines the number of additional lines in the columns in comparison to the number that the balancing routine has calculated. It will automatically be restored to zero after the environment has finished. To demonstrate the effect, the next example uses the text from Example 3-5-16 on the facing page but requests one extra line.

1 Here is some text to be distributed over several columns. In this example ragged-right typesetting is used.

```
\usepackage{multicol,ragged2e}
\addtolength{\columnsep}{2pt}
\begin{multicols}{3}
\RaggedRight
\setcounter{unbalance}{1}
\end{multicols}
```

Here is some text to be distributed over several columns. In this example ragged-right typesetting is used.

3-5-17

Column balancing is further controlled by the two counters `columnbadness` and `finalcolumnbadness`. Whenever TeX is constructing boxes (such as a column) it will compute a badness value expressing the quality of the box—that is, the amount of excess white space. A zero value is optimal, and a value of 10000 is infinitely bad in TeX's eyes.<sup>2</sup> While balancing, the algorithm compares the badness of possible solutions and, if any column except the last one has a badness higher than `columnbadness`, the solution is ignored. When the algorithm finally finds a solution, it looks at the badness in the last column. If it is larger than `finalcolumnbadness`, it will typeset this column with the excess space placed at the bottom, allowing it to come out short.

### Collecting material

To be able to properly balance columns the `multicols` environment needs to collect enough material to fill the remaining part of the page. Only then does it cut the collected material into individual columns. It tries to do so by assuming that not more than the equivalent of one line of text per column vanishes into the margin due to breaking at vertical spaces. In some situations this assumption is incorrect and it becomes necessary to collect more or less material. In such a case

<sup>1</sup>Very bad for reading but too good to fix: this problem of a break-stack with "the" four times in a row will not be detected by TeX's paragraph algorithm—only a complete paragraph rewrite would resolve it.

<sup>2</sup>For an overfull box the badness value is set to 100000 by TeX, to mark this special case.

you can adjust the default setting for the counter `collectmore`. Changing this counter by one means collecting material for one more (or less) `\baselineskip`.

There are, in fact, reasons why you may want to reduce that collection. If your document contains many footnotes and a lot of surplus material is collected, there is a higher chance that the unused part will contain footnotes, which could come out on the wrong page. The smallest sensible value for the counter is the negative number of columns used. With this value `multicols` will collect exactly the right amount of material to fill all columns as long as no space gets lost at a column break. However, if spaces are discarded in this set up, they will show up as empty space in the last column.

### Tracing the algorithm

You can trace the behavior of the `multicol` package by loading it with one of the following options. The default, `errorshow`, displays only real errors. With `infoshow`, `multicol` becomes more talkative and you will get basic processing information such as

```
Package multicol: Column spec: 185.0pt = indent + columns + sep =
(multicol)      0.0pt + 3 x 55.0pt + 2 x 10.0pt on input line 32.
```

which is the calculated column width.

With `balancingshow`, you get additional information on the various trials made by `multicols` when determining the optimal column height for balancing, including the resulting badness of the columns, reasons why a trial was rejected, and so on.

Using `markshow` will additionally show which marks for the running header or footer are generated on each page. Instead of using the options you can (temporarily) set the counter `tracingmulticols` to a positive value (higher values give more tracing information).

### Manually breaking columns

Sometimes it is necessary to overrule the column-breaking algorithm. We have already seen how the `unbalance` counter is used to influence the balancing phase. But on some occasions one wishes to explicitly end a column after a certain line. In standard L<sup>A</sup>T<sub>E</sub>X this can be achieved with a `\pagebreak` command, but this approach does not work within a `multicols` environment because it will end the collection phase of `multicols` and thus end *all* columns on the page. As an alternative the command `\columnbreak` is provided. If used within a paragraph it marks the end of the current line as the desired breakpoint. If used between paragraphs it forces the next paragraph into the next column (or page) as shown in the following example. If `\flushcolumns` is in force, the material in the column is vertically stretched (if possible) to fill the full column height. If this effect is not desired one can prepend a `\vfill` command to fill the bottom of the column with white space.

Here is some text to be distributed over several columns.

3-5-18

With the help of the `\columnbreak` command this paragraph was forced into the second column.

```
\usepackage{multicol,ragged2e}
\begin{multicols}{2} \RaggedRight
Here is some text to be distributed over several
columns. \par \vfill\columnbreak
With the help of the \verb=\columnbreak= command
this paragraph was forced into the second column.
\end{multicols}
```

### Floats and footnotes in multicol

Floats (e.g., figures and tables) are only partially supported within `multicols`. You can use starred forms of the float environments, thereby requesting floats that span all columns. Column floats and `\marginpars`, however, are not supported.

Footnotes are typeset (full width) on the bottom of the page, and not under individual columns (a concession to the fact that varying column widths are supported on a single page).

Under certain circumstances a footnote reference and its text may fall on subsequent pages. If this is a possibility, `multicols` produces a warning. In that case, you should check the page in question. If the footnote reference and footnote text really are on different pages, you will have to resolve the problem locally by issuing a `\pagebreak` command in a strategic place. The reason for this behavior is that `multicols` has to look ahead to assemble material and may not be able to use all material gathered later on. The amount of looking ahead is controlled by the `collectmore` counter.

### 3.5.4 changebar—Adding revision bars to documents

When a document is being developed it is sometimes necessary to (visually) indicate the changes in the text. A customary way of doing that is by adding bars in the margin, known as “changebars”. Support for this functionality is offered by the `changebar` package, originally developed by Michael Fine and Neil Winton, and now supported by Johannes Braams. This package works with most PostScript drivers, but in particular `dvi`, which is the default driver when the package is loaded. Other drivers can be selected by using the package option mechanism. Supported options are `dvitoln03`, `dvitops`, `dvi`, `emtex`, `textures`, and `vtx`.

*Supported printer  
drivers*

```
\begin{changebar}[barwidth] \cbstart[barwidth] ... \cbend
```

When you add text to your document and want to signal this fact, you should surround it with the `changebar` environment. Doing so ensures that L<sup>A</sup>T<sub>E</sub>X will warn you when you forget to mark the end of a change. This environment can be (properly) nested within other environments. However, if your changes start within one L<sup>A</sup>T<sub>E</sub>X environment and end inside another, the environment form cannot be used as this would result in improperly nested environments. Therefore, the package also provides the commands `\cbstart` and `\cbend`. These should be used with

care, because there is no check that they are properly balanced. Spaces after them might get ignored.

If you want to give a single bar a different width you may use the optional argument and specify the width as a normal L<sup>A</sup>T<sub>E</sub>X length.

`\cbdelete[barwidth]`

Text that has been removed can be indicated by inserting the `\cbdelete` command. Again, the width of the bar can be changed.

```
\usepackage{changebar}
\cbstart
This is the text in the first paragraph.
This is the text in the first paragraph.\cbend
```

```
This is the text in the second paragraph.
\cbdelete
This is the text in the second paragraph.
```

```
\setcounter{changebargrey}{35}
\begin{changebar}[4pt]
This is paragraph three. \par
This is paragraph four.
\end{changebar}
```

This is the text in the first paragraph. This is the text in the first paragraph.

This is the text in the second paragraph. This is the text in the second paragraph.

This is paragraph three.

This is paragraph four.

3-5-19

`\nochangebars`

When your document has reached the final stage you can remove the effect of using the `changebar` package by inserting the command `\nochangebars` in the preamble of the document.

### Customizations

*Changing the width* If you want to change the width of *all* changebars you can do so by changing the value of `\changebarwidth` via the command `\setlength`. The same can be done for the deletion bars by changing the value of `\deletebarwidth`.

*Positioning changebars* By default, the changebars will show up in the “inner margin”, but this can be changed by using one of the following options: `outerbars`, `innerbars`, `leftbars`, or `rightbars`.

The distance between the text and the bars is controlled by `\changebarssep`. It can be changed only in the preamble of the document.

*Coloring changebars* The color of the changebars can be changed by the user as well. By default, the option `grey` is selected so the changebars are grey (grey level 65%). The drivers `dviolt03` and `emtex` are exceptions that will produce black changebars.

The “blackness” of the bars can be controlled with the help of the L<sup>A</sup>T<sub>E</sub>X counter `changebargrey`. A command like `\setcounter{changebargrey}{85}` changes

that value. The value of the counter is a percentage, where 0 yields black bars, and 100 yields white bars.

The option `color` makes it possible to use colored changebars. It internally loads `dvinames`, so you can use a name when selecting a color.

```
\cbbcolor{name}
```

The color to use when printing changebars is selected with the command `\cbbcolor`, which accepts the same arguments as the `\color` command from the `color` package [57, pp.317-326].

```
\usepackage[rightbars,color]{changebar}
\cbbcolor{blue}
\setlength\changebarssep{10pt}
\cbstart
This is the text in the first paragraph.
This is the text in the first paragraph.\cbend
```

This is the text in the second paragraph.

\cbdelete

This is the text in the second paragraph.

\begin{changebar}

This is paragraph three. \par

This is paragraph four.

\end{changebar}

This is the text in the first paragraph.  
This is the text in the first paragraph.

This is the text in the second paragraph.  
This is the text in the second paragraph.

This is paragraph three.

This is paragraph four.

3-5-20

You can trace the behavior of the `changebar` package by loading it with one of the following options. The default, `traceoff`, displays the normal information TEX always shows. The option `traceon` informs you about the beginning and end points of changebars being defined. The `additional` option `tracestacks` adds information about the usage of the internal stacks.

*Tracing the  
algorithm*



## C H A P T E R 4

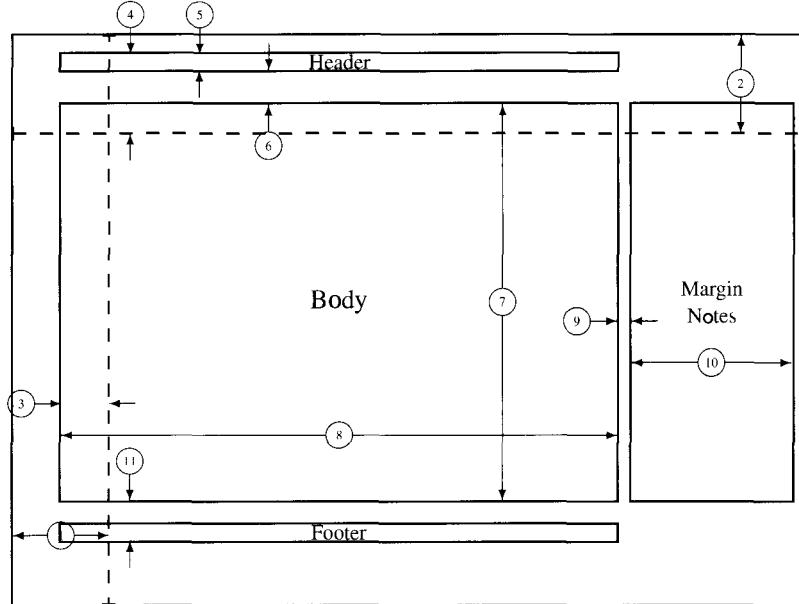
# The Layout of the Page

In this chapter we will see how to specify different page layouts. Often a single document requires several different page layouts. For instance, the layout of the first page of a chapter, which carries the chapter title, is generally different from that of the other pages in that chapter.

We first introduce  $\text{\LaTeX}$ 's dimensional parameters that influence the page layout and describe ways to change them and visualize their values. This is followed by an in-depth discussion of the packages `typearea` and `geometry`, both of which provide sophisticated ways to implement page layout specifications. The third section deals with the  $\text{\LaTeX}$  concepts used to provide data for running headers and footers. This is followed by a section that explains how to format such elements, including many examples deploying the `fancyhdr` package and others. The fifth section then introduces commands that help in situations when the text does not fit into the layout and manual intervention is required. The chapter concludes with a brief look at two generic classes that go a long way toward providing almost full control over the page layout specification process.

### 4.1 Geometrical dimensions of the layout

The text of a document usually occupies a rectangular area on the paper—the so-called *type area* or *body*. Above the text there might be a *running header* and below it a *running footer*. They can consist of one or more lines containing the page number; information about the current chapter, section, time, and date; and possibly other markers. If they are visually heavy and closely tied to the text, then



- |  |  |
|--|--|
| 1    one inch + \hoffset                                       | 2    one inch + \voffset   |
| 3    \oddsidemargin = -36pt                                    | 4    \topmargin = -58pt  |
| 5    \headheight = 12pt  | 6    \headsep = 25pt   |
| 7    \textheight = 296pt                                       | 8    \textwidth = 418pt  |
| 9    \marginparsep = 11pt                                      | 10   \marginparwidth = 121pt<br>\marginparpush = 5pt (not shown) |
| 11   \footskip = 30pt<br>\hoffset = 0pt<br>\paperwidth = 597pt | \voffset = 0pt<br>\paperheight = 423pt                           |

4-1-1

\paperheight Height of the paper to print on.

\paperwidth Width of the paper to print on.

\textheight Height of the body (without header and footer).

\textwidth Width of the body.

\columnsep Width of space between columns of text in multicolumn mode.

\columnseprule Width of a vertical line separating the two adjacent columns in multicolumn output (default 0pt, i.e., no visible rule).

\columnwidth Width of a single column in multicolumn mode. Calculated by L<sup>A</sup>T<sub>E</sub>X from \textwidth and \columnsep as appropriate.

\linewidth Width of the current text line. Usually equals \columnwidth but might get different values in environments that change the margins.

\evensidemargin For two-sided printing, the extra space added at the left of even-numbered pages.

\oddsidemargin For two-sided printing, the extra space added at the left of odd-numbered pages; otherwise the extra space added at the left of all pages.

\footskip Vertical distance separating the baseline of the last line of text and the baseline of the footer.

\headheight Height of the header.

\headsep Vertical separation between header and body.

\topmargin Extra vertical space added at the top of the header.

\marginparpush Minimal vertical space between two successive marginal notes (not shown in the figure).

\marginparsep Horizontal space between body and marginal notes.

\marginparwidth Width of marginal notes.

Figure 4.1: Page layout parameters and visualization

<code>letterpaper</code>	$8\frac{1}{2} \times 11$	inches
<code>legalpaper</code>	$8\frac{1}{2} \times 14$	inches
<code>executivepaper</code>	$7\frac{1}{4} \times 10\frac{1}{2}$	inches
<code>a4paper</code>	$\approx 8\frac{1}{4} \times 11\frac{3}{4}$	inches
<code>a5paper</code>	$\approx 5\frac{7}{8} \times 8\frac{1}{4}$	inches
<code>b5paper</code>	$\approx 7 \times 9\frac{7}{8}$	inches
		210 × 297 mm
		148 × 210 mm
		176 × 250 mm

Table 4.1: Standard paper size options in L<sup>A</sup>T<sub>E</sub>X

these elements are considered to belong to the type area; this is often the case for running headers, especially when underlined. Otherwise, they are considered to belong to the top or bottom *margins*. This distinction is important when interpreting size specifications.

The fields to the left and the right of the body are also called *margins*. Usually they are left blank, but small pieces of text, such as remarks or annotations—so-called *marginal notes*—can appear there.

In general one talks about the *inner* and the *outer* margins. For two-sided printing, inner refers to the middle margins—that is, the left margin on recto (odd-numbered) pages and the right margin on verso (even-numbered) ones. For one-sided printing, inner always indicates the left margin. In a book spread, odd-numbered pages are those on the right-hand side.

The size, shape, and position of these fields and margins on the output medium (paper or screen) and the contents of the running headers and footers are collectively called a *page layout*.

The standard L<sup>A</sup>T<sub>E</sub>X document classes allow document formatting for recto-verso (*two-sided*) printing. Two-sided layouts can be either asymmetrical or symmetrical (the L<sup>A</sup>T<sub>E</sub>X default). In the latter case the type areas of recto and verso pages are positioned in such a way that they overlap if one holds a sheet to the light. Also, marginal notes are usually swapped between left/right pages.

The dimensional parameters controlling the page layout are described and shown schematically in Figure 4.1 on the facing page.<sup>1</sup> The default values of these parameters depend on the paper size. To ease the adjustments necessary to print on different paper sizes, the L<sup>A</sup>T<sub>E</sub>X class files support a number of options that set those parameters to the physical size of the requested paper as well as adjust the other parameters (e.g., `\textheight`) that depend on them.

Table 4.1 shows the paper size options known to standard L<sup>A</sup>T<sub>E</sub>X classes together with the corresponding page dimensions. Table 4.2 on the following page presents the page layout parameter values for the `letterpaper` paper size option, the default when no explicit option is selected. They are identical for the three standard L<sup>A</sup>T<sub>E</sub>X document classes (`article`, `book`, and `report`). If a different paper size option is selected the values may change. Thus, to print on A4 paper, you can simply specify `\documentclass[a4paper]{article}`.

<sup>1</sup>The graphical presentation was produced with the `layouts` package, described in Section 4.2.1.

Parameter	Two-sided printing			One-sided printing		
	10pt	11pt	12pt	10pt	11pt	12pt
\oddsidemargin	44pt	36pt	21pt	63pt	54pt	39pt
\evensidemargin	82pt	74pt	59pt	63pt	54pt	39pt
\marginparwidth	107pt	100pt	85pt	90pt	83pt	68pt
\marginparsep	11pt	10pt	10pt		<i>ditto</i>	
\marginparpush	5pt	5pt	7pt		<i>ditto</i>	
\topmargin	27pt	27pt	27pt		<i>ditto</i>	
\headheight	12pt	12pt	12pt		<i>ditto</i>	
\headsep	25pt	25pt	25pt		<i>ditto</i>	
\footskip	30pt	30pt	30pt		<i>ditto</i>	
\textheight	43 × \baselineskip	38	36		<i>ditto</i>	
\textwidth	345pt	360pt	390pt		<i>ditto</i>	
\columnsep	10pt	10pt	10pt		<i>ditto</i>	
\columnseprule	0pt	0pt	0pt		<i>ditto</i>	

Table 4.2: Default values for the page layout parameters (`letterpaper`)

Additional or different options may be available for other classes. Nevertheless, there seems to be little point in providing, say, an `a0paper` option for the `book` class that would produce incredibly wide text lines.

Most of the layout parameters in L<sup>A</sup>T<sub>E</sub>X class files are specified in terms of the physical page size. Thus, they automatically change when `\paperwidth` or `\paperheight` is modified via one of the paper size options. Changing these two parameters in the preamble of your document does not have this effect, since by then the values for the other parameters are already calculated.

*One-inch default margins* Standard-conforming dvi drivers place the reference point for T<sub>E</sub>X one inch down and to the right of the upper-left corner of the paper. These one-inch offsets are called *driver margins*. The reference point can be shifted by redefining the lengths `\hoffset` and `\voffset`. By default, their values are zero. In general, the values of these parameters should never be changed. They provide, however, a convenient way to shift the complete page image (body, header, footer, and marginal notes) on the output plane without disturbing the layout. The driver margins are inherited from T<sub>E</sub>X, and are not needed in L<sup>A</sup>T<sub>E</sub>X's parameterization of the page layout. A change to `\topmargin` shifts the complete text vertically, while changes to `\oddsidemargin` and `\evensidemargin` shift it horizontally.

Note that some dvi drivers introduce their own shifts in the placement of the text on paper. To make sure that the reference point is properly positioned,

you can run the test file `testpage.tex` (by Leslie Lamport, with modifications by Stephen Gildea) through L<sup>A</sup>T<sub>E</sub>X and the `dvi` driver in question. The resulting output page will show the position of the reference point with respect to the edges of the paper. For L<sup>A</sup>T<sub>E</sub>X2<sub>E</sub> this file was rewritten by Rainer Schöpf to allow the specification of a paper size option.

## 4.2 Changing the layout

When you want to redefine the value of one or more page layout parameters, the `\setlength` and `\addtolength` commands should be used. It is important to keep in mind that changes to the geometrical page layout parameters should be made only in class or package files and/or in the preamble (i.e., before the `\begin{document}` command). Although changing them in mid-document is not absolutely impossible, it will most likely produce havoc, due to the inner workings of T<sub>E</sub>X, which involve a number of subtle dependencies and timing problems. For example, if you change the `\textwidth` you might find that the running header of the previous page is changed.

Initially, it is advisable to use T<sub>E</sub>X's `\baselineskip` parameter for setting vertical distances. This parameter is the distance between the baselines of two consecutive lines of text set in the “normal” document type size inside a paragraph. The `\baselineskip` parameter may be considered to be the height of one line of text. Therefore, the following setting always means “two lines of text”:

```
\normalsize                                % set normal \baselineskip
\setlength{\headheight}{2\baselineskip}    % Height of heading
```

To guarantee that `\baselineskip` is set properly, first set up the fonts used in the document (if necessary), and then invoke `\normalsize` to select the type size corresponding to the document base size.

Sometimes it is convenient to calculate the page layout parameters according to given typographic rules. For example, the requirement “the text should contain 50 lines” can be expressed using the command given below. It is assumed that the height of all (except one) lines is `\baselineskip` and the height of the top line of the text body is `\topskip` (this is T<sub>E</sub>X's `\baselineskip` length parameter for the first line with a default value of 10pt). Note that the examples in this chapter use the L<sup>A</sup>T<sub>E</sub>X package `calc` (which simplifies the calculational notation) and the extended control structures of L<sup>A</sup>T<sub>E</sub>X2<sub>E</sub> (see Appendix A, Sections A.3.1 and A.3.2).

```
\setlength{\textheight}{\baselineskip*49+\topskip}
```

A requirement like “the height of the body should be 198mm” can be met in a similar way, and the calculation is shown below. First calculate the number of lines that the body of the desired size can contain. To evaluate the number of

 Change  
parameters only  
in the preamble

lines, divide one dimension by another to obtain the integer part. As TeX is unable to perform this kind of operation directly, the dimensions are first assigned to counters. The latter assignment takes place with a high precision because sp units are used internally.

```
\newcounter{tempc} \newcounter{tempcc} % define two temporary counters
\setlength{\textheight} % subtract top line
{198mm-\topskip} % from desired size
\setcounter{tempc}{\textheight} % assign counter 1
\setcounter{tempcc}{\baselineskip} % assign counter 2
\setcounter{tempc}{% % divide counters
  {\value{tempc}}/{\value{tempcc}}}
\setlength{\textheight}{\baselineskip*\value{tempc}+\topskip}
```

The value of the vertical distance, `\topmargin`, can also be customized. As an example, suppose you want to set this margin so that the space above the text body is two times smaller than the space below the text body. The following calculation shows how to determine the needed value in the case of A4 paper (the paper height is 297mm).

```
\setlength{\topmargin}
{(297mm-\textheight)/3 - 1in - \headheight - \headsep}
```

In general, when changing the page layout you should take into account some elementary rules of legibility (see, for example, [150]). Studies of printed material in the English language have shown that a line should not contain more than 10–12 words, which corresponds to not more than 60–70 characters per line.

The number of lines on a page depends on the type size being used. The code below shows one way of calculating a `\textheight` that depends on the document base size. Use the fact that in most document classes the internal L<sup>A</sup>T<sub>E</sub>X command `\@ptsize` holds the number 0, 1, or 2 for the base font size 10pt, 11pt, or 12pt, respectively. This command is set when you select an option such as 11pt.

```
\ifthenelse{\@ptsize = 0}%
  {10 point typeface as base size}
  {\setlength{\textheight}{53\baselineskip}\{}%
\ifthenelse{\@ptsize = 1}%
  {11 point typeface as base size}
  {\setlength{\textheight}{46\baselineskip}\{}%
\ifthenelse{\@ptsize = 2}%
  {12 point typeface as base size}
  {\setlength{\textheight}{42\baselineskip}\{}%
\addtolength{\textheight}{\topskip}
```

Another important parameter is the amount of white space surrounding the text. As printed documents are likely to be bound or stapled, enough white space should be left in the inner margin of the text to allow for this possibility. If

`\oddsidemargin` is fixed, then the calculation of `\evensidemargin` for two-sided printing is based on the following relationship:

```
width_of_paper =
  1in + \oddsidemargin + \textwidth + \evensidemargin + 1in
```

In most classes two-sided printing is turned on by specifying the `twoside` class option, which sets the Boolean register `@twoside` to `true`. Using commands from the `ifthen` package we can set parameters depending on the value of this Boolean register, also taking into account the selected document base size:

```
\ifthenelse{\@ptsize = 0}%
  {10 point typeface as base size}
  {\setlength\textwidth{5in}%
   \setlength\marginparwidth{1in}%
   \ifthenelse{\boolean{@twoside}}%
     {\setlength\oddsidemargin {0.55in}%
      \setlength\evensidemargin{0.75in}%
      \setlength\oddsidemargin {0.55in}%
      \setlength\evensidemargin{0.55in}%
    }{}%
   \ifthenelse{\@ptsize = 1}{...}%
   \ifthenelse{\@ptsize = 2}{...}%
  }
```

Similarly, when a document contains a lot of marginal notes, it is worthwhile changing the layout to increase the margins. As an example, the (obsolete) `a4` package defines a command `\WideMargins`. This macro modifies the geometrical parameters in such a way that the width reserved for marginal notes is set to 1.5 inches by decreasing the width of the text body.

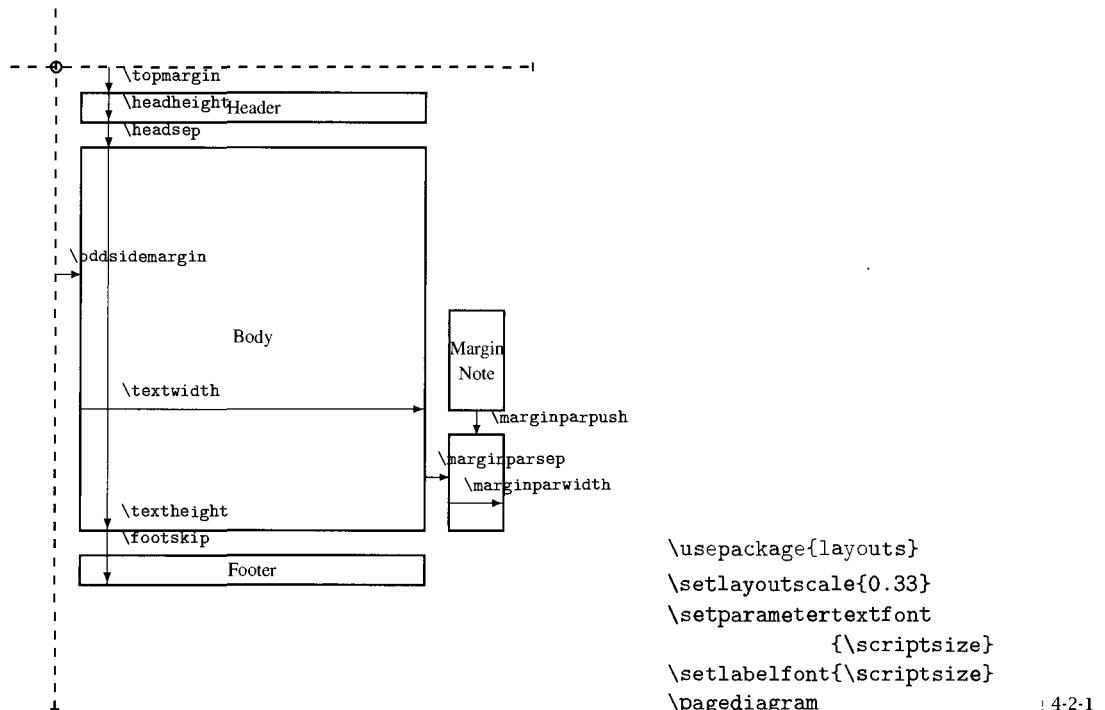
### 4.2.1 layouts—Displaying your layout

To visualize your layout parameter settings and help you experiment with different values there are two packages available. The package `layout` (originally written by Kent McPherson and converted to `LATEX 2 $\varepsilon$`  by Johannes Braams) provides the command `\layout`, which produces a graphical representation of the current page parameters with all sizes reduced by a factor of two. If the class option `twoside` is used then two pages are produced.

A more flexible solution is provided by the package `layouts` written by Peter Wilson. This package can be used for two purposes: to produce an abstract graphical representation of the layout parameters (not reflecting the current settings) via `\pagediagram` (as shown in the next example) or to produce trial layouts that show the effect of setting parameters to trial values and then applying the

command `\pagedesign`. In either mode `\setlayoutscale` sets the scale factor to the specified value.

The circle is at 1 inch from the top and left of the page. Dashed lines represent  $(\text{\hoffset} + 1 \text{ inch})$  and  $(\text{\voffset} + 1 \text{ inch})$  from the top and left of the page.



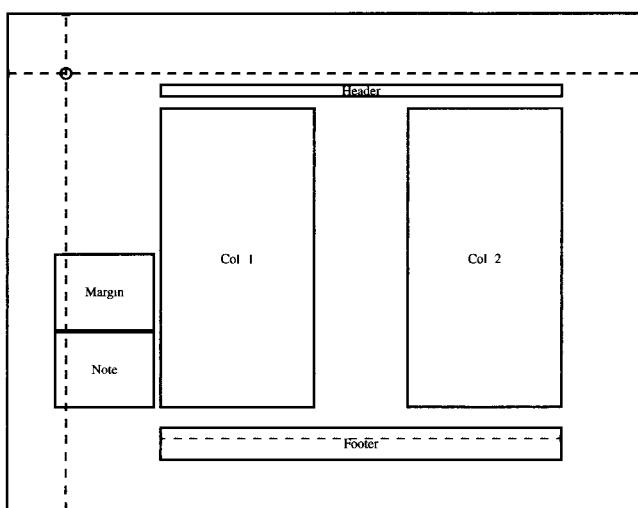
To produce a trial layout you first have to specify suitable values for all page layout parameters. For each parameter *param*, there exists a declaration `\try{param}` that accepts the trial values for this parameter as an argument. For example, `\try{headsep}{18pt}` would produce a layout with `\headsep` set to 18pt.

In addition, there are four Boolean-like declarations: `\oddpagefalse` produces an “even page” (default is to produce odd pages), the declaration `\twocolumnlayouttrue` produces a two-column layout (default is a single-column layout). The command `\reversemarginpartrue` mimics the result of L<sup>A</sup>T<sub>E</sub>X’s `\reversemarginpar`, and `\marginparswitchfalse` prevents marginal notes from changing sides between verso and recto pages (a suitable setting for asymmetrical layouts, which are easily produced using the `geometry` package; see page 208).

To facilitate the specification of trial values you can start your trial by specifying `\currentpage`. It sets all trial values and Boolean switches to the values currently used in your document.

By default, the footer has a height of one line, as L<sup>A</sup>T<sub>E</sub>X has no explicit parameter to change the box size of the footer. However, depending on the page style used this choice might not be appropriate, as the footer box defined by the page style might have an exceptionally large depth. To produce a diagram that is (approximately) correct in this case, one can set the footer box height and depth explicitly using `\setfootbox` as we do in the example below.

This example also shows that you can combine this package with the `calc` package to allow arithmetic expressions in your trial declarations.



Lengths are to the nearest pt.

page height = 614pt	page width = 795pt
\hoffset = 0pt	\voffset = 0pt
\evensidemargin = 120pt	\topmargin = 16pt
\headheight = 12pt	\headsep = 18pt
\textheight = 370pt	\textwidth = 500pt
\footskip = 40pt	\marginparsep = 11pt
\marginparpush = 5pt	\columnsep = 120pt
\columnseprule = 3.0pt	

4-2-2

```
\usepackage{calc,layouts}
\setlayoutscale{0.3}
\currentpage
\oddpagefalse
\twocolumntrue

\trypaperwidth{11in}
\trypaperheight{8.5in}
\trytextwidth{500pt}
\trytextheight{\topskip
+ 30\baselineskip}
\trycolumnssep{120pt}
\trycolumnsrule{3pt}

\tryheadheight{12pt}
\tryheadsep{18pt}
\tryfootskip{40pt}

\tryevensidemargin{120pt}
\setfootbox{12pt}{24pt}
\setlabelfont{\tiny}
\drawdimensionsfalse
\printheadingsfalse
\pagedesign
```

A number of display control statements influence the visual representation of the printed page designs, some of which were used in the previous example. The *Controlling the presentation* most important are discussed here, whilst others are described in the documentation accompanying the package.

With the `\setlabelfont` declaration the font size used for the textual labels can be changed. Similarly, `\setparameter{textfont}` influences the font sizes for parameters if they are shown (e.g., Example 4-2-1 on the preceding page).

The heading text displayed on top of the example can be suppressed with `\printheadingsfalse`. The Boolean flag `\printparametersfalse` suppresses

the tabular listing of parameter values below the diagram. A similar table can be generated separately using the command `\pagevalues`.

With `\drawdimensionstrue` arrows are drawn to indicate where parameters apply (by default, this feature is turned on in `\pagediagram` and off when `\pagedesign` is used).

*Visualizing other layout objects*

The layouts package is not restricted to page layouts. It also supports the visualization of other objects. Eight “diagram” commands can be used to show the general behavior of other L<sup>A</sup>T<sub>E</sub>X layout parameters. The `\listdiagram` command visualizes the list-related parameters (it is used in Figure 3.3 on page 145). The `\tocdiagram` command shows which parameters influence table of content lists and how they relate to each other. Float-related parameters are visualized using `\floatdiagram` and `\floatpagediagram`. Parameters for sectioning commands are displayed with `\headingdiagram`, and parameters related to footnotes and general paragraphs can be shown with `\footnotediagram` and `\paragraphdiagram`. Finally, the `\stockdiagram` command produces a page layout diagram similar to `\pagediagram` but displays parameters available only in the memoir document class and its derivatives (see Section 4.6.2 on page 237).

There also exist corresponding “design” commands, such as `\listdesign`, `\tocdesign`, `\floatdesign`, `\floatpagedesign`, `\headingdesign`, and so on, that allow you to experiment with different parameter settings. For each parameter a declaration `\try(param)` allows you to set its value for visualization. The full list of parameters supported this way is given in the package documentation. But if you know the applicable L<sup>A</sup>T<sub>E</sub>X parameters (or look them up on the “diagram” command results) you can start experimenting straight away.

#### 4.2.2 A collection of page layout packages

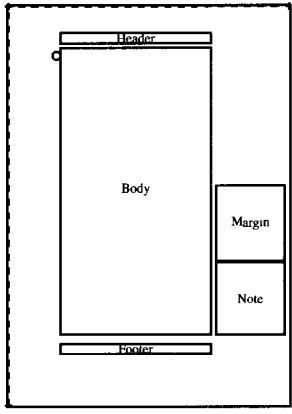
Because the original L<sup>A</sup>T<sub>E</sub>X class files were based on American page sizes, European users developed several packages that adapt the page layout parameters for metric sizes. All such packages are superseded by the `typearea` or `geometry` package (described in the next two sections) and for new documents we recommend that you use these packages. As you will find the original attempts still in the archives, we mention them here in passing.

Examples of such packages are `a4`, which generates rather small pages; `a4dutch` (by Johannes Braams and Nico Poppelier), which is well documented; and `a4wide` (by Jean-François Lamy), which produces somewhat longer lines. Moreover, often there exist locally developed files under such names. For A5 pages one has the package files `a5` and `a5comb` (by Mario Wolczko). The problem with all of these early packages was that they allowed little to no customization with respect to the size and placement of the text area and, for some of them, incompatible implementations exist.

A more general approach was taken by the `vmargin` package written by Volker Kuhlmann. His package supports a variety of paper sizes and allows you to specify a number of layout parameters with a single declaration, calculating others

from the input (a number of variant declarations exist). In the example below the margins are specified and the text area is calculated.

4-2-3



```
\usepackage{vmargin}
\setpapersize[portrait]{A5}
\setmarginsrb{80pt}{40pt}%
{120pt}{80pt}%
{12pt}{10pt}%
{12pt}{30pt}%
\setlength\marginparwidth{100pt}
% Code to display the resulting layout:
\usepackage{layouts}
\newcommand\showpage{%
\setlayoutscale{0.25}\setlabelfont{\tiny}%
\printheadingsfalse\printparametersfalse
\currentpage\pagedesign}
\showpage
```

The package internally cancels the default offset of one inch (added normally by TeX output devices) by using a negative `\hoffset` and `\voffset`, a fact that can cause some surprise. This behavior can be seen in the example, where the dashed lines normally indicating this offset have vanished behind the page border and only the circle at (1 inch, 1 inch) remains.

### 4.2.3 typearea—A traditional approach

In books on typography one usually finds a section that deals with page layout, often describing construction methods for placing the text body and providing one or the other criterion for selecting text width, number of text lines, relationship between margins, and other considerations.

The package `typearea` by Markus Kohm and Frank Neukam, which is distributed as part of the KOMA-Script bundle, offers a simple way to deploy one of the more traditional page layout construction methods that has been used for many books since the early days of printing.

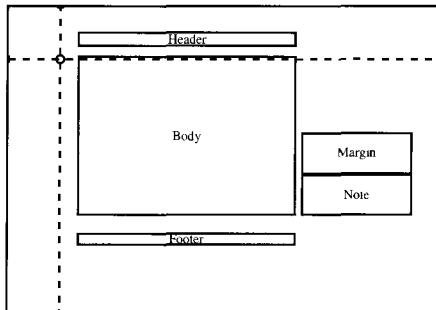
In a nutshell, the page layout generated by `typearea` provides a text body with the same spatial relationship as given by the paper size on which the document is being printed. In addition, the outer margin will be twice as wide as the inner margin and the bottom margin will be twice as wide as the top margin.

The construction method works by dividing the paper horizontally and vertically into  $n$  equal slices and then using one slice at the top and inner edges and two slices at the bottom and outer edges for the margins. By default, the variable  $n$  is calculated automatically by the package. It can also be requested explicitly (for example, to overwrite a configuration setting in the file `typearea.cfg`) by using the option `DIVcalc`. This option works by examining the document font and selecting a value that results in approximately 60–70 characters per text line,

assuming a portrait page. Alternatively, one can explicitly set the value of  $n$  by specifying the option `DIV $n$` , resulting in  $n$  slices. As a third possibility, one can specify the option `DIVclassic`, which results in a page layout close to that found in certain types of medieval books.

The page height resulting from the chosen or calculated DIV value is automatically adjusted to produce an integral number of text lines. For this approach to work, the effective `\baselineskip` used throughout the document has to be established first. Thus, when using a package like `setspace` or applying the command `\linespread` this step should be taken prior to loading `typearea`.

For defining the paper `typearea` offers all of the paper size options of L<sup>A</sup>T<sub>E</sub>X's standard classes (see Table 4.1 on page 195) as well as all sizes of the ISO-A, ISO-B, and ISO-C series (e.g., `a0paper` or `c5paper`). To change the text orientation use `landscape`, as in the example below.



```
\usepackage[a5paper,landscape,DIVcalc]{typearea}
% to display the resulting layout:
\usepackage{layouts}
\newcommand\showpage{%
  \setlayoutscale{0.27}\setlabelfont{\tiny}%
  \printheadingfalse\printparametersfalse
  \currentpage\pagedesign}
\showpage
```

4-2-4

The calculated DIV value is recorded in the `.log` file of the L<sup>A</sup>T<sub>E</sub>X run together with the values chosen for other page parameters. In the above example this value was 7, so instead of `DIVcalc` we could have used `DIV7`.

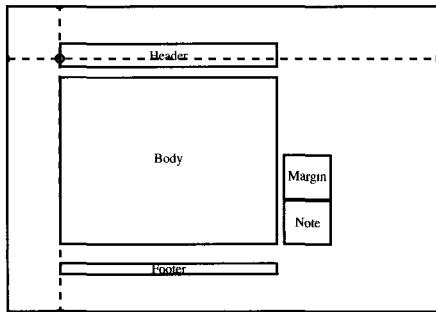
*Determining the body area* So far, we have explained how the package chooses the text body dimensions and how it places that body on the page, but we have not discussed whether the running header and footer participate in that calculation. This issue must be decided depending on their content. If, for example, the running header contains a lot of material, perhaps even with a rule underlining it, and thus contributes considerably to the grey value of the page, it is best regarded as part of the page body. In other cases it might be more appropriate to consider it as being part of the margin (e.g., if it is unobtrusive text in small type). For the same reason a footer holding only the page number should normally be considered as lying outside the text body and not contributing to the placement calculations.

The choices for a particular document can be explicitly specified with the options `headinclude`, `footinclude`, `headexclude`, and `footexclude`. The latter two options are used by default. With large DIV values (i.e., small margins), excluding the header or footer might make it fall off the page boundary so you may have to adjust one or the other setting.

In a similar fashion (using `\mpinclude` or `\mpexclude`), one can include or exclude the `\marginpar` area into the calculation for left and right margins. This, too, is turned off by default but it might be appropriate to include it for layouts with many objects of this type.

The header size is by default 1.25 text lines high. This value can be adjusted by using an option of the type `numheadlines`, where `num` is a decimal number, such as `2.3`, denoting the number of text lines the header should span.

The next example has header and marginals included and the header size is enlarged to 2.5 lines. Compare this example to the layout in Example 4-2-4 on the preceding page, where header, footer, and marginals are excluded.

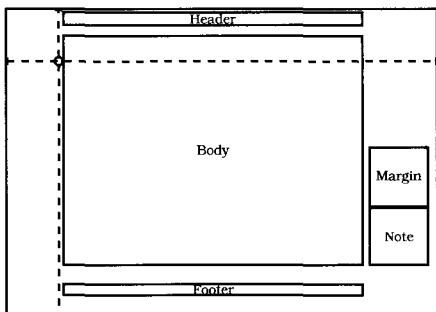


4-2-5

```
\usepackage[a5paper,landscape,
2.5headlines,
headinclude,\mpinclude,
DIVcalc]{typearea}
\usepackage{layouts}
% \showpage as previously defined
\showpage
```

Depending on the type of binding for the final product, more or less of the inner margin will become invisible. To account for this loss of white space the package supports the option `BCOR{val}`, where `val` is the amount of space (in any `LATEX` unit) taken up by the binding. For example, `BCOR1.2cm` would subtract 1.2 centimeters from the page width prior to doing the page layout calculations.

As an alternative to customizing the layout through options to the package, one can perform the parameter calculations with the command `\typearea`; for details, see the KOMA-Script documentation. This ability is useful, for example, if a document class, such as one of the classes in the KOMA-Script bundle, already loads the `typearea` package and you want to use an unusual body font by loading it in the preamble of the document. In that case the layout calculations need to be redone to account for the properties of the chosen font.



4-2-6

```
\usepackage[a5paper,landscape]{typearea}
\usepackage{bookman}

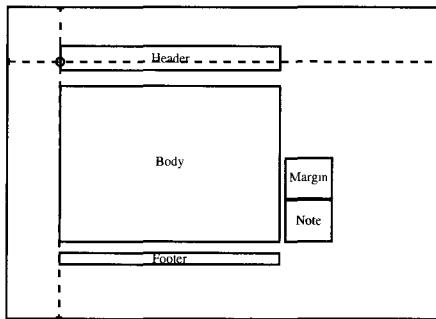
% syntax: \typearea[<binding corr.>]{<slices>}
\typearea[10mm]{11}

\usepackage{layouts}
% \showpage as previously defined
\showpage
```

#### 4.2.4 geometry—Layout specification with auto-completion

The `geometry` package written by Hideo Umeki provides a comprehensive and easy-to-use interface to all geometrical aspects of the page layout. It deploys the `keyval` package so that all parameters (and their values) can be specified as options to the `\usepackage` declaration.

In contrast to the `typearea` package, `geometry` does not implement a certain typographical concept but rather carries out specifications as requested. It knows, however, about certain relationships between various page parameters and in case of incomplete specifications can calculate the remaining parameter values automatically. The following example shows a layout very similar to the one produced by `typearea` in Example 4-2-5 on the preceding page. Here a number of values have been explicitly set (e.g., those for the top and left margins), but the size of the page body has been automatically calculated from the paper size (`a5paper`), the values for top margin (`tmargin`) and left margin (`lmargin`), and a specified margin ratio of 1:2 (`marginratio`).



```
\usepackage[marginratio=1:2,
           paper=a5paper,landscape=true,
           tmargin=52pt,lmargin=74pt,
           headheight=30pt,marginparwidth=62pt,
           includehead,includemp]{geometry}
\usepackage{layouts}
% \showpage as previously defined
\showpage
```

4-2-7

The example also shows that with Boolean options it is permissible to leave out the value part (which then defaults to `=true`); with all other options the value part is mandatory.

The remainder of this section discusses the various page layout aspects that are supported by `geometry`. In most cases there is more than one way to achieve the same result because some of the parameters have to satisfy certain relations. If your specification violates such a relation, `geometry` will warn you and then ignore one or the other option setting.

The paper size can be specified with the `paper` option, which accepts the *Paper sizes* values `a0paper` to `a6paper`, and `b0paper` to `b6paper`. Alternatively, the values `letterpaper`, `legalpaper`, and `executivepaper` can be used. For convenience you are allowed to denote the paper size by specifying the named paper as an option; for example, `a5paper` is equivalent to the specification `paper=a5paper`.

When formatting for a computer display you might want to try the option `screen`. To specify other nonstandard sizes you can use `paperwidth` and `paperheight` to define the appropriate dimensions explicitly.

With respect to general page characteristics, `geometry` supports the Boolean options `twoside`, `landscape` (switching paper height and width), and `portrait`. Obviously, `portrait=false` is just a different way of specifying `landscape`.

*General page characteristics*

If a certain part of the page becomes invisible due to the binding method, you can specify this loss of white space with the option `bindingoffset`. It will add the specified value to the inner margin.

When the Boolean option `twocolumn` is specified, the text area will be set up to contain two columns. In this case the separation between columns can be specified through the option `columnsep`.

In Section 4.2.3 describing the `typearea` package, we stated that, depending on the nature of the document, it may be appropriate to consider the running header and/or footer (and in some cases even the part of the margin taken up by marginal notes) as being part of the text body. By default, `geometry` excludes the header, footer, and marginals. As these settings modify the relationship between body and margin sizes used for calculating missing values, they should be set appropriately. To change the defaults, a number of Boolean options<sup>1</sup> are available: `includemp`, to include the marginals, which is seldom necessary; `includehead`, to be used with heavy running headers; `includefoot`, which is rarely ever necessary, as the footer normally contains only a page number; and `includeheadfoot` and `includeall`, which are shorthand for combinations of the other options.

*What constitutes the body area*

Footnotes are always considered to be part of the text area. With the option `footnotesep` you specify only the separation between the last text line and the footnotes; the calculation of the margins remains unaffected.

For specifying the text body size several methods are available; the choice of which to use is largely a matter of taste. You can explicitly specify the text area size by giving values for `textwidth` and `textheight`. In that case you should normally ensure that `textheight` holds an integral number of text lines to avoid underfull box messages for pages consisting only of text. A convenient way to achieve this goal is to use the `lines` option, which calculates the appropriate `\textheight` using the current values for `\baselineskip` and `\topskip`.

*Text area*

Alternatively, you can set the Boolean option `heightrounded`, in which case `geometry` will adjust the `\textheight` appropriately. This Boolean option is especially useful if the body size is calculated automatically by the package—for example, if you specify the values for only some of the margins and let the package work out the rest.

As an alternative to specifying the text area and having the package calculate the body size by adding the sizes of the header, footer, and/or marginals as specified through the above options, you can give values for the whole body area and have the package calculate the text area by subtracting. This is done with the options `width` and `height` (this approach, of course, differs from the previ-

<sup>1</sup>The `typearea` package offers the same functionality, with similar (though in fact different) option names, such as `headinclude` instead of `includehead`.

ous approach only if you have included header and/or footer). If this method is used consider specifying `heightrounded` to let the package adjust the calculated `\textheight` as needed.

If you do not like specifying fixed values but prefer to set the body size relative to the page size, you can do so via the options `hscale` and `vscale`. They denote the fraction of the horizontal or vertical size of the page that should be occupied by the body area.

The size of the margins can be explicitly specified through the options *Margins* `lmargin`, `rmargin`, `tmargin`, and `bmargin` (for the left, right, top, and bottom margins, respectively). If the Boolean option `twoside` is `true`, then `lmargin` and `rmargin` actually refer to the inner and outer margins, so the option names are slightly misleading. To account for this case, the package supports `inner` and `outer` as alternative names—but remember that they are merely aliases. Thus, if used with the `asymmetric` option (described below), they would be confusing as well. To give you even more freedom there exists another set of option names: `left`, `right`, `top`, and `bottom`. If you choose to specify only verso pages (the recto pages being automatically produced by selecting `twoside` or `asymmetric`), then the first or the last set of names is probably the best choice.

If none, or only some, of the margin sizes are specified, the missing ones are calculated. Given the equations

$$\text{paperwidth} = \text{left} + \text{width} + \text{right} \quad (4.1)$$

$$\text{paperheight} = \text{top} + \text{height} + \text{bottom} \quad (4.2)$$

then knowing two values from the righthand side allows the calculation of the third value (instead of `width` or `height` the body area might be specified through some of the other methods discussed above). If only one value from the righthand side is specified, the package employs two further equations to reduce the free variables:

$$\text{left/right} = \text{hmarginratio} \quad (4.3)$$

$$\text{top/bottom} = \text{vmarginratio} \quad (4.4)$$

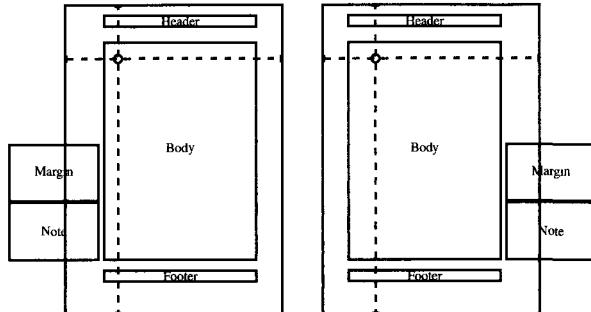
The default value for the `hmarginratio` option is `2:3` when `twoside` is `true`, and otherwise `1:1`. The default for `vmarginratio` is `2:3` without exception.

The allowed values for these “ratio” options are restricted: both numbers have to be positive integers less than 100 separated with a colon. For example, you would use `4:5` instead of `1:1.25`.

If you wish to center the body area, use the option `centering`. It is a convenient shorthand for setting `hmarginratio` and `vmarginratio` both to `1:1`.

In standard L<sup>A</sup>T<sub>E</sub>X classes the option `twoside` actually fulfills a dual purpose: beside setting up the running header and footer to contain different content on verso and recto pages, it automatically implements a symmetrical layout with left and right margins (including marginal notes) swapped on verso pages. This outcome is shown in the next example, which also highlights the fact that `geometry`

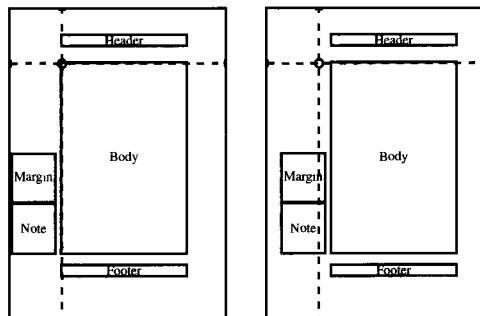
by default selects a very large text area but does not adjust the size of the marginal boxes to fit in the remaining margin.



4.2-8

```
\usepackage[a6paper,twoside]
{geometry}
\usepackage{layouts}
% \showpage as previously defined
\showpage \newpage \showpage
```

With the `geometry` package, however, asymmetrical page layouts are possible, simply by using the option `asymmetric`. The use of `bindingoffset` in the next example proves that an asymmetrical two-sided layout is indeed produced, as the offset is applied to the inner margins and not always to the left margin, even though the marginal notes always appear on the left. As we want the larger margin on the left, we have to change `hmarginratio` appropriately. At first glance the right margin on the verso page might appear incorrectly large given a marginal ratio of 2:1; this is due to the `bindingoffset` being added to it.



4.2-9

```
\usepackage[a6paper,asymmetric,
bindingoffset=18pt,
marginparwidth=.8in,reversemp,
hmarginratio=2:1,vmarginratio=4:5,
left=1in,top=1in]{geometry}
\usepackage{layouts}
% \showpage as previously defined
\showpage \newpage \showpage
```

The dimensions for the running header and its separation from the text area can be specified through the options `headheight` and `headsep`. The distance between the text area and the footer is available through `footskip`. There also exist the Boolean options `nohead`, `nofoot`, and `noheadfoot`, which set these dimensions to zero. In most circumstances, however, it is better to use `ignorehead`, etc. as this will allow you to attach the header or footer on one or the other page without affecting the margin calculations.

As most documents do not contain many marginal notes, the space occupied by them by default does not count toward the margin calculations. This space can be specified with `marginparwidth`, and the separation from the text area can be

*Running header  
and footer*

*Marginal notes*

set with `marginparsep`. Unless `includemp` is specified it is the user's responsibility to ensure that this area falls within the calculated or specified margin size.

By default, the marginal notes appear in the outer margin. By specifying the Boolean option `reversemp` this set-up can be reversed.

*Miscellaneous features* Instead of using an external package, such as `layouts`, to visualize the results produced by `geometry`, one can use its built-in option `showframe`. By default, all settings, including any calculated values, are recorded in the transcript file of the current L<sup>A</sup>T<sub>E</sub>X run. Setting the Boolean option `verbose` ensures that these settings are also displayed on the terminal.

Some T<sub>E</sub>X extensions or device drivers such as pdfT<sub>E</sub>X or VT<sub>E</sub>X like to know about the dimensions of the paper that is being targeted. The `geometry` package accounts for this by providing the options `pdftex`, `vtex`, `dvipdfm`, and `dvips`. Naturally, at most one of them should be specified. If a document is processed with the pdfT<sub>E</sub>X program then the `pdftex` option is automatically selected (and the others are disabled).

Like most packages these days, `geometry` supports the extended syntax of the `calc` package if the latter is loaded before `geometry`.

To account for unusual behavior of the printing device, L<sup>A</sup>T<sub>E</sub>X maintains two dimension registers, `\hoffset` and `\voffset`, which will shift all output (on every page) horizontally to the right and vertically downward by the specified amount. The package supports the setting of these registers via the options `hoffset` and `voffset`. They have no effect on the calculation of other page dimensions.

*Magnification* The T<sub>E</sub>X program offers a magnification feature that magnifies all specified dimensions and all used fonts by a specified factor. Standard L<sup>A</sup>T<sub>E</sub>X has disabled this feature, but with `geometry` it is again at the disposal of the user via the option `mag`. Its value should be an integer, where 1000 denotes no magnification. For example, `mag=1414` together with `a5paper` would result in printing on `a4paper`, as it enlarges all dimensions by  $1.414 (= \sqrt{2})$ , the factor distinguishing two consecutive paper sizes of the ISO-A series. This ability can be useful, for example, if you later wish to photomechanically reduce the printed output to achieve a higher print resolution. As this option also scales fonts rather than using fonts designed for a particular size, it is usually not adequate if the resulting (magnified) size is your target size.

When magnification is used, you can direct T<sub>E</sub>X to leave certain dimensions unmagnified by prepending the string `true` to the unit. For example, `left=1truein` would leave a left margin of exactly one inch regardless of any magnification factor. Implicitly specified dimensions (such as the paper size values, when specifying a `paper` option) are normally subject to magnification unless the option `truedimen` is given.

*Shortcuts* The previously described options allow you to specify individual values, but for the most common cases `geometry` also provides combination options. They allow you to set several values in one pass by specifying either a single value (to be used repeatedly) or a comma-separated list of values (which must be surrounded by braces so that the commas are not mistaken for option delimiters).

The option `papersize` takes a list of two dimensions denoting the horizontal and vertical page dimensions.

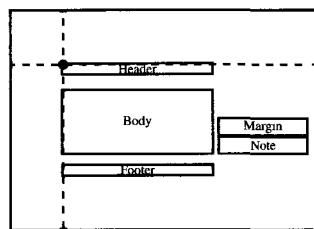
The option `hmargin` sets the left and right margins, either to the same value if only a single value is given, or to a list of values. Similarly, `vmargin` sets the top and bottom margins. This operation can sometimes be shortened further by using the option `margin`, which passes its value (or list) to `hmargin` and `vmargin`. In the same way `marginratio` passes its value to `hmarginratio` and `vmarginratio` for further processing.

The text area dimensions can be specified using the `body` option, which takes one or two values setting `textwidth` and `textheight`. Alternatively, you can use the option `total`, which is a shortcut for setting `width` and `height`. You can also provide one or two scaling factors with the option `scale` that are then passed to `hscale` and `vscale`.

If the `geometry` package is used as part of a class you may wish to overwrite some of its settings in the preamble of your document. In that case the `\usepackage` option interface is of little use because the package is already loaded. To account for such situations the package offers the command `\geometry`, which takes a comma-separated list of options as its argument. It can be called multiple times, each time overwriting the previous settings. In the next example its use is demonstrated by first loading the package and setting all margins to one inch and the header, footer, and marginals to be part of the body area, and then changing the right margin to two inches and excluding the marginals from the calculation.

*Preamble usage*

```
\usepackage[a6paper,landscape,
           margin=1in,includeall]{geometry}
% overwriting some values:
\geometry{right=2in,ignoremp}
\usepackage{layouts}
% \showpage as previously defined
\showpage
```



4-2-10

Two other options might be handy when using the `\geometry` interface. With `reset` you restore the package defaults and with `pass` you basically disable the package itself.

#### 4.2.5 `\scape`—Typesetting individual pages in `landscape` mode

For most documents the longer side of the paper corresponds to the vertical direction (so-called *portrait* orientation). However, for some documents, such as slides and tables, it is better to use the other (*landscape*) orientation, where the longer side is horizontally oriented. Modern printers and dvi drivers usually allow printing in both orientations.

The landscape and portrait orientations require different page layouts, and with packages like `geometry` you have the tools at hand to design them as needed. But sometimes it is desirable to switch between portrait and landscape mode for only some pages. In that case the previously discussed packages do not help, as they set up the page design for the whole document.

For this case you can use the `lscape` package by David Carlisle that defines the environment `landscape` to typeset a selected set of pages in landscape orientation without affecting the running header and footer. It works by first ending the current page (with `\clearpage`, thereby typesetting any dangling floats). It then internally exchanges the values for `\textheight` and `\textwidth` and rotates every produced page body within its scope by 90 degrees. For the rotation it deploys the `graphics` package, so it works with any output device supported by that package capable of rotating material. When the environment ends it issues another `\clearpage` before returning to portrait mode.

For rotating individual floats, including or excluding their captions, a better alternative is provided by the `rotating` package, described in Section 6.3.3.

#### 4.2.6 crop—Producing trimming marks

When producing camera-ready copy for publication, the final printing is normally done on “stock paper” having a larger size than the logical page size of the document. In that case the printed copy needs trimming before it is finally bound. For accurate trimming the printing house usually requires so-called crop marks on each page. Another reason for requiring crop marks is the task of mounting two or more logical pages onto a physical one, such as in color production where different colors are printed separately.

The `crop` package created by Melchior Franz supports these tasks by providing a simple interface for producing different kinds of crop marks. It also offers the ability to print only the text or only the graphics from a document, and the chance of inverting, mirroring, or rotating the output, among other things—all features useful during that part of the printing process.

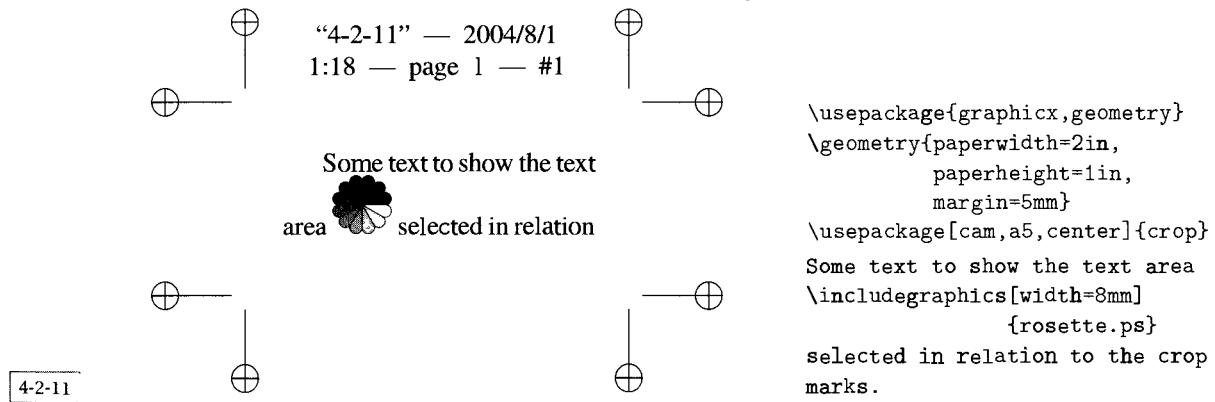
Crop marks can be requested by using one of the following options:

- `cam` Produces four marks that show the logical paper dimensions without touching them (see Example 4-2-11 on the next page). They are mainly intended for camera alignment.
- `cross` Produces four large crosses at the corners of the logical page touching its edges.
- `frame` Produces a frame around the logical page; mainly intended for clearly visualizing the page dimensions.

The package assumes that the `\paperheight` and `\paperwidth` dimensions correctly reflect the size of the *logical* page you want to produce. The size of the *physical* page (the stock paper) you are actually printing on is then given as

an option to the package. Options include `a0`, `a1`, `a2`, `a3`, `a4`, `a5`, `a6`, `b0`, `b1`, `b2`, `b3`, `b4`, `b5`, `b6`, `executive`, `legal`, and `letter`. If you use the physical paper in landscape orientation (i.e., with the long side horizontally), you can also specify the option `landscape`. If none of these options matches your physical paper sizes, you can specify the exact sizes through the options `width` and `height`, both of which take dimensional values.

The following example sets up an artificially small logical page (to fit the example area of this book) using the `geometry` package and centers it on a physical page of A5 size. However, since all our examples are actually cropped to their “visible” size and since, for obvious reasons, we have not actually marked the borders of the A5 paper, you cannot see that it was properly centered at one stage—either believe us or try it yourself.



It should be clear from the description and the example that this package should be loaded *after* the document layout has been specified.

The informational text between the top crop marks is added by default. It can be suppressed by adding the option `noinfo`, though it is usually a good idea to keep it. The information contains both the page number (as known to L<sup>A</sup>T<sub>E</sub>X) and a page index, which starts with 1 and is incremented for every page being printed. Especially with large publications using several page numbering methods at once, this is a helpful device to ensure that pages are not misordered.

Several options of the `crop` package rely on support given by the printer driver. If no driver option is explicitly given, the package tries to determine the driver from installation settings for the `graphics` or `color` package. It is also possible to indicate the driver explicitly by using options such as `dvips`, `pdflatex`, or `vtx`. If one of these options is selected the paper size information is passed to the external driver program, which is important if you want to view the document using `ghostview` or similar programs.

If you want to print graphics separately—for example, because running the complete document through a color printer is infeasible—you can produce different versions of the same document: one containing only the text but no graphics

(or, more precisely, without graphics included via `\includegraphics`) and one containing only the graphics (or, more precisely, with all text printed in the color “white”). These effects can be achieved using the options `nographics` and `notext`, respectively. Clearly, the latter option can be used only if the target device is capable of understanding color commands since internally the `color` package is being deployed. The next example<sup>1</sup> shows the use of the `nographics` and `cross` options; compare it to the output of Example 4-2-11.

“4-2-12” — 2004/8/1  
1:18 — page 1 — #1

Some text to show the text  
area        selected in relation

```
\usepackage{graphicx,geometry}
\geometry{paperwidth=2in,
          paperheight=1in,
          margin=5mm}
\usepackage[cross,a5,nographics]{crop}
Some text to show the text area
\includegraphics[width=8mm]
               {rosette.ps}
selected in relation to the crop
marks.
```

4-2-12

Three other options require the output device to be able to obey the extended commands of the `graphics` and `color` packages for rotation, mirroring, and background coloring. With the option `rotate` the pages are turned through 180 degrees. The option `mirror` flips each page as shown in the next example. Finally, the option `invert` will invert white and black, so that the text appears in white on a black surface.

“4-2-13” — 2004/8/1  
1:18 — page 1 — #1

Some text to show the text  
area        selected in relation



```
\usepackage{graphicx,geometry}
\geometry{paperwidth=2in,
          paperheight=1in,
          margin=5mm}
\usepackage[frame,a5,mirror]{crop}
Some text to show the text area
\includegraphics[width=8mm]
               {rosette.ps}
selected in relation to the crop
marks.
```

4-2-13

<sup>1</sup>The cross crop marks look admittedly rather weird at this measure.

### 4.3 Dynamic page data: page numbers and marks

$\text{\LaTeX}$ 's output routine, which produces the typeset pages, works asynchronously. That is,  $\text{\LaTeX}$  assembles and prepares enough material to be sure that a page can be filled and then builds that page, usually leaving some residual material behind to be used on the next page(s). Thus, while preparing headings, paragraphs, and other page elements, it is usually not known on which page this material will eventually be placed because  $\text{\LaTeX}$  might eventually decide that this material will not fit on the current page. (We have already discussed this problem in the section about page-wise footnote numbering.)

When the final page is typeset, we might want to repeat some information from its contents in the running header or footer (e.g., the current section head), to give the reader extra guidance. You cannot save this information in commands when the material is collected; during this phase  $\text{\LaTeX}$  often reads too far ahead and your command would then contain data not appearing on the final page.  $\text{\LaTeX}$  solves this problem by providing a mark mechanism through which you can identify data as being of interest for the assembled page. In the output routine all marks from the page are collected and the first and the last mark are made available. The detailed mechanism is explained in this section together with some useful extension packages.

#### 4.3.1 $\text{\LaTeX}$ page numbers

The page number is controlled through a counter named `page`. This counter is automatically stepped by  $\text{\LaTeX}$  whenever it has finished a page—that is, *after* it has been used. Thus, it has to be initialized to 1, whereas most other  $\text{\LaTeX}$  counters require an initialization to 0 as they are stepped just before they get used.

The command to access the typographical representation of the page number is `\thepage`, following standard  $\text{\LaTeX}$  convention. There is, however, another subtle difference compared to other  $\text{\LaTeX}$  counters: the `\thepage` command is not defined by the  $\text{\LaTeX}$  kernel but instead comes into existence only after the first execution of a `\pagenumbering` declaration, which typically happens in the document class file.

The best (though perhaps not the most convenient) way to get at the page number for the current page in the middle of the text is via a combination of the commands `\label` and `\pageref`, which should be put directly one following the other so that no page break can interfere.

We are now on page 6. This type of coding always gives correct results while “page 6”, though okay

4.3-1

6

here, will be wrong at a later point in the paragraph, such as here: “page 6”, because  $\text{\LaTeX}$  decided to break

7

We are now on page~\label{p1}\pageref{p1}. This type of coding always gives correct results while ‘‘page \thepage{}’’, though okay here, will be wrong at a later point in the paragraph, such as here: ‘‘page \thepage{}’’, because \LaTeX{} decided to break the paragraph over two pages.

Because of the asynchronous nature of the output routine you cannot safely use `\thepage` within the document body. It is reliable only in declarations that influence the look and feel of the final page built by the output routine.

### `\pagenumbering{style}`

The `\pagenumbering` command resets the page counter to 1 and redefines the command `\thepage` to `\style{page}`. Ready-to-use page counter styles include: `Alph`, `alph`, `Roman`, `roman`, and `arabic` (see Section A.1.4).

For example, an often seen convention is to number the pages in the front matter with `roman` numerals and then to restart the page numbers using `arabic` numbers for the first chapter of the main matter. You can manually achieve this effect by deploying the `\pagenumbering` command twice; the `\frontmatter` and `\mainmatter` commands available with the `book` class implement this set-up implicitly behind the scenes.

### 4.3.2 `lastpage`—A way to reference it

Standard L<sup>A</sup>T<sub>E</sub>X has no way to refer to the number of pages in a document; that is, you cannot write “this document consists of 6 pages” or generate “page 5 of 10” without manually counting the pages yourself. The package `lastpage` by Jeffrey Goldberg sets out to overcome this problem by automatically generating a label with the name `LastPage` on the last page, so that you can refer to its page number via `\pageref{LastPage}`. Example 4-4-5 on page 226 demonstrates its use.

The string produced by that call to `\pageref` is the content of `\thepage` as it would appear on the last page. If your document restarts page numbering midway through—for example, when the front matter has its own numbering—this string will not reflect the absolute number of pages.

The package works by generating the label within the `\AtEndDocument` hook, making sure that any pending floats are placed first. However, as this hook might also be used by other packages to place textual material at the end of the document, there is a chance that the label may be placed too early. In that case you can try to load `lastpage` after the package that generates this extra material.

### 4.3.3 `chappg`—Page numbers by chapters

For some publications it is required to restart numbering with every chapter and to display the page number together with the chapter number on each page. This can already be done with the commands at our disposal by simply putting

```
% Page numbers per chapter (repeat after each \chapter):
\pagenumbering{arabic} % first reset page numbering and then overwrite ...
\renewcommand\thepage{\thechapter--\arabic{page}} % ... the display style
```

after each `\chapter` command. But this technique is clumsy and requires us to put a lot of layout information in our document, something that is better avoided.

A better approach is to use the package `chappg`, originally written by Max Hailperin and later reimplemented and extended by Robin Fairbairns. It works with any document class that has a `\chapter` command and provides a new page numbering style `bychapter` to achieve the desired page numbering scheme. Furthermore, it extends the `\pagenumbering` command to accept an optional argument that enables you to put a prefix different from the chapter number before the page number. This ability is, for example, useful in the front matter where typically unnumbered headings are used.

4-3-2

... here we are in the middle of the front matter where

Preface-1

chapters are usually unnumbered.

Preface-2

```
\usepackage{chappg}
% \chapter*[Preface] % --- not shown
\pagenumbering[Preface]{bychapter}
\ldots here we are in the middle of
the front matter where chapters are
usually unnumbered.
```

In fact, by exerting some care you can even use this package together with a class that does not define a `chapter` command. Suppose your highest heading level is `\section` and each section automatically starts a new page (the latter is an important requirement). Then the declaration

```
\makeatletter \c@addtoreset{page}{section} \makeatother
\pagenumbering[\thesection]{bychapter}
```

will give you page numbers within sections. However, if sections do not start a new page this approach might fail, as L<sup>A</sup>T<sub>E</sub>X may have seen an upcoming section and incremented `\thesection` without actually putting that section onto the current page. If so, you will experience the same problem that we saw earlier with respect to `\thepage`.

Finally, the en dash between the prefix and the page number is also customizable, since it is produced by the command `\chappgsep`. Thus,

```
\renewcommand\chappgsep{/}
```

will give you pages like 3/1, 3/2, 3/3, 3/4, and so on, if “3” is the current chapter number.

#### 4.3.4 L<sup>A</sup>T<sub>E</sub>X mark commands

The T<sub>E</sub>X primitive `\mark`, which you may encounter inside package code dealing with page layout or output routines, is ultimately responsible for associating some text (its argument) with a position on a page (i.e., the position where the `\mark`

is executed). When producing the final page  $\text{\TeX}$  makes the first mark on the assembled page available in  $\text{\firstmark}$ , the last in  $\text{\botmark}$ , and the  $\text{\botmark}$  from the previous page as  $\text{\topmark}$ . If there are no marks on that page then  $\text{\firstmark}$  and  $\text{\botmark}$  also inherit the value of the previous  $\text{\botmark}$ . Thus, if each heading command would internally issue a  $\text{\mark}$  with the heading text as its argument, then one could display the first or last heading text on a page in the running header or footer by using these commands.

*Low level  $\text{\TeX}$  marks cannot be used in  $\text{\LaTeX}$*

However, it is *not* possible to use these commands directly in  $\text{\LaTeX}$ , as  $\text{\LaTeX}$  uses a higher-level protocol to control marks, so please do not try this. We mention them here only to explain the underlying general mechanism.  $\text{\LaTeX}$  effectively structures the content of the  $\text{\mark}$  argument so that the direct use of this command will most likely result in strange error messages.

As a replacement for the  $\text{\mark}$  command, standard  $\text{\LaTeX}$  offers the following two commands to generate marks:

$\text{\markboth}{\textit{main-mark}}{\textit{sub-mark}}$      $\text{\markright}{\textit{sub-mark}}$

The first command sets a pair of marker texts at the current point in the document. The second command also internally generates a pair of markers, but it changes only the *sub-mark* one, inheriting the *main-mark* text from a previous  $\text{\markboth}$ .

The original intention behind these commands was to provide somewhat independent marks—for example, chapter headings as *main-marks* and section headings as *sub-marks*. However, the choice of the command name  $\text{\markright}$  already indicates that Leslie Lamport had a specific marking scheme in mind when he designed those commands, which will become even more apparent when we look at the commands to retrieve the marker values in the output routine.

In the output routine  $\text{\leftmark}$  contains the *main-mark* argument of the last  $\text{\markboth}$  command before the end of the page. The  $\text{\rightmark}$  command contains the *sub-mark* argument of the first  $\text{\markright}$  or  $\text{\markboth}$  on the page, if one exists; otherwise, it contains the one most recently defined.

The marking commands work reasonably well for right markers “numbered within” left markers—hence the names (for example, when the left marker is changed by a  $\text{\chapter}$  command and the right marker is changed by a  $\text{\section}$  command). However, it produces somewhat anomalous results if a  $\text{\markboth}$  command is preceded by some other mark command on the same page—see the pages receiving L2 R1.1 and L5 R3.2 in Figure 4.2 on the next page. This figure shows schematically which left and right markers are generated for the pages being shipped out. For some type of running headers it would be better to display the first *main-mark* or the last *sub-mark*. For this purpose you could enlist the help of the *extramarks* package described below, as standard  $\text{\LaTeX}$  does not offer this possibility. Also notice that there is no way to set a *main-mark* without setting (and thus overwriting) the *sub-mark*.

In layouts that use running headers generated from heading texts it would be nice if these markers are automatically generated from the corresponding heading

<i>galley material</i>	<i>marker pair</i>	<i>retrieved markers</i>	
		\leftmark	\rightmark
\markboth{L1}{} \newpage% ----page break ----	{L1}{} {L1}{R1.1}	L1	
\markright{R1.1} \markboth{L2}{} \markright{R2.1}	{L2}{} {L2}{R2.1}		R1.1
\newpage% ----page break --- \markright{R2.2} \markright{R2.3} \markright{R2.4}	{L2}{R2.2} {L2}{R2.3} {L2}{R2.4}	L2	R2.2
\newpage% ----page break ---- \markboth{L3}{} \markright{R3.1}	{L3}{} {L3}{R3.1}	L3	R3.1
\newpage% ----page break ---- \newpage% ----page break ---- \markright{R3.2} \markboth{L4}{} \markboth{L5}{} \newpage% ----page break ----	{L3}{R3.2} {L4}{} {L5}{} {L5}{R5.1}	L5	R3.2
\markright{R5.1} \end{document}		L5	R5.1

Figure 4.2: Schematic overview of how L<sup>A</sup>T<sub>E</sub>X's marker mechanism works

commands. Fortunately, there exists an interface that allows us to define which heading commands produce markers and what text is passed to the mark. This scheme works as follows: all standard heading commands internally invoke a command `\namemark`, where *name* is the name of the heading command (e.g., `\chaptermark`, `\sectionmark`). These commands have one argument in which they receive the heading text or its short form from the optional argument of the heading command.

By default, they all do nothing. If redefined appropriately, however, they can produce a marker pair as needed by L<sup>A</sup>T<sub>E</sub>X. For instance, in the `book` class these commands are defined (approximately) as follows:

```
\renewcommand\chaptermark[1]{\markboth{\chaptername\ \thechapter. #1}{}}
\renewcommand\sectionmark[1]{\markright{\thesection. #1}}
```

In the case of a chapter, the word “Chapter” (or its equivalent in a given language; see Table 9.2 on page 547 in Section 9.1.3) followed by the sequence number of the chapter (stored in the counter `chapter`) and the contents of (a short version of) the chapter title will be placed in the *main-mark* argument of `\markboth`; at the same time the *sub-mark* will be cleared. For a section, the section number (stored in the counter `section`) followed by the contents of (a short version of)

the section title will be passed to `\markright`, which generates a marker pair with a new *sub-mark*.

#### 4.3.5 extramarks—Providing new marks

As we have seen so far, L<sup>A</sup>T<sub>E</sub>X's mark mechanism was built with a certain layout in mind and is, therefore, only partially usable for other applications. As a result a number of attempts have been made to extend or replace it with code that supports more complex marking mechanisms.

Part of the limitation is inherent in T<sub>E</sub>X itself, which provides only one type of marks and thus makes different independent marks difficult (though not impossible) to implement. This issue is resolved in eT<sub>E</sub>X, which provides independent mark classes. However, since this program is not yet in widespread use, there are no packages available that explore the new possibilities offered by the extension of the marking mechanism.

An extended mechanism within the main L<sup>A</sup>T<sub>E</sub>X model is provided by the *extramarks* package written by Piet van Oostrum (distributed as part of *fancyhdr*). It offers two additional (partially) independent marks, as well as further control over L<sup>A</sup>T<sub>E</sub>X standard marks by allowing one to retrieve the first or the last mark on a page for both *main-mark* and *sub-mark*.

To refer to the first or last *main-mark* on a given page, the package offers the commands `\firstleftmark` and `\lastleftmark`, respectively. Similarly, `\firstrightmark` and `\lastrightmark` allow you to access the first or last *sub-mark*.<sup>1</sup> An application is shown in Example 4-4-9 on page 229.

```
\extramarks{left-xmark}{right-xmark}
```

To add additional marks to the document the package provides the command `\extramarks`. It takes two mandatory arguments: the texts for two marks at the current point. To refer to the first *left-xmark* on a page `\firstleftxmark` is used; `\lastleftxmark` retrieves the last mark. In the same way `\firstrightxmark` and `\lastrightxmark` can be used in the output routine to access the *right-xmark*.

The next example shows these commands in action. With the help of *fancyhdr* (described in Section 4.4.2), a page layout is constructed in which the first *left-xmark* is shown at the top of a page and the last *right-xmark* is displayed at the bottom right of each page. Of particular interest in the example is the use of the `\extramarks`. We start with an `\extramarks` that contains “A story” in *left-xmark* and an empty *right-xmark*. It is immediately followed by a second set of marks, this time with the values “...continued” and “turn page to continue”. As a result the first *left-xmark* on the first page will contain “A story” while on later pages it will contain “...continued”. The last *right-xmark* on each page will always contain “turn page to continue”. Thus, as long as our story continues, we will get proper

<sup>1</sup>As the reader will notice, `\lastleftmark` and `\firstrightmark` are simply aliases for L<sup>A</sup>T<sub>E</sub>X's `\leftmark` and `\rightmark`, with names providing a clearer indication of their functionalities.

continuation marks on the top and the bottom of each page. However, at the end of the story, there should be no “turn page to continue”. To cancel that bottom mark, the example contains another `\extramarks` at the very end with an empty `right-xmark`. Its `left-xmark` still contains “...continued” to ensure that the last page displays the correct text at the top.

A story	...continued	<pre>\usepackage{fancyhdr,extramarks} \pagestyle{fancy}           \cfoot{} \lhead{\firstleftxmark} \rfoot{\lastrightxmark} \newcommand\sample{ Some text for our                   page that is reused over and over again.} \extramarks{A story}{} \extramarks{\ldots continued}{}                            {turn page to continue} \sample \sample \sample \extramarks{\ldots continued}{}</pre>
4-3-3	turn page to continue	

The extra marks can be mixed with  $\text{\LaTeX}$  standard marks produced by the sectioning commands or through `\markboth` and `\markright`. Note, however, that the marks are not fully independent of each other: whenever `\extramarks` or one of the standard  $\text{\LaTeX}$  mark commands is issued,  $\text{\LaTeX}$  effectively generates all four marks (reusing the values for those not explicitly set). As a result the first mark of a particular kind may not be what you expect. For example, if your document starts with an `\extramarks` command, it implicitly generates an empty `main-mark` and `sub-mark`.

A third type of primitive, `\topmark`, is also present in the mark model of  $\text{\TeX}$ , which is normally not made available by  $\text{\LaTeX}$ . It holds the value of the `\botmark` from the previous page, reflecting the “mark situation” at the very top of the page—hence its name. The reason that it is not made available by standard  $\text{\LaTeX}$  is that it conflicts with  $\text{\LaTeX}$ ’s float and `\marginpar` mechanism. In other words, each such object internally triggers the output routine, with the result that the `\topmark` value for the current page is clobbered.

If, however, neither floats nor `\marginpars` are used, the `\topmark` information could be used, and for such situations `extramarks` offers an interface to it. People, who have an application for such a top mark can, therefore, access the `left-xmark` and `right-xmark` produced via `\extramarks` with the commands `\topleftxmark` and `\toprightxmark`, respectively.

## 4.4 Page styles

While the dimensions remain the same for almost all pages of a document, the format of the running headers and footers may change in the course of a document. In  $\text{\LaTeX}$  terminology the formatting of running headers and footers is called

a *page style*, with different formatings being given names like `empty` or `plain` to be easily selectable.

New page styles can be selected by using the command `\pagestyle` or the command `\thispagestyle`, both of which take the name of a page style as their mandatory argument. The first command sets the page style of the current and succeeding pages; the second applies to the current page only.

In small or medium-size documents sophisticated switching of page styles is normally not necessary. Instead, one can usually rely on the page styles automatically selected by the document class. For larger documents, such as books, typographic tradition, publisher requirements, or other reasons might force you to manually adjust the page style at certain places within the document.

*L<sup>A</sup>T<sub>E</sub>X's standard page styles* L<sup>A</sup>T<sub>E</sub>X predefines four basic page styles, but additional ones might be provided by special packages or document classes.

`empty` Both the header and the footer are empty.

`plain` The header is empty and the footer contains the page number.

`headings` The header contains information determined by the document class and the page number; the footer is empty.

`myheadings` Similar to `headings`, but the header can be controlled by the user.

The first three page styles are used in the standard classes. Usually for the title page, a command `\thispagestyle{empty}` is issued internally. For the first page of major sectioning commands (like `\part` or `\chapter`, but also `\maketitle`), the standard L<sup>A</sup>T<sub>E</sub>X class files issue a `\thispagestyle{plain}` command. This means that when you specify a `\pagestyle{empty}` command at the beginning of your document, you will still get page numbers on a page where a `\chapter` or `\maketitle` command is issued. Thus, to prohibit page numbers on all pages of your document, you must follow each such command with a `\thispagestyle{empty}` command or redefine the `plain` style to `empty`, by using `\let\ps@plain=\ps@empty` in your private customization package.

In the `headings` page style the sectioning commands set the page headers automatically by using `\markboth` and `\markright`, as shown in Table 4.3 on the facing page.

The standard page style `myheadings` is similar to `headings`, but it allows the user to customize a header by manually using the commands `\markboth` and `\markright`. It also provides a way to control the capture of titles from other sectional units like a table of contents, a list of figures, or an index. In fact, the commands (`\tableofcontents`, `\listoffigures`, and `\listoftables`) and the environments (`thebibliography` and `theindex`) use the `\chapter*` command, which does not invoke `\chaptermark`, but rather issues a `\@mkboth` command. The page style `headings` defines `\@mkboth` as `\markboth`, while the page style `myheadings` defines `\@mkboth` to do nothing and leaves the decision to the user.

	<i>Command</i>	<i>Document Class</i>	
		<i>book, report</i>	<i>article</i>
<i>Two-sided Printing</i>	\markboth <sup>a</sup>	\chapter	\section
	\markright	\section	\subsection
<i>One-sided Printing</i>	\markright	\chapter	\section

<sup>a</sup>Specifies an empty right marker (see Figure 4.2 on page 219).

Table 4.3: Page style defining commands in L<sup>A</sup>T<sub>E</sub>X

#### 4.4.1 The low-level page style interface

Internally, the page style interface is implemented by the L<sup>A</sup>T<sub>E</sub>X kernel through four internal commands, of which two are called on any one page in order to format the running headers and footers. By redefining these commands different actions can be carried out.

\@oddhead For two-sided printing, it generates the header for the odd-numbered pages; otherwise, it generates the header for all pages.

\@oddfoot For two-sided printing, it generates the footer for the odd-numbered pages; otherwise, it generates the footer for all pages.

\@evenhead For two-sided printing, it generates the header of the even-numbered pages; it is ignored in one-sided printing.

\@evenfoot For two-sided printing, it generates the footer of the even-numbered pages; it is ignored in one-sided printing.

A named page style simply consists of suitable redefinitions for these commands stored in a macro with the name \ps@*style*; thus, to define the behavior of the page style *style*, one has to (re)define this command. As an example, the kernel definition of the plain page style, producing only a centered page number in the footer, is similar to the following code:

```
\newcommand\ps@plain{%
  \renewcommand\@oddhead{}%                                % empty recto header
  \let\@evenhead\@oddhead                               % empty verso header
  \renewcommand\@evenfoot
    {\hfil\normalfont\textrm{\thepage}\hfil}%           % centered
  \let\@oddfoot\@evenfoot                               % page number
}
```

#### 4.4.2 fancyhdr—Customizing page styles

Given that the page styles of standard L<sup>A</sup>T<sub>E</sub>X allow modification only via internal commands, it is not surprising that a number of packages have appeared that provide special page layouts—for example, `rplain` changes the `plain` page style so that the page number prints on the right instead of being centered. More elaborate packages exist as well. For example, the page style declaration features of the package `titlesec` (for defining heading commands, see Section 2.2.6) are worth exploring.

A well-established stand-alone package in this area is `fancyhdr`<sup>1</sup> by Piet van Oostrum, which allows easy customization of page headers and footers. The default page style provided by `fancyhdr` is named `fancy`. It should be activated via `\pagestyle` after any changes to `\textwidth` are made, as `fancyhdr` initializes the header and footer widths using the current value of this length.

The look and feel of the `fancy` page style is determined by six declarations that define the material that will appear on the left, center, and right of the header and footer areas. For example, `\lhead` specifies what should show up on the left in the header area, while `\cfoot` defines what will appear in the center of the footer area. The results of all six declarations are shown in the next example.

LEFT	CENTER	RIGHT	
Some text for our page that might get reused over and over again.			<code>\usepackage{fancyhdr} \pagestyle{fancy}</code> <code>\lhead{LEFT} \chead{CENTER} \rhead{RIGHT}</code> <code>\lfoot{very-very-very-long-left} \cfoot{}</code> <code>\rfoot{very-long-right}</code> <code>\renewcommand\headrulewidth{2pt}</code> <code>\renewcommand\footrulewidth{0.4pt}</code> <code>\newcommand\sample{ Some text for our page</code> <code>that might get reused over and over again. }</code> <code>\sample \par \sample</code>
Some text for our page that might get reused over and over again.			
very-very-very-long-left long-right			4-4-1

In many cases only one part of the footer and header areas receives material for typesetting. If you give more than one declaration with a non-empty argument, however, you have to ensure that the printed text does not get too wide. Otherwise, as the above example clearly shows, you will get partial overprints.

The thickness of the rules below the header and above the footer is controlled by the commands `\headrulewidth` (default 0.4pt) and `\footrulewidth` (default 0pt). A thickness of 0pt makes a rule invisible. Note that both are commands, not length parameters, and thus need changing via `\renewcommand`. More complicated changes are possible by redefining the `\headrule` and/or `\footrule` commands that produce the actual rules, as demonstrated in Example 4-4-6 on page 227. If you redefine these commands you may have to add negative vertical spaces be-

<sup>1</sup>In this book we describe version 2.0 of `fancyhdr`. Earlier versions were known under the name `fancyheadings`.

cause by default your material will appear at a distance of `\baselineskip` below the header text (or above the footer text).

Shown in the next example is the possibility of producing several lines of text in the running header or footer by using `\\"` in any of the declaration commands. If you take this tack, you usually have to enlarge `\headheight` (the height of the running header or footer box) because it is typically set to a value suitable only for holding a single line. If `fancyhdr` detects that `\headheight` is too small, it will issue a warning suggesting the smallest possible value that would be sufficient for the current document.

From: Frank                          Page: 6  
 To: Michel ..... February 29, 2004

Some text for our page that might  
 get reused over and over again.

Some text for our page that might  
 get reused over and over again.

```
\usepackage{fancyhdr} \pagestyle{fancy}
\setlength\headheight{23pt}
\lhead[From: Frank\\ To: Michel]
\rhead[Page: \thepage\\ \today]
\chead{} \lfoot{} \cfoot{} \rfoot{}
\renewcommand\headrule{\vspace{-8pt}\dotfill}
% \sample defined as before
\sample \par \sample
```

Notice in the previous example that the use of `\\"` will result in stacked lines that are aligned according to the type of declaration in which they appear. For example, inside `\lhead` they align on the left and inside `\rhead` they align on the right. If this outcome is not what you want, consider using a simple `tabular` environment instead. Note the `@{}` in the column declaration for the tabular material, which acts to suppress the standard white space after the column. Without it the header material would not align properly at the border.

From: Frank                          Page: 6  
 To: Michel ..... February 29, 2004

Some text for our page that might  
 get reused over and over again.

Some text for our page that might  
 get reused over and over again.

```
\usepackage{fancyhdr} \pagestyle{fancy}
\setlength\headheight{23pt}
\lhead[From: Frank\\ To: Michel]
\rhead[\begin{tabular}[b]{@{}l@{}} Page: \thepage\\ \today \end{tabular}]
\chead{} \lfoot{} \cfoot{} \rfoot{}
% \sample defined as before
\sample \par \sample
```

The declarations we have seen so far do not allow you to change the page style depending on the type of the current page. This flexibility is offered by the more general declarations `\fancyhead` and `\fancyfoot`. They take an additional optional argument in which you specify to which type of page and to which field of the header/footer the declaration should apply. Page selectors are O or E denoting odd or even pages, respectively; the fields are selected with L, C, or R. If the page or field selector is missing the declaration applies to all page types or all fields.

*Full control*

Thus, L0 means the left field on odd pages, while C would denote the center field on all pages. In other words, the declarations discussed earlier are shorthands for the more general form.

As the next example shows the selectors can even be sequenced. For example, R0,LE means apply this in the right field on odd pages and the left field on even pages.

6	Memo	7
Some text for our page that might get reused over and over again.	Some text for our page that might get reused over and over again.	
Author: Frank	Author: Frank	

```
\usepackage{fancyhdr}\pagestyle{fancy}
\fancyhead{} % clear header fields
\fancyhead[R0,LE]{\thepage}
\fancyhead[L0,RE]{Memo}
\fancyfoot{} % clear footer fields
\fancyfoot[L]{Author: Frank}
\renewcommand\headrulewidth{0.4pt}
\renewcommand\footrulewidth{0.4pt}
% \sample defined as before
\sample \par \sample
```

4-4-4

In fact, `\fancyhead` and `\fancyfoot` are derived from an even more general declaration, `\fancyhf`. It has an identical syntax but supports one additional specifier type. In its optional argument you can use H or F to denote header or footer fields. Thus, `\fancyfoot[LE]` and `\fancyhf[FLE]` are equivalent, though the latter is perhaps less readable, which is why we stick with the former forms. The `\fancyhf` declaration is only an advantage if you want to clear all fields.

The next example shows an application of the `lastpage` package: in the footer we display the current and the total number of pages.

1 A TEST	1 A TEST
<b>1 A test</b>	Some text for our page that might get reused over and over again.
Some text for our page that might get reused over and over again.	Page 7 of 7

```
\usepackage{fancyhdr,lastpage}
\pagestyle{fancy}
\fancyhf{} % --- clear all fields
\fancyhead[R0,LE]{\leftmark}
\fancyfoot[C]{Page \thepage\
of \pageref{LastPage}}
% \sample defined as before
\section{A test}
\sample \par \sample
```

4-4-5

The headers and footers are typeset in boxes that, by default, have the same width as `\textwidth`. The boxes can be made wider (or narrower) with the help of the command `\fancyhfoffset`.<sup>1</sup> It takes an optional argument to denote which box (header or footer) should be modified, at which side (left or right), and on what kind of page (even or odd)—the specification employs a combination

<sup>1</sup>This feature was added in version 2.1. Earlier releases used a different method.

of the letters HFLREO for this purpose. The mandatory argument then specifies the amount of extension (or reduction). In the same fashion as seen for other commands there also exist two useful shorthand forms: `\fancyheadoffset` and `\fancyfootoffset` are like `\fancyhfoffset` with H or F preset.

For example, to produce a running heading that spans marginal notes, use the sum of `\marginparsep` and `\marginparwidth` in the mandatory argument of `\fancyheadoffset`. With the `calc` package this can be specified elegantly with the declaration

```
\fancyheadoffset [R0,LE]{\marginparsep+\marginparwidth}
```

once these parameters have been assigned their correct values (this technique was, for example, used for the page styles used in this book).

In the next example the heading is extended into the outer margin while the page number is centered within the bounds of the text column. This result proves that the header and footer settings are, indeed, independent.

Within the header and footer fields the total width is available in the register `\headwidth` (recalculated for header and footer independently). It can be used to position objects in the fields. Below we redefine the `\headrule` command to produce a decorative heading line consisting of two blue rules spanning the whole head width.

TITLE	1 A-HEAD		1.1 B-head	TITLE
	<b>1 A-head</b>			
4-4-6	<b>1.1 B-head</b>	6	Some text for our page that might get reused over and over again.	7

```
\usepackage{color,fancyhdr}
\pagestyle{fancy} \fancyhf{}
\fancyheadoffset [R0,LE]{30pt}
\fancyhead[R0,LE]{TITLE}
\fancyhead[LO]{\rightmark}
\fancyhead[RE]{\leftmark}
\fancyfoot[C]{\thepage}
\renewcommand\headrule
{\color{blue}%
 \hrule height 2pt
 width\headwidth
 \vspace{1pt}%
 \hrule height 1pt
 width\headwidth
 \vspace{-4pt}}}
% \sample defined as before
\section{A-head}
\subsection{B-head}
\sample \sample
```

You may have guessed one or the other default used by `fancyhdr` from the previous examples. The next example will show all of them (for ease of reference they are repeated as comments in the example code). By default, we have a thin rule below the header and no rule above the footer, the page number is centered

*The fancyhdr defaults*

in the footer, and the header displays both `\leftmark` and `\rightmark` with the order depending on the page type.

<i>1 TEST</i> <b>1 Test</b> <b>1.1 B-head</b> Some text for our page that might get reused over and over again.	<i>1 TEST 1.2 B-head2</i> <b>1.2 B-head2</b> Some text for our page that might get reused over and over again.
6	7

```
\usepackage{fancyhdr}
\pagestyle{fancy}
%\fancyhead[LE,RO]
%{\slshape\rightmark}
%\fancyhead[LO,RE]
%{\slshape\leftmark}
%\fancyfoot[C]{\thepage}
%\renewcommand\headrulewidth{0.4pt}
%\renewcommand\footrulewidth{0pt}
% \sample defined as before
\section{Test}
\subsection{B-head} \sample
\subsection{B-head2}\sample
```

4-4-7

The separation between number and text in the running header is clearly too large but this is due to our extremely small measure in the example, so let us ignore this problem for the moment. How useful are these defaults otherwise? As we already mentioned, L<sup>A</sup>T<sub>E</sub>X's `\leftmark` and `\rightmark` commands have been designed primarily with "sections within chapters" in mind—that is, for the case where the `\leftmark` is associated with a heading that always starts on a new page. If this is not the case then you might end up with somewhat strange headers as exemplified below.

We put a section on page 5 (the page is not shown) that continues onto page 6. As a result we see the subsection 1.1 together with section 2 in the header of page 6, and a similar situation on page 7.

<i>1.1 B-head 2 A-HEAD2</i> <b>1.1 B-head</b> Some text for our page that we reuse.	<i>3 A-HEAD3 2.1 B-head2</i> <b>2.1 B-head2</b> Some text for our page that we reuse.
6	7

```
\usepackage{fancyhdr}
\pagestyle{fancy}
\newcommand\sample{ Some text for our page that we reuse.}

\setcounter{page}{5}
\section{A-head} \newpage
% Above makes a section on
% page 5 (not displayed)
\subsection{B-head} \sample
\section{A-head2} \sample
\subsection{B-head2}\sample
\section{A-head3} \sample
```

4-4-8

To understand this behavior recall that `\leftmark` refers to the last mark produced by `\markboth` on that particular page, while `\rightmark` refers to the first mark produced from either `\markright` or `\markboth`.

If you are likely to produce pages like the above, such as in a document containing many short subsections, then the `fancyhdr` defaults are probably not suitable for you. In that case overwrite them in one way or another, as we did in most of the examples in this section. The question you have to ask yourself is this: what information do I want to present to the reader in such a heading? If the answer is, for example, the situation at the top of the page for even (left-hand) pages and the status on the bottom for odd pages, then a possible solution is given through the use of `\firstleftmark` and `\lastrightmark` from the `extramarks` package.

<p><i>1.1 B-head    1 A-HEAD</i></p> <p><b>1.1 B-head</b> Some text for our page that we reuse.</p> <p><b>2 A-head2</b> Some text for our page that we reuse.</p>	<p><i>3 A-HEAD3</i></p> <p><b>2.1 B-head2</b> Some text for our page that we reuse.</p> <p><b>3 A-head3</b> Some text for our page that we reuse.</p>	<pre>\usepackage{extramarks} \usepackage{fancyhdr} \pagestyle{fancy} \fancyhead[RO]{\lastrightmark} \fancyhead[RE]{\firstleftmark} % \sample defined as before \setcounter{page}{5} \section{A-head} \newpage % Above makes a section on % page 5 (not displayed) \subsection{B-head} \sample \section{A-head2} \sample \subsection{B-head2}\sample \section{A-head3} \sample</pre>
4-4-9	6	7

To test your understanding explain why page 7 now shows only the A-head and try to guess what headers you would get if the first B-head (but not all of its section text) had already been on page 5.

Despite the claim made earlier, there are two more defaults set by the `fancy` page style. Because they are somewhat hidden we have ignored them until now. We have not said how `\leftmark` and `\rightmark` receive their values; that they receive some data should be clear from the previous examples. As explained in Section 4.3.4 the sectioning commands pass their title argument to commands like `\sectionmark`, which may or may not be set up to produce page marks via `\markboth` or `\markright`. The fancy page style now sets up two such commands: `\chaptermark` and `\sectionmark` if the current class defines a `\chapter` command, or `\sectionmark` and `\subsectionmark` if it does not. Thus, if you want to provide a different marking mechanism or even if you just want to provide a somewhat different layout (for example, suppressing section numbers in the heading or not using `\MakeUppercase` for the mark text), you may have to define these commands yourself.

The next example repeats Example 4-4-7 on the preceding page, except that this time we provide our own `\sectionmark` and `\subsectionmark` that shorten

the separation between number and text and avoid using `\MakeUppercase`.

<i>1 Test</i> <b>1 Test</b> <b>1.1 B-head</b> Some text for our page that might get reused over and over again. 6	<i>1 Test</i> <b>1.2 B-head2</b> Some text for our page that might get reused over and over again. 7	<pre>\usepackage{fancyhdr} \pagestyle{fancy} \renewcommand\sectionmark[1]   {\markboth{\thesection\ #1}{}}% \renewcommand\subsectionmark[1]   {\markright{\thesubsection\ #1}}% % \sample defined as before \section{Test} \subsection{B-head} \sample \subsection{B-head2}\sample</pre> <span style="float: right;">4-4-10</span>
---	---	--

So far, all of our examples have customized the fancy page style over and over again. However, the `fancyhdr` package also allows you to save your customizations under a name that can then be selected through the `\pagestyle` or `\thispagestyle` command. This is done with a `\fancypagestyle` declaration. It takes two arguments: the name of the page style and the customizations that should be applied when the page style is later called. Fields not set (or cleared) as well as the rule width settings are inherited from the `fancyhdr` defaults. This explains why we first use `\fancyhf` to clear all fields.

6                  Memo Some text for our page that might get reused over and over again. March 15, 2004	Memo              7 Some text for our page that might get reused over and over again. March 15, 2004	<pre>\usepackage{fancyhdr} \fancypagestyle{memo}{\fancyhf{}%   \fancyhead[R0,LE]{\thepage}%   \fancyhead[L0,RE]{Memo}%   \fancyfoot[R]{\scriptsize\today}%   \renewcommand\headrulewidth{1pt}}% \pagestyle{memo} % \sample defined as before \sample \par \sample</pre> <span style="float: right;">4-4-11</span>
--	--	---

Some `LATEX` commands, like `\chapter` and `\maketitle`, use `\thispagestyle` to automatically switch to the `plain` page style, thereby overriding the page style currently in effect. To customize page styles for such pages you can either modify the definitions of these commands (which could be painful) or change the meaning of the `plain` page style by providing a new definition with `\fancypagestyle`. This is, strictly speaking, not really the right approach—just assume that your new `plain` page style is now doing something fancy. But the fault really lies with `LATEX`'s standard classes,<sup>1</sup> which failed to use specially named page styles for these cases and instead directly referred to the most likely candidate. In practice, such

<sup>1</sup>The KOMA-Script classes, for example, use commands like `\chapterpagestyle` to refer to such special page styles, thus allowing easy customization.

a redefinition usually works very well for documents that need a fancy page style for most pages.

Sometimes it is desirable to modify the page style depending on the floating objects found on the current page. For this purpose `fancyhdr` provides a number of control commands. They can be applied in the page style declarations, thereby allowing the page style to react to the presence or absence of footnotes on the current page (`\iffootnote`), floats in the top area (`\iftopfloat`), or floats in the bottom area (`\ifbottomfloat`). Each takes two arguments: the first to typeset when the condition is satisfied, the second to execute otherwise.

In the next example we omit the head rule if there are top floats by redefining `\headrulewidth`. We also show the use of different heading texts on pages with or without top floats.

<b>SPECIAL</b> <div style="border: 1px solid black; padding: 2px; display: inline-block;">Sample t-figure</div> Some text for our page that might get reused over and over again. Some text	<b>NORMAL</b> for our page that might get reused over and over again.
4-4-12	7

```

• \usepackage{fancyhdr}
\pagestyle{fancy} \fancyhf{}
\chead{\iftopfloat{SPECIAL}{NORMAL}}
\cfoot{\thepage}
\renewcommand\headrulewidth
  {\iftopfloat{0pt}{0.4pt} }

% \sample defined as before
\sample
\begin{figure}[t]
  \centering
  \fbox{Sample t-figure}
\end{figure}
\sample

```

A similar control, `\iffloatpage`, is available to customize page styles for pages consisting only of floats—for example, to suppress running headers on such pages. If the page style is supposed to depend on several variables the controls can be nested, though that soon gets a little muddled. For example, to suppress head rules on all pages that contain either top or page floats, one would have to define `\headrulewidth` as follows:

```

\renewcommand\headrulewidth
  {\iftopfloat{0pt}{\iffloatpage{0pt}{0.4pt}}}

```

In dictionaries and similar works the running header often shows the first and the last word explained on a page to allow easy access to the dictionary data. By defining a suitable command that emits a mark for each dictionary item, such a scheme can be easily implemented. In the example below we use L<sup>A</sup>T<sub>E</sub>X's *right-mark* to store such marks, retrieving them via `\firstrightmark` and `\lastrightmark` from the `extramarks` package. On pages devoted to only a single entry, we collapse the entry by testing whether both commands contain the same value via

*Page styles  
depending on float  
objects*

*Layout for float  
pages*

*Dictionary type  
headers*

commands from the `ifthen` package. With a similar mechanism we prepared the the running headers of the index for this book.

galley—mark
<b>galley</b> Text formatted but not cut into pages.
<b>OR</b> Output routine.
<b>mark</b> An object in the galley used to communicate with the

6

running header
OR.

**running header** page title changing with page contents.

7

```
\usepackage{ifthen,fancyhdr,extramarks}
\pagestyle{fancy} \fancyhf{}
\newcommand\combinemarks{\ifthenelse
  {\equal{\firstrightmark}{\lastrightmark}}%
  {\firstrightmark}% equal values
  {\firstrightmark---\lastrightmark}}
\chead{\combinemarks} \cfoot{\thepage}
\newcommand\idxitem[1]{\par\vspace{8pt}%
  \textbf{\#1}\markright{\#1}\quad\ignorespaces}
\idxitem{galley} Text formatted but not
cut into pages.
\idxitem{OR} Output routine.
\idxitem{mark} An object in the galley
used to communicate with the OR.
\idxitem{running header} page title
changing with page contents.
```

[4-4-13]

Dictionaries are often typeset in two or more columns per page. Unfortunately, L<sup>A</sup>T<sub>E</sub>X's standard `twocolumn` mode is defective with respect to marks—the `\leftmark` always reflects the mark situation of the second column instead of containing the first mark from the first column. If this poses a problem use the reimplementation provided in the package `fixltx2e`. Alternatively, you can use the `multicol` package which also handles marks properly.

#### 4.4.3 `truncate`—Truncate text to a given length

A potential problem when producing running headers or footers is the restricted space available: if the text is too long it will simply overprint. To help in this and similar situations you can deploy the package `truncate` written by Donald Arseneau. It provides a command to truncate a given text to a given width.

```
\truncate[marker]{width}{text}
```

If the argument `text` is too wide to fit the specified `width`, it will be truncated and a continuation `marker` placed at the end. If the optional `marker` argument is missing, a default marker stored in `\TruncateMarker` is used (its value, as provided by the package, is `\dots`).

By default, truncation is done at word boundaries and only if the words are not connected via an unbreakable space specified with a `\~`. For this reason the following example truncates the text after the word `has`. It also illustrates the use of a `marker` that requires an extra set of braces to hide the brackets that are

supposed to appear as part of the text. To help you visualize the space occupied by the truncated text, | characters have been added to the left and right.

4-4-14

```
\usepackage{truncate}
|This text has been~truncated|
|\truncate{50pt}
  {This text has been~truncated}|
|This text has been truncated|
|This text... |
|This text has[..] |
|\truncate[{\,,[.]}]{100pt}
  {This text has been~truncated}|
```

Truncation within words can be achieved by specifying one of the options `hyphenate`, `breakwords`, or `breakall` to the package. The first two support truncation at hyphenation points, with the difference being that `breakwords` suppresses the hyphen character (the more common solution). The third option allows truncation anywhere within words. With these options the above example would have the following result:

This text has been trun-[..]	( <code>hyphenate</code> )
This text has been trun[..]	( <code>breakwords</code> )
This text has been trunc[..]	( <code>breakall</code> )

By default, the text (whether truncated or not) is printed flush left in a box of the specified `width`. Using the package option `fit` causes the printed text to have its natural width, up to a maximum of the specified `width`.

The next example combines the `truncate` package with `fancyhdr`. Notice the use of the `fit` option. Without it the header would always be flush left (the `\headwidth` was slightly reduced to better show its effect).

4-4-15

1 SECTION WITH...

## 1 Section with a long title

Some text for our page that might get reused over and over again.

6

1 SECTION WITH...

Some text for our page that might get reused over and over again.

7

```
\usepackage[fit]{truncate}
\usepackage{fancyhdr}
\pagestyle{fancy}
\fancyhf{} % --- clear all fields
\fancyhead[R,O,LE]{\truncate{.95\headwidth}{\leftmark}}
\fancyfoot[C]{\thepage}
% \sample defined as before
\section{Section with a long title}
\sample \par \sample
```

## 4.5 Visual formatting

The final stage of the production of an important document often needs some hand-formatting to avoid bad page breaks. For this purpose, standard L<sup>A</sup>T<sub>E</sub>X offers the `\pagebreak`, `\nopagebreak`, `\newpage`, and `\clearpage` commands as well as the `\samepage` declaration, although the latter is considered obsolete in L<sup>A</sup>T<sub>E</sub>X 2 <sub>$\varepsilon$</sub> . A `\samepage` declaration together with a suitable number of `\nobreak` commands lets you request that a certain portion of your document be kept together. Unfortunately, the results are often not satisfactory; in particular, L<sup>A</sup>T<sub>E</sub>X will never make a page larger than its nominal height (`\textheight`) but rather moves everything in the scope of the `\samepage` declaration to the next page. The L<sup>A</sup>T<sub>E</sub>X 2 <sub>$\varepsilon$</sub>  command `\enlargethispage*` described below offers an alternative approach.

It is common in book production to “run” a certain number of pages (normally double spreads) short or long to avoid bad page breaks later on. This means that the nominal height of the pages is reduced or enlarged by a certain amount—for example, a `\baselineskip`. To support this practice, L<sup>A</sup>T<sub>E</sub>X 2 <sub>$\varepsilon$</sub>  offers the command `\enlargethispage{size}`.

```
\enlargethispage{size}
```

If, for example, you want to enlarge or reduce the size of some pages by one (or more) additional lines of text, you could define

```
\newcommand\longpage[1][1]{\enlargethispage{\#1\baselineskip}}
\newcommand\shortpage[1][1]{\enlargethispage{-\#1\baselineskip}}
```

and use those commands between two paragraphs on the pages in question.<sup>1</sup> The `\enlargethispage` command enlarges the `\textheight` for the current page but otherwise does not change the formatting parameters. Thus, if `\flushbottom` is in force, the text will fill the `\textheight` for the page in question, if necessary by enlarging or shrinking vertical space within the page. In this way, the definitions add or remove exactly one line of text from a page while maintaining the positions of the other lines. This consideration is important to give a uniform appearance.

```
\enlargethispage*{size}
```

The companion command, `\enlargethispage*`, also enlarges or reduces the page height, but this time the resulting final page will be squeezed as much as possible (i.e., depending on the available white space on the page). This technique can be helpful if you wish to keep a certain portion of your document together

<sup>1</sup>Because this book contains so many examples, we had to use this trick a few times to avoid half-empty pages. For example, in this chapter all pages from 222 onward are run short by one line. This was necessary because of the many (large) examples in Section 4.4.2—all other formattings we tried ended in a half-empty page somewhere.

on one page, even if it makes the page slightly too long. (Otherwise, just use the `minipage` environment.) The trick is to request a large enough amount of extra space and then place an explicit page break where you want the page break to happen. For example:

```
\enlargethispage*{100cm}          % absurd request
\begin{center}
\begin{tabular}{l}           % slightly too long
...
\end{tabular}
\end{center}
\pagebreak                      % forced page break
```

From the description above it is clear that both commands should be used only in the last stages of the production process, since any later alterations to the document (adding or removing a single word, if you are unlucky) can make your hand-formatting obsolete—resulting in ugly-looking pages.

To manually correct final page breaks, such as in a publication like this book (which poses some formidable challenges due to the many examples that cannot be broken across pages), it can be helpful to visualize TeX's reasons for breaking at a certain point and to find out how much flexibility is available on certain pages. Tools for this purpose are described in Appendix B.3.2.

### 4.5.1 `nextpage`—Extensions to `\clearpage`

In standard L<sup>A</sup>T<sub>E</sub>X the commands `\clearpage` and `\cleardoublepage` terminate the current paragraph and page after placing all dangling floats (if necessary, by producing a number of float pages). In two-sided printing `\cleardoublepage` also makes sure that the next page is a right-hand (odd-numbered) one by adding, if necessary, an extra page with an empty text body. However, this extra page will still get a page header and footer (as specified by the currently active page style), which may not be desirable.

1 <b>1 A Test</b> <b>1.1 A subsection</b> Some text for our page. <div style="border: 1px solid black; padding: 2px; margin-top: 10px;">4-5-1</div>	2 <i>I A TEST</i> <pre>\pagestyle{headings} % right-hand page on the left in % this example due to: \setcounter{page}{1}  \section{A Test} \subsection{A subsection} Some text for our page. \cleardoublepage \section{Another Section} This would appear on page 3.</pre>
---	---

The package `nextpage` by Peter Wilson extends this concept by providing the commands `\cleartoevenpage` and `\cleartooddpage`. Both commands accept an optional argument in which you can put text that should appear on the potentially generated page. In the next example we use this ability to provide a command `\myclearpage` that writes `BLANK PAGE` on such generated pages.

1 <b>1 A Test</b> <b>1.1 A subsection</b> Some text for our page.	2 <i>1 A TEST</i> BLANK PAGE	<pre>\usepackage{nextpage}\pagestyle{headings} \newcommand{\myclearpage}{\cleartooddpage   [\vspace*{\fill} \centering   BLANK PAGE \vspace*{\fill}]} \setcounter{page}{1} %right-hand page \section{A Test} \subsection{A subsection} Some text for our page. \myclearpage \section{Another Section} This would appear on page 3.</pre>
--	------------------------------------	--

This code still results in a running header, but by now you surely know how to fix the example: just add a `\thispagestyle{empty}` to the above definition.

The `nextpage` package also provides two commands, `\movetoevenpage` and `\movetooddpage`, that offer the same functionality, except that they do not output dangling floats.

## 4.6 Doing layout with class

Page layout is normally defined by the document class, so it should come as no great surprise that the techniques and packages described in this chapter are usually applied behind the scenes (within a document class).

The standard classes use the `LATEX` parameters and interfaces directly to define the page proportions, running headers, and other elements. More recently developed classes, however, often deploy packages like `geometry` to handle certain aspects of the page layout.

In this section we introduce two such implementations. By searching through the CTAN archive you might discover additional treasures.

### 4.6.1 KOMA-Script—A drop-in replacement for article et al.

The KOMA-Script classes, developed by Markus Kohm and based on earlier work by Frank Neukam, are drop-in replacements for the standard `article/report/book` classes that emphasize rules of typography laid down by Tschichold. The `article` class, for example, becomes `scrartcl`.

Page layout in the KOMA-Script classes is implemented by deploying the `typearea` package (see Section 4.2.3), with the classes offering the package options as class options. Extended page style design is done with the package `scrpage2` (offering features similar to those provided by `fancyhdr`). Like `typearea` this package can also be used on a stand-alone basis with one of the standard classes. Layout specifications such as font control, caption layout, and so on have been extended by providing customization possibilities that allow manipulation in the preamble of a document.

Besides offering all features available in the standard classes, the KOMA-Script classes provide extra user control inside front and back matter as well as a number of other useful extensions.

The distribution is well documented. There exists both a German and an English guide explaining all features in detail. The German documentation is also available as a nicely typeset book [100], published by DANTE, the German T<sub>E</sub>X Users Group.

#### 4.6.2 memoir—Producing complex publications

The `memoir` class written by Peter Wilson was originally developed as an alternative to the standard `book` class. It incorporates many features otherwise found only as add-on packages. The current version also works as an replacement for `article` and `can`, therefore, be used for all types of publications, from small memos to complex books.

Among other features it supports an extended set of document sizes (from 9pt to 17pt), configurable sectional headings, page headers and footers, and captions. Predefined layout styles are available for all such objects and it is possible to declare new ones as needed. The class supports declarative commands for all aspects of setting the page, text, and margin sizes, including support for trimming (crop) marks. Many components of the class are also available as stand-alone packages, for those users who wish to add a certain functionality to other classes (e.g., epigraphs, caption formatting).

Like the KOMA-Script classes, the `memoir` class is accompanied by an excellent manual of nearly 200 pages, discussing all topics related to document design and showing how to resolve potential problems with `memoir`.



## CHAPTER 5

# Tabular Material

Data is often most efficiently presented in tabular form. TEX uses powerful primitives for arranging material in rows and columns. Because they implement only a low-level, formatting-oriented functionality, several macro packages have been developed that build on those primitives to provide a higher-level command language and a more user-friendly interface.

In LATEX, two types of environments for constructing tables are provided. Most commonly the `tabular` environment or its math-mode equivalent, the `array` environment, is used. However, in some circumstances the `tabbing` environment might prove useful.

Tables typically form large units of the document that must be allowed to “float” so that the document may be paginated correctly. The environments described in this chapter are principally concerned with the table layout. To achieve correct pagination they will often be used within the `table` environment described in Chapter 6. An exception is the environments for multipage tables described in Section 5.4, which should never be used in conjunction with the LATEX float mechanism. Be careful, however, not to confuse the `tabular` environment with the `table` environment. The former allows material to be aligned in columns, while the latter is a logical document element identifying its contents as belonging together and allowing the material to be floated jointly. In particular, one `table` environment can contain several `tabular` environments.

*Tables contained  
within floating  
environments*

After taking a quick look at the `tabbing` environment, this chapter describes the extensions to LATEX’s basic `tabular` and `array` environments provided by the `array` package. This package offers increased functionality, especially in terms of a more flexible positioning of paragraph material, a better control of inter-column

and inter-row spacing, and the possibility of defining new preamble specifiers. Several packages build on the primitives provided by the `array` package to provide specific extra functionality. By combining the features in these packages, you will be able to construct complex tables in a simple way. For example, the `tabularx` and `tabulary` packages provide extra column types that allow table column widths to be calculated automatically.

Standard  $\text{\LaTeX}$  tabular environments do not produce tables that may be broken over a page. We give several examples of multipage tables using the `supertabular` and `longtable` environments provided by the similarly named packages.

We then briefly look at the use of color in tables and at several packages that give finer control over rules, and the spacing around rules, in tables. Next, we discuss table entries spanning multiple rows, created via the `multirow` package, and the `dcolumn` package, which provides a mechanism for aligning columns of figures on a decimal point.

We also discuss the use of footnotes in tables. The `threeparttable` package provides a convenient mechanism to have table notes and captions combined with a tabular layout.

The final section gives some practical advice on handling nested tables and large entries spanning multiple columns.

Mathematically oriented readers should consult the chapter on advanced mathematics, especially Section 8.2 on page 468, which discusses the alignment structures for equations. Further examples of table layouts may be found in the section on the `graphics` package, Section 10.3 on page 628.

## 5.1 Standard $\text{\LaTeX}$ environments

$\text{\LaTeX}$  has two families of environments that allow material to be lined up in columns—namely, the `tabbing` environment, and the `tabular` and `array` environments. The main differences between the two kinds of environments are:

- The `tabbing` environment is not as general as the `tabular` environment. It can be typeset only as a separate paragraph, whereas a `tabular` environment can be placed anywhere in the text or inside mathematics.
- The `tabbing` environment can be broken between pages, whereas the standard `tabular` environment cannot.
- With the `tabbing` environment the user must specify the position of each tab stop explicitly. With the `tabular` environment  $\text{\LaTeX}$  can automatically determine the width of the columns.
- Multiple `tabbing` environments cannot be nested, whereas `tabular` environments can, thus allowing complex alignments to be realized.

### 5.1.1 Using the tabbing environment

This section deals with some of the lesser-known features of the tabbing environment. First, it must be realized that formatting is under the complete control of the user. Somewhat unexpectedly, when moving to a given tab stop, you will always end up at the exact horizontal position where it was defined, independently of where the current point is. As a consequence, the current point can move backward and overwrite previous text. The scope of commands in rows is usually limited to the region between tab stops.

Be aware that the usual L<sup>A</sup>T<sub>E</sub>X commands for making accents, \', \', and \=, are redefined inside the tabbing environment. The accents are available by typing \a', \a', and \a= instead. The \- command, which normally signals a possible hyphenation point, is also redefined, but this consideration is not so important because the lines in a tabbing environment are never broken.

*Alternative names  
for accent  
commands*

A style parameter \tabbingsep, used together with the \' command, allows text to be typeset at a given distance flush right from the following tab stop. Its default value is set equal to \labelsep, which in turn is usually 5pt.

There exist a few common ways to define tab stops—that is, using a line to be typeset, or explicitly specifying a skip to the next tab stop. The \kill command may be used to terminate a line that is only used to set tab stops: the line itself is not typeset. The following example demonstrates this, and demonstrates the redefinition of tab stops on the third line.

<span style="border: 1px solid black; padding: 2px;">5-1-1</span>	<pre>\begin{tabbing} First Tab Stop \= Second \= Third \= \kill one &gt; two &gt; three &gt; four      \\ one &gt; two                               \\[3mm] new tab\ \= two &gt; \a'{e}\a'{e}           \\                                 `'(accent commands)\\ one &gt; two &gt; three &gt; four      \\ \end{tabbing}</pre>																
<span style="border: 1px solid black; padding: 2px;">5-1-2</span>	<table border="0"> <tr> <td>one</td> <td>two</td> <td>three</td> <td>four</td> </tr> <tr> <td>one</td> <td>two</td> <td></td> <td></td> </tr> <tr> <td>new tab</td> <td>two</td> <td>éé</td> <td>(accent commands)</td> </tr> <tr> <td>one</td> <td>two</td> <td>three</td> <td>four</td> </tr> </table>	one	two	three	four	one	two			new tab	two	éé	(accent commands)	one	two	three	four
one	two	three	four														
one	two																
new tab	two	éé	(accent commands)														
one	two	three	four														

If you use accents within the definition of a command that may be used inside a tabbing environment you must use the \a... forms because the standard accent commands such as \' will be interpreted as tabbing commands, as shown below. You may find it more convenient to use the inputenc package and enter the accented letters directly.

<span style="border: 1px solid black; padding: 2px;">5-1-2</span>	<pre>\usepackage[latin1]{inputenc} \newcommand\acafe{caf\'e} \newcommand\bcafe{caf\`a'e} \newcommand\ccafe{caf�} \begin{tabbing} Tab one Tab two \\ 7 bit\cafe      \\ 7 bit  caf�    \\ 8 bit  caf�    \\ 8 bit  \&gt; \ccafe \end{tabbing}</pre>
---	--

An alternative is provided by the `Tabbing` package (by Jean-Pierre Drucbert), which provides a `Tabbing` environment in which the accent commands are *not* redefined. Instead, the tabbing commands are named `\TAB'`....

```
\usepackage[latin1]{inputenc} \usepackage{Tabbing}
% definitions as before
\begin{Tabbing} Tab one \TAB= Tab two \\
7 bit   café      7 bit   \TAB> \acafe \\
7 bit   café      7 bit   \TAB> \bcafe \\
8 bit   café      8 bit   \TAB> \ccafe \end{Tabbing}
```

5-1-3

The `tabbing` environment is most useful for aligning information into columns whose widths are constant and known. The following is from Table A.1 on page 855.

```
\newcommand\lenrule[1]{\makebox[#1]{%
  \rule{.4pt}{4pt}\hrulefill\rule{.4pt}{4pt}}}
\begin{tabbing}
dd\quad \= \hspace{.55\linewidth} \= \kill
pc \> Pica = 12pt    \_\_ \\
cc \> Cicero = 12dd   \_\_ \\
cm \> Centimeter = 10mm \_\_\_
\end{tabbing}
```

5-1-4

### 5.1.2 Using the `tabular` environment

In general, when tables of any degree of complexity are required, it is usually easier to consider the `tabular`-like environments defined by `LATEX`. These environments align material horizontally in rows (separated by `\backslash\backslash`) and vertically in columns (separated by `&`).

```
\begin{array}{[pos][cols]} & rows \end{array}
\begin{tabular}{[pos][cols]} & rows \end{tabular}
\begin{tabular*}{width}{[pos][cols]} rows \end{tabular*}
```

The `array` environment is essentially the math mode equivalent of the `tabular` environment. The entries of the table are set in math mode, and the default inter-column space is different (as described below), but otherwise the functionality of the two environments is identical.

The `tabular*` environment has an additional width argument that specifies the required total width of the table. `TEX` may adjust the inter-column spacing to produce a table with this width, as described below.

Table 5.1 shows the various options available in the `cols` preamble declaration of the environments in the standard `LATEX` `tabular` family. The `array` package introduced in the next section extends the list of preamble options.

<code>l</code>	Left-aligned column.
<code>c</code>	Center-aligned column.
<code>r</code>	Right-aligned column.
<code>p{width}</code>	Equivalent to <code>\parbox[t]{width}</code> .
<code> </code>	Inserts a vertical line between two columns. The distance between the two columns is unaffected.
<code>@{decl}</code>	Suppresses inter-column space and inserts <i>decl</i> instead.
<code>*{num}{opts}</code>	Equivalent to <i>num</i> copies of <i>opts</i> .

Table 5.1: The preamble options in the standard L<sup>A</sup>T<sub>E</sub>X `tabular` environment

The visual appearance of the `tabular`-like environments can be controlled by various style parameters. These parameters can be changed by using the `\setlength` or `\addtolength` commands anywhere in the document. Their scope can be general or local. In the latter case the scope should be explicitly delimited by braces or another environment.

*Style parameters*

`\arraycolsep` Half the width of the horizontal space between columns in an `array` environment (default value 5pt).

`\tabcolsep` Half the width of the horizontal space between columns in a `tabular` environment (default value 6pt).

`\arrayrulewidth` The width of the vertical rule that separates columns (if a `|` is specified in the environment preamble) and the rules created by `\hline`, `\cline`, or `\vline` (default value 0.4pt).

When using the `array` package, this width is taken into account when calculating the width of the table (standard L<sup>A</sup>T<sub>E</sub>X sets the rules in such a way that they do not affect the final width of the table).

`\doublerulesep` The width of the space between lines created by two successive `||` characters in the environment preamble, or by two successive `\hline` commands (default value 2pt).

`\arraystretch` Fraction with which the normal inter-row space is multiplied. For example, a value of 1.5 would move the rows 50% farther apart. This value is set with `\renewcommand` (default value 1.0).

## 5.2 array—Extending the tabular environments

Over the years several extensions have been made to the `tabular` environment family, as described in the *L<sup>A</sup>T<sub>E</sub>X Manual*. This section explores the added functionality of the `array` package (developed by Frank Mittelbach, with contributions

	<i>Changed Option</i>
	Inserts a vertical line. The distance between two columns will be enlarged by the width of the line, in contrast to the original definition of L <sup>A</sup> T <sub>E</sub> X.
	<i>New Options</i>
<code>m{width}</code>	Defines a column of width <i>width</i> . Every entry will be centered vertically in proportion to the rest of the line. It is somewhat like <code>\parbox{width}</code> .
<code>b{width}</code>	Coincides with <code>\parbox[b]{width}</code> .
<code>&gt;{decl}</code>	Can be used before an <code>l</code> , <code>r</code> , <code>c</code> , <code>p{...}</code> , <code>m{...}</code> , or <code>b{...}</code> option. It inserts <i>decl</i> directly in front of the entry of the column.
<code>&lt;{decl}</code>	Can be used after an <code>l</code> , <code>r</code> , <code>c</code> , <code>p{...}</code> , <code>m{...}</code> , or <code>b{...}</code> option. It inserts <i>decl</i> immediately after the entry of the column.
<code>!{decl}</code>	Can be used anywhere and corresponds with the <code> </code> option. The difference is that <i>decl</i> is inserted instead of a vertical line, so this option does not suppress the normally inserted space between columns, in contrast to <code>@{...}</code> .

Table 5.2: Additional preamble options in the array package

from David Carlisle). Many of the packages described later in the chapter build on the functionality of the `array` package so as to extend or adapt the `tabular` environment.

Table 5.2 shows the new options available in the `cols` preamble declaration of the environments in the `tabular` family.

### 5.2.1 Examples of preamble commands

If you would like to use a special font, such as `\bfseries` in a flush left column, you can write `>{\bfseries}l`. You no longer have to start every entry of the column with `\bfseries`.

A	B	C
100	10	I

```
\usepackage{array}
\begin{tabular}{|>{\large c}>{\large\bfseries l}>{\itshape c|}}
\hline A & B & C\\ \hline 100 & 10 & I \\ \hline
\end{tabular}
```

5-2-1

*Extra space between rows* Notice the use of the `\extrarowheight` declaration in the second example below. It adds a vertical space of 4pt above each row. In fact, the effect of `\extrarowheight` will be visible only if the sum of its value, added to the product `\baselineskip × \arraystretch`, is larger than the actual height of the cell or, more precisely, in the case of `p`, `m`, or `b`, the height of the *first row* of the cell.

This consideration is important for tables with horizontal lines because it is often necessary to fine-tune the distance between those lines and the contents of the table. The default value of `\extrarowheight` is 0pt.

5-2-2

A	B	C
100	<b>10</b>	I

```
\usepackage{array}
\setlength{\extrarowheight}{4pt}
\begin{tabular}{|c|c|c|}
\large c|>{\large \bfseries l|}>{\itshape c|} \\
\hline A & B & C \\ \hline 100 & 10 & I \\ \hline
\end{tabular}
```

There are few restrictions on the declarations that may be used with the `>` preamble option. Nevertheless, for technical reasons beyond the scope of this book, it is not possible to change the font encoding for the table column. For example, if the current encoding is not T1, then `>\fontencoding{T1}\selectfont` does *not* work. No error message is generated but incorrect characters may be produced at the start of each cell in the column. If a column of text requires a special encoding then the encoding command should be placed explicitly at the start of each cell in the column.

*Font encoding  
changes not  
supported in a  
>{...} argument*

The differences between the three paragraph-building options `p` (the paragraph box is aligned at the top), `m` (the paragraph box is aligned in the center), and `b` (the paragraph box is aligned at the bottom) are shown schematically in the following examples.

5-2-3

1 1 1 1	2 2 2 2	3 3 3 3
1 1 1 1	2 2 2 2	
1 1 1 1		

```
\usepackage{array}
\begin{tabular}{|p{1cm}|p{1cm}|p{1cm}|}
\hline 1 1 1 1 1 1 1 1 1 1 1 1 &
2 2 2 2 2 2 2 2 & 3 3 3 3 \\ \hline
\end{tabular}
```

5-2-4

1 1 1 1	2 2 2 2	3 3 3 3
1 1 1 1	2 2 2 2	
1 1 1 1		

```
\usepackage{array}
\begin{tabular}{|m{1cm}|m{1cm}|m{1cm}|}
\hline 1 1 1 1 1 1 1 1 1 1 1 1 &
2 2 2 2 2 2 2 2 & 3 3 3 3 \\ \hline
\end{tabular}
```

5-2-5

1 1 1 1	2 2 2 2	
1 1 1 1	2 2 2 2	
1 1 1 1	2 2 2 2	3 3 3 3

```
\usepackage{array}
\begin{tabular}{|b{1cm}|b{1cm}|b{1cm}|}
\hline 1 1 1 1 1 1 1 1 1 1 1 1 &
2 2 2 2 2 2 2 2 & 3 3 3 3 \\ \hline
\end{tabular}
```

In columns that have been generated with `p`, `m`, or `b`, the default value of `\parindent` is 0pt. It can be changed with the `\setlength` command as shown

in the next example where we indent the first column by 5mm.

```
\usepackage{array}
\begin{tabular}{|l>{\setlength{\parindent{5mm}}}{p{2cm}|p{2cm}|}
  1 2 3 4 5 6   1 2 3 4 5 6 7 8
  7 8 9 0 1 2 3 4   9 0 1 2 3 4 5 6 &
  5 6 7 8 9 0   7 8 9 0
\end{tabular}
```

5-2-6

The `<` preamble option was originally developed for the following application: `>{$}c<{$}` generates a column in math mode in a tabular environment. The use of this type of preamble in an array environment results in a column in LR mode because the additional `$`s cancel the existing `$`s.

$10^{10!}$	a big number
$10^{-999}$	a small number

```
\usepackage{array}
\setlength{\extrarowheight{4pt}}
\begin{tabular}{|>{$}l<{$}|l|} \hline
  10^{10!} & a big number \\
  10^{-999} & a small number \\ \hline
\end{tabular}
```

5-2-7

A major use of the `!` and `@` options is to add rubber length with the `\extracolsep` command so that TeX can stretch the table to the desired width in the `tabular*` environment. The use of `\extracolsep` in the array package environments is subject to two restrictions: there can be at most one `\extracolsep` command per `@` or `!` expression, and the command must be directly entered into the `@` expression, not as part of a macro definition. Thus, `\newcommand\ef{\extracolsep{\fill}}`, and then later `@{\ef}` in a tabular preamble, does not work, but `\newcolumntype{e}{@{\extracolsep{\fill}}}` could be used instead.

### Typesetting narrow columns

TeX does not hyphenate the first word in a paragraph, so very narrow cells can produce overflows. This is corrected by starting the text with `\hspace{0pt}`.

Characteristics

Char-  
acteris-  
tics

```
\fbox{\parbox{11mm}{Characteristics}}%
\hfill
\fbox{\parbox{11mm}{\hspace{0pt}Characteristics}}
```

5-2-8

When you have a narrow column, you must not only make sure that the first word can be hyphenated, but also consider that short texts are easier to typeset in ragged-right mode (without being aligned at the right margin). This result is obtained by preceding the material with a `\raggedright` command (see Section 3.1.11). This command redefines the line-breaking command `\backslash`, so we must use the command `\tabularnewline`, which is defined in the array package, as

in standard L<sup>A</sup>T<sub>E</sub>X, to be the original definition of the row-ending \\ command of the tabular or array environment. Alternatively, we could have used the \arraybackslash command after the \raggedright in the third column. This locally redefines \\ to end the table row, as shown in Example 5-2-12 on page 249.

As shown in the example below, we can now typeset material inside a tabular environment ragged right, ragged left, or centered and still have control of the line breaks. The first word is now hyphenated correctly, although in the case of the Dutch text, we helped T<sub>E</sub>X a little by choosing the possible hyphenation points ourselves.

Super-consciousness is a long word	Possibilités et espérances	Mogelijkheden en hoop
Ragged left text in column one	Centered text in column two	Ragged right text in column three

5-2-9

```
\usepackage{array}
\begin{tabular}{|c|c|c|}
\hline
Super-consciousness is a long word & Possibilit\'es et esp\'erances & Mogelijkheden en hoop \\tabularnewline
\hline
Ragged left text in column one & Centered text in column two & Ragged right text in column three \\tabularnewline
\hline
\end{tabular}
```

### Controlling the horizontal separation between columns

The default inter-column spacing is controlled by setting the length parameters \arraycolsep (for array) and \tabcolsep (for tabular). However, it is often desirable to alter the spacing between individual columns, or more commonly, before the first column and after the last column of the table.

5-2-10

one	two	three	–	four	–	five
1	2	3	–	4	–	5

```
\usepackage{array}
\begin{tabular}{c@{}c!{}c@{}c!{--}c!{}c@{}c}
one&two&three&four&five\\
1&2&3&4&5
\end{tabular}
```

In the example above, @{} has been used to remove the inter-column space between columns 1 and 2. An empty !{} has no effect, as demonstrated between columns 2 and 3. Note that a dash appears in place of the default inter-column space when specified using @{--} between columns 3 and 4, but is placed in the center of the default inter-column space when specified using !{--} between columns 4 and 5.

*Using @{} to remove space at the side of the table*

A common use of `@{}` is to remove the space equal to the value of `\tabcolsep` (for `tabular`) that, by default, appears on each side of the table, as shown in the following example.

<code>text text text text</code>	<code>\begin{flushleft} \textbf{text text text text}\\"</code>
<code>one two material following ...</code>	<code>\begin{tabular}{lr}</code>
<code>three four</code>	<code>one&amp;two\\ three&amp;four\\</code>
<code>text text text text</code>	<code>\end{tabular}\textbf{material following \ldots}\\"</code>
<code>text text text text</code>	<code>\begin{tabular}{@{}lr@{}}</code>
<code>one two material following ...</code>	<code>one&amp;two\\ three&amp;four\\</code>
<code>three four</code>	<code>\end{tabular}\textbf{material following \ldots}\\"</code>
<code>text text text text</code>	<code>\textbf{text text text} \end{flushleft}</code>

5-2-11

### 5.2.2 Defining new column specifiers

If you have a one-off column in a table, then you may use the `>` and `<` options to modify the style for that column:

`>{some declarations}c<{some more decls}`

This code, however, becomes rather verbose if you often use columns of this form. Therefore, for repetitive use of a given type of column specifier, the following command has been defined:

`\newcolumntype{col}[narg]{decl}`

Here, `col` is a one-letter specifier to identify the new type of column inside a preamble; `narg` is an optional parameter, giving the number of arguments this specifier takes; and `decl` are legal declarations. For example:

`\newcolumntype{x}{>{some declarations}c<{some more decls}}`

The newly defined `x` column specifier can then be used in the preamble arguments of all array and tabular environments in which one needs columns of this form.

Quite often you may need math mode and LR mode columns inside a tabular or array environment. Thus, you can define the following column specifiers:

```
\newcolumntype{C}{>{$}c<{$}}
\newcolumntype{L}{>{$}l<{$}}
\newcolumntype{R}{>{$}r<{$}}
```

From now on you can use `C` to get centered LR mode in an array environment, or centered math mode in a tabular environment.

The `\newcolumntype` command takes the same first optional argument as `\newcommand`, which declares the number of arguments of the column specifier being defined. However, `\newcolumntype` does not take the additional optional

argument forms of `\newcommand`; in the current implementation column specifiers may have only mandatory arguments.

Super-consciousness is a long word	Possibilités et espérances	Mogelijkheden en hoop
Ragged left text in column one	Centered text in column two	Ragged right text in column three

5-2-12

```
\usepackage{array}
\newcolumntype{P}[1]
  {>{\#1\hspace{0pt}\arraybackslash}p{14mm}|}
\begin{tabular}
  {|P{\raggedleft}P{\centering}P{\raggedright}|}
\hline
  Superconsciousness is a long word &
  Possibilit\'es et esp\'erances &
  Mogelijkheden en hoop \\ \hline
  Ragged left text in column one &
  Centered text in column two &
  Ragged right text in column three \\ \hline
\end{tabular}
```

A rather different use of the `\newcolumntype` command takes advantage of the fact that the replacement text in `\newcolumntype` may refer to more than one column. The following example shows the definition of a preamble option Z. Modifying the definition in the document preamble would change the layout of all tables in the document using this preamble option in a consistent manner.

one	two	three	<code>\usepackage{array} \newcolumntype{Z}{clr}</code>
1	2	3	<code>\begin{tabular}{Z} one&amp;two&amp;three\\1&amp;2&amp;3 \end{tabular}</code>

5-2-13

The replacement text in a `\newcolumntype` command can be any of the primitives of `array`, or any new letter defined in another `\newcolumntype` command.

Any column specification in a `tabular` environment that uses one of these newly defined column types is “expanded” to its primitive form during the first stage of table processing. This means that in some circumstances error messages generated when parsing the column specification refer to the preamble argument *after* it has been rewritten by the `\newcolumntype` system, not to the preamble entered by the user.

To display a list of all currently active `\newcolumntype` definitions on the terminal, use the `\showcols` command in the preamble.

*Debugging column type declarations*

## 5.3 Calculating column widths

As described in Appendix A.2, L<sup>A</sup>T<sub>E</sub>X has two distinct modes for setting text: LR mode, in which the text is set in a single line, and paragraph mode, in which text is broken into lines of a specified length. This distinction strongly influences the design of the L<sup>A</sup>T<sub>E</sub>X table commands. The l, c, and r column types specify table entries set in LR mode whereas p, and the array package m and b types, specify table entries set in paragraph mode.

The need to specify the width of paragraph mode entries in advance sometimes causes difficulties when setting tables. We will describe several approaches that calculate the required column widths based on the required total width of the table and/or the table contents.

### 5.3.1 Explicit calculation of column widths

The environment `tabularc` can generate a table with a given number of equal-width columns and a total width for the table equal to `\ linewidth`. This approach uses the `calc` package, discussed in Appendix A.3.1. It also uses the command `\tabularnewline`, mentioned in Section 5.2.1. The environment takes the number of columns as its argument. This number (let us call it  $x$ ) is used to calculate the actual width of each column by subtracting two  $x$  times the column separation and  $(x + 1)$  times the width of the rules from the width of the line. The remaining distance is divided by  $x$  to obtain the length of a single column. The contents of the column are centered, and hyphenation of the first word is allowed.

```
\usepackage{array,calc}      \newlength\mylen
\newenvironment{tabularc}[1]
{\setlength\mylen
 {\linewidth/(\#1)-\tabcolsep*2-\arrayrulewidth*(\#1+1)/(\#1)}%
 \par\noindent             % new paragraph, flush left start
 \begin{tabular*}{\linewidth}{%
 \begin{array}{|c|c|c|}\hline
 *{\#1}{|>{\centering\hspace{0pt}} p{\the\mylen}|}\hline
 \end{array}\hline
 \end{tabular*}\par}
\begin{tabularc}{3}
\hline
Material in column one & column two & This is column three
\tabularnewline\hline
... text omitted ...

```

5-3-1

Material in column one	column two	This is column three
Column one again	and column two	This is column three
Once more column one	column two	Last time column three

Calculating column widths in this way gives you full control over the amount of space allocated to each column. Unfortunately, it is difficult to incorporate information depending on the contents of the table into the calculation. For example, if some columns in the table use the `c` column type and so are set to their natural width, you may wish to allocate the remaining space among the columns using paragraph mode. As this width is not known until after the table has been typeset, it is not possible to calculate all widths in advance. Two packages implement different algorithms that set the table multiple times so as to allocate widths to certain columns. The first, `tabularx`, essentially tries to allocate space equally

between specified paragraph mode columns. The second, `tabulary`, tries to allocate more space to columns that contain “more data”.

### 5.3.2 `tabularx`—Automatic calculation of column widths

The package `tabularx` (by David Carlisle) implements a version of the `tabular*` environment in which the widths of certain columns are calculated automatically depending on the total width of the table. The columns whose widths are automatically calculated are denoted in the preamble by the `X` qualifier. The latter column specification will be converted to `p{some value}` once the correct column width has been calculated.

```
\usepackage{tabularx}
\newcolumntype{Y}{>{\small\raggedright\arraybackslash}X}
5-3-2 \noindent\begin{tabularx}{100mm}{|Y|Y|Y|}
... text omitted ...
```

The Two Gentlemen of Verona	The Taming of the Shrew	The Comedy of Errors
Love's Labour's Lost	A Midsummer Night's Dream	The Merchant of Venice
The Merry Wives of Windsor	Much Ado About Nothing	As You Like It
Twelfth Night	Troilus and Cressida	Measure for Measure
All's Well That Ends Well	Pericles Prince of Tyre	The Winter's Tale
Cymbeline	The Tempest	

Changing the `width` argument to specify a width of `\linewidth` will produce the following table layout:

```
\usepackage{tabularx}
\newcolumntype{Y}{>{\small\raggedright\arraybackslash}X}
5-3-3 \noindent\begin{tabularx}{\linewidth}{|Y|Y|Y|}
... text omitted ...
```

The Two Gentlemen of Verona	The Taming of the Shrew	The Comedy of Errors
Love's Labour's Lost	A Midsummer Night's Dream	The Merchant of Venice
The Merry Wives of Windsor	Much Ado About Nothing	As You Like It
Twelfth Night	Troilus and Cressida	Measure for Measure
All's Well That Ends Well	Pericles Prince of Tyre	The Winter's Tale
Cymbeline	The Tempest	

*Commands used to typeset the X columns*

By default, the X specification is turned into p{*some value*}. Such narrow columns often require a special format, which may be achieved using the > syntax. Thus, you may give a specification like >{\small}X.

Another format that is useful in narrow columns is ragged right. As noted earlier, one must use the command \tabularnewline to end the table row if the last entry in a row is being set ragged right. This specification may be saved with \newcolumntype{Y}{>{\small\raggedright}X} (perhaps additionally adding \arraybackslash to make \\ denote the end of a row again). You may then use Y as a tabularx preamble argument.

The X columns are set using a p column, which corresponds to \parbox[t]. You may want to set the columns with, for example, an m column corresponding to \parbox[c]. It is impossible to change the column type using the > syntax, so another system is provided. The command \tabularxcolumn can be defined as a macro, with one argument, which expands to the tabular preamble specification to be used for X henceforth. When the command is executed, the supplied argument determines the actual column width.

The default definition is \newcommand\tabularxcolumn[1]{p{#1}}. A possible alternative definition is

```
\renewcommand\tabularxcolumn[1]{>{\small}m{#1}}
```

*Column widths* Normally, all X columns in a single table are set to the same width. It is nevertheless possible to make tabularx set them to different widths. A preamble like the following

```
>{\setlength\hspace{.5\hspace}}X>{\setlength\hspace{1.5\hspace}}X}
```

specifies two columns; the second column will be three times as wide as the first. However, when using this method two rules should be obeyed:

- The sum of the widths of all X columns should remain unchanged. In the above example, the new widths should add up to the width of two standard X columns.
- Any \multicolumn entries that cross any X column should not be used.

Superconsciousness is a long word	Mogelijkheden en hoop
Some text in col- umn one	A somewhat longer text in column two

```
\usepackage{tabularx}  \tracingtabularx
\noindent
\begin{tabularx}{\linewidth}{%
    >{\setlength\hspace{.85\hspace}}X|%
    >{\setlength\hspace{1.15\hspace}}X|}
Superconsciousness is a long word &
Moge\ -lijk\ -heden en hoop          \\
Some text in column one               &
A somewhat longer text in column two \\
\end{tabularx}
```

If a `\tracingtabularx` declaration is made, say, in the document preamble, then all following `tabularx` environments will print information to the terminal and the log file about column widths as they repeatedly reset the tables to find the correct widths. For instance, the last example produced the following log:

*Tracing tabularx calculations*

```
Package tabularx Warning: Target width: \linewidth = 207.0pt..
(tabularx)      Table Width     Column Width     X Columns
(tabularx)      439.19998pt    207.0pt          3
(tabularx)      206.99998pt    90.90001pt       2
(tabularx) Reached target.
```

### 5.3.3 tabulary—Column widths based on content

An alternative algorithm for determining column widths is provided by the `tabulary` package (also written by David Carlisle), which defines the `tabulary` environment. It is most suitable for cases in which the column widths must be calculated based on the content of the table. This often arises when you use L<sup>A</sup>T<sub>E</sub>X to typeset documents originating as SGML/XML or HTML, which typically employ a different table model in which multiple line material does not have a prespecified width and the layout is left more to the formatter.

The `tabulary` package provides the column types shown in Table 5.3 on the next page plus those provided by the `array` package in Table 5.2 on page 244, and any other preamble options defined via `\newcolumntype`.

```
\begin{tabulary}{width}[pos]{cols} rows \end{tabulary}
```

The main feature of this package is its provision of versions of the `p` column specifier in which the width of the column is determined automatically depending on the table contents. The following example is rather artificial as the table only has one row. Nevertheless, it demonstrates that the aim of the column width allocation made by `tabulary` is to achieve equal row height. Normally, of course, the same row will not hold the largest entry of each column but in many cases of tabular material, the material in each cell of a given column has similar characteristics. In those situations the width allocation appears to provide reasonable results.

```
\usepackage{tabulary}
\setlength\tymin{10pt}
\setlength\tymax{\maxdimen}

a   b   cccc  d d d d d d d d d d d d
b   cccc  d d d d d d d d d d d d
b   cccc  d d d d d d d d d d d d
b   cccc  d d d d d d d d d d d d
                           cc       d d d d d d

\begin{tabulary}{200pt}{|C|C|C|C|} 
a & b b b &
c c c c c c c c c c c c c c c c c c c c &
d d d d d d d d d d d d d d d d d d d d d d
... text omitted ...

```

- J Justified p column set to some width to be determined
- L Flush left p column set to some width to be determined
- R Flush right p column set to some width to be determined
- C Centered p column set to some width to be determined

Table 5.3: The preamble options in the tabulary package

*Controlling the column width allocation*

The tabulary package has two length parameters, `\tymmin` and `\tymax`, which control the allocation of widths. By default, widths are allocated to each L, C, R, or J column in proportion to the natural width of the longest entry in each column. To determine this width tabulary always sets the table twice. In the first pass the data in L, C, R, and J columns is set in LR mode (similar to data in columns specified by the standard preamble options such as c). Typically, the paragraphs that are contained in these columns are set on a single line, and the length of this line is measured. The table is then typeset a second time to produce the final result, with the widths of the columns being set as if with a p preamble option and a width proportional to the natural lengths recorded on the first pass.

To stop very narrow columns from being too “squeezed” by this process, any columns that are narrower than `\tymmin` are set to their natural widths. This length may be set with `\setlength` and is arbitrarily initialized to 10pt. If you know that a column will be narrow, it may be preferable to use, say, c rather than C so that the tabulary mechanism is never invoked on that column, and the column is set to its natural width.

Similarly, one very large entry can force its column to be too wide. To prevent this problem, all columns with natural length greater than `\tymax` (as measured when the entries are set in LR mode) are set to the same width (with the proportion being taken as if the natural length was *equal* to `\tymax`). This width is initially set to twice the text width.

The table in the above example is dominated by the large entry in the fourth column. By setting `\tymmin` to 30pt we can prevent the first two columns from becoming too narrow, and by setting `\tymax` to 200pt we can limit the width of the fourth column and produce a more even spread of column widths.

a	b b b b	c c c c c c	d d d d d d d d d d	<code>\usepackage{tabulary}</code>
		c c c c c c	d d d d d d d d	<code>\setlength\tymmin{30pt}</code>
		c c c c	d d d d d d d d	<code>\setlength\tymax{200pt}</code>
			d d d d d d d d	<code>\begin{tabulary}{200pt}{ C C C C }</code>
			d d d d d d d d	... text omitted ...

5-3-6

Narrow p columns are sometimes quite challenging to set, and so you may redefine the command `\tyformat` to be any declarations made just after the

\centering or \ragged... declaration. By default, it redefines \everypar to insert a zero space at the start of every paragraph, so the first word may be hyphenated. (See Section 5.2.1 on page 246.)

Like tabularx, tabulary supports the optional alignment argument of tabular. Also because the whole environment is saved and evaluated twice, care should be taken with any L<sup>A</sup>T<sub>E</sub>X constructs that may have side effects such as writing to files.

### 5.3.4 Differences between tabular\*, tabularx, and tabulary

All three of these environments take the same arguments, with the goal of producing a table of a specified width. The main differences between them are described here:

- tabularx and tabulary modify the widths of the *columns*, whereas tabular\* modifies the widths of the inter-column *spaces*.
  - The tabular and tabular\* environments may be nested with no restrictions. However, if one tabularx or tabulary environment occurs inside another, then the inner one *must* be enclosed within { }.
  - The bodies of tabularx and tabulary environments are, in fact, the arguments to commands, so certain restrictions apply. The commands \verb and \verb\* may be used, but they may treat spaces incorrectly, and their arguments cannot contain a % or an unmatched { or }.
  - tabular\* uses a primitive capability of T<sub>E</sub>X to modify the inter-column space of an alignment. tabularx has to set the table several times as it searches for the best column widths, and is therefore much slower. tabulary always sets the table twice. For the latter two environments the fact that the body is expanded several times may break certain T<sub>E</sub>X constructs. Be especially wary of commands that write to external files, as the data may be written several times when the table is reset.
  - tabularx attempts to distribute space equally among the X columns to achieve the desired width, whereas tabulary attempts to allocate greater widths to columns with larger entries.
- \verb only partially supported*

## 5.4 Multipage tabular material

With Leslie Lamport's original implementation, a tabular environment must always fit on one page. If it becomes too large, the text will overwrite the page's bottom margin, and you will get an Overfull \vbox message.

Two package files are available to construct tables longer than one page, supertabular and longtable. They share a similar functionality, but use rather different syntax. The longtable package uses a more complicated mechanism, work-

*Multipage tables in  
multicolumn  
typesetting*

ing with TeX’s output routine to obtain optimal page breaks and to preserve the width of columns across all pages of a table. However, this mechanism may require the document to be processed several times before the correct table widths are calculated. The *supertabular* package essentially breaks the table into a sequence of page-sized tabular environments, and each page is then typeset separately. This approach does not require multiple passes and works in a larger range of circumstances. In particular, the *longtable* package does not support two-column or multicolumn mode.

#### 5.4.1 supertabular—Making multipage tabulars

```
\begin{supertabular}{cols}           rows \end{supertabular}
\begin{supertabular*}{width}{cols}   rows \end{supertabular*}
\begin{mpsupertabular}{cols}         rows \end{mpsupertabular}
\begin{mpsupertabular*}{width}{cols} rows \end{mpsupertabular*}
```

The package *supertabular* (originally created by Theo Jurriens, and revised by Johannes Braams) defines the environment *supertabular*. It uses the *tabular* environment internally, but it evaluates the amount of used space every time it encounters a `\\\` command. When this amount reaches the value of `\textheight`, the package automatically inserts an `\end{tabular}` command, starts a new page, and inserts the table head on the new page, continuing the *tabular* environment. This means that the widths of the columns, and hence the width of the complete table, can vary across pages.

Three variant environments are also defined. The *supertabular\** environment uses *tabular\** internally, and takes a mandatory *width* argument to specify the width of the table. The *mpsupertabular* and *mpsupertabular\** environments have the same syntax as *supertabular* and *supertabular\**, respectively, but wrap the table portion on each page in a *minipage* environment. This allows the use of the *\footnote* command inside the tables, with the footnote text being printed at the end of the relevant page.

Inside a *supertabular* environment new lines are defined as usual by `\\\` commands. All column definition commands can be used, including `@{...}` and `p{...}`. If the *array* package is loaded along with *supertabular*, the additional *tabular* preamble options may be used. You cannot, however, use the optional positioning arguments, like *t* and *b*, that can be specified with `\begin{tabular}` and `\begin{tabular*}`.

Several new commands are available for use with *supertabular* as described below. Each of these commands should be used before the *supertabular* environment, as they affect all following *supertabular* environments.

```
\tablehead{rows}    \tablefirsthead{rows}
```

The argument to `\tablehead` contains the rows of the table to be repeated at the top of every page. If `\tablefirsthead` is also included, the first heading will use

these rows in preference to the rows specified by `\tablehead`. The argument may contain full rows (ended by `\\"`) as well as inter-row material like `\hline`.

```
\tabletail{rows}      \tablelasttail{rows}
```

These commands specify material to be inserted at the end of each page of the table. If `\tablelasttail` is used, these rows will appear at the end of the table in preference to the rows specified by `\tabletail`.

```
\topcaption[lot caption]{caption}    \bottomcaption[lot caption]{caption}  
\tablecaption[lot caption]{caption}
```

These commands specify a caption for the `supertabular`, either at the top or at the bottom of the table. The optional argument has the same use as the optional argument in the standard `\caption` command—namely, it specifies the form of the caption to appear in the list of tables. When `\tablecaption` is used the caption will be placed at the default location, which is at the top. This default may be changed within a package or class file by using the declaration `\@topcaptionfalse`.

The format of the caption may be customized using the `caption` package, as shown in Example 5-4-4 on page 262.

```
\shrinkheight{length}
```

The `supertabular` environment maintains an estimate of the amount of space left on the current page. The `\shrinkheight` command, which must appear at the start of a table row, may be used to reduce this estimate. In this way it may be used to control the page-breaking decisions made by `supertabular`.

#### Example of the `supertabular` environment

```
\usepackage{supertabular}  
\tablecaption{The ISOGRK3 entity set}  
\tablehead  
  {\bfseries Entity\&\bfseries Unicode Name\&\bfseries Unicode\\ \hline}  
\tabletail  
  {\hline \multicolumn{3}{r}{\emph{Continued on next page}}}\\  
\tablelasttail{\hline}  
\begin{supertabular}{lll}  
alpha          & GREEK SMALL LETTER ALPHA          & & 03B1\\  
beta          & GREEK SMALL LETTER BETA          & & 03B2\\  
chi           & GREEK SMALL LETTER CHI           & & 03C7\\  
Delta          & GREEK CAPITAL LETTER DELTA        & & 0394\\  
delta          & GREEK SMALL LETTER DELTA         & & 03B4\\  
epsi           & GREEK SMALL LETTER EPSILON        & & 03B5\\  
epsis          & GREEK LUNATE EPSILON SYMBOL       & & 03F5\\  
5-4-1 ... text omitted ...
```

Page 1

Page 2

Table 1: The ISOGRK3 entity set

Entity	Unicode Name	Unicode
alpha	GREEK SMALL LETTER ALPHA	03B1
beta	GREEK SMALL LETTER BETA	03B2
chi	GREEK SMALL LETTER CHI	03C7
Delta	GREEK CAPITAL LETTER DELTA	0394
delta	GREEK SMALL LETTER DELTA	03B4
eps1	GREEK SMALL LETTER EPSILON	03B5
eps1s	GREEK LUNATE EPSILON SYMBOL	03F5
epsiv	GREEK SMALL LETTER EPSILON	03B5
eta	GREEK SMALL LETTER ETA	03B7
Gamma	GREEK CAPITAL LETTER GAMMA	0393
gamma	GREEK SMALL LETTER GAMMA	03B3
gammad	GREEK SMALL LETTER DIGAMMA	03DD
iota	GREEK SMALL LETTER IOTA	03B9
kappa	GREEK SMALL LETTER KAPPA	03BA
kappav	GREEK KAPPA SYMBOL	03F0
Lambda	GREEK CAPITAL LETTER LAMDA	039B
lambda	GREEK SMALL LETTER LAMDA	03BB
mu	GREEK SMALL LETTER MU	03BC
nu	GREEK SMALL LETTER NU	03BD
Omega	GREEK CAPITAL LETTER OMEGA	03A9
omega	GREEK SMALL LETTER OMEGA	03C9
Phi	GREEK CAPITAL LETTER PHI	03A6

*Continued on next page*

Entity	Unicode Name	Unicode
phis	GREEK PHI SYMBOL	03D5
phiv	GREEK SMALL LETTER PHI	03C6
Pi	GREEK CAPITAL LETTER PI	03A0
pi	GREEK SMALL LETTER PI	03C0
piv	GREEK PI SYMBOL	03D6
Psi	GREEK CAPITAL LETTER PSI	03A8
psi	GREEK SMALL LETTER PSI	03C8
rho	GREEK SMALL LETTER RHO	03C1
rhow	GREEK RHO SYMBOL	03F1
Sigma	GREEK CAPITAL LETTER SIGMA	03A3
sigma	GREEK SMALL LETTER SIGMA	03C3
sigmav	GREEK SMALL LETTER FINAL SIGMA	03C2
tau	GREEK SMALL LETTER TAU	03C4
Theta	GREEK CAPITAL LETTER THETA	0398
thetas	GREEK SMALL LETTER THETA	03B8
thetav	GREEK THETA SYMBOL	03D1
Upsi	GREEK UPSILON WITH HOOK SYMBOL	03D2
upsi	GREEK SMALL LETTER UPSILON	03C5
Xi	GREEK CAPITAL LETTER XI	039E
xi	GREEK SMALL LETTER XI	03B6
zeta	GREEK SMALL LETTER ZETA	03B6

Page 1

Page 2

### Example of the supertabular\* environment

The width of a *supertabular* environment can be fixed to a given width, such as the width of the text, `\textwidth`. In the example below, in addition to specifying *supertabular\**, a rubber length has been introduced between the last two columns that allows the table to be stretched to the specified width. As usual with *supertabular*, each page of the table is typeset separately. The example demonstrates that the result may have different spacings between the columns on the first (left) and second (right) page.

```
\usepackage{array,supertabular}
\tablecaption{The ISOGRK3 entity set}
\tablefirsthead
{\bfseries Entity\&\bfseries Unicode Name\&\bfseries Unicode\\ \hline}
\tablehead
{\bfseries Entity\&\bfseries Unicode Name\&\bfseries Unicode\\ \hline}
\tabletail{\hline \multicolumn{3}{r}{\emph{Continued on next page}}\\ \hline}
\tablelasttail{\hline}
\centering
\begin{supertabular*}{\textwidth}{l!{\extracolsep{\fill}}l}
alpha & GREEK SMALL LETTER ALPHA & & 03B1\\
beta & GREEK SMALL LETTER BETA & & 03B2\\
chi & GREEK SMALL LETTER CHI & & 03C7\\
... text omitted ...

```

Page 1

Page 2

Table 1· The ISOGRK3 entity set

Entity	Unicode Name	Unicode	Entity	Unicode Name	Unicode
alpha	GREEK SMALL LETTER ALPHA	03B1	Phi	GREEK CAPITAL LETTER PHI	03A6
beta	GREEK SMALL LETTER BETA	03B2	phi	GREEK PHI SYMBOL	03D5
chi	GREEK SMALL LETTER CHI	03C7	Pi	GREEK CAPITAL LETTER PI	03A0
Delta	GREEK CAPITAL LETTER DELTA	0394	pi	GREEK SMALL LETTER PI	03C0
delta	GREEK SMALL LETTER DELTA	03B4	piv	GREEK PI SYMBOL	03D6
epsi	GREEK SMALL LETTER EPSILON	03B5	Psi	GREEK CAPITAL LETTER PSI	03A8
epsis	GREEK LUNATE EPSILON SYMBOL	03F5	psi	GREEK SMALL LETTER PSI	03C8
epsiv	GREEK SMALL LETTER EPSILON	03B5	rho	GREEK SMALL LETTER RHO	03C1
eta	GREEK SMALL LETTER ETA	03B7	rhow	GREEK RHO SYMBOL	03F1
Gamma	GREEK CAPITAL LETTER GAMMA	0393	Sigma	GREEK CAPITAL LETTER SIGMA	03A3
gamma	GREEK SMALL LETTER GAMMA	03B3	sigma	GREEK SMALL LETTER SIGMA	03C3
gammad	GREEK SMALL LETTER DIGAMMA	03DD	sigmav	GREEK SMALL LETTER FINAL SIGMA	03C2
iota	GREEK SMALL LETTER IOTA	03B9	tau	GREEK SMALL LETTER TAU	03C4
kappa	GREEK SMALL LETTER KAPPA	03BA	Theta	GREEK CAPITAL LETTER THETA	0398
kappav	GREEK KAPPA SYMBOL	03F0	thetas	GREEK SMALL LETTER THETA	03B8
Lambda	GREEK CAPITAL LETTER LAMDA	039B	thetav	GREEK THETA SYMBOL	03D1
lambda	GREEK SMALL LETTER LAMDA	03BB	Upsi	GREEK UPSILON WITH HOOK SYMBOL	03D2
mu	GREEK SMALL LETTER MU	03BC	upsi	“GREEK SMALL LETTER UPSILON	03C5
nu	GREEK SMALL LETTER NU	03BD	Xi	GREEK CAPITAL LETTER XI	039E
Omega	GREEK CAPITAL LETTER OMEGA	03A9	xi	GREEK SMALL LETTER XI	03BE
omega	GREEK SMALL LETTER OMEGA	03C9	zeta	GREEK SMALL LETTER ZETA	03B6

Continued on next page

Page 1

Page 2

### 5.4.2 longtable—Alternative multipage tabulars

As pointed out at the beginning of this section, for more complex long tables, where you want to control the width of the table across page boundaries, the package `longtable` (by David Carlisle, with contributions from David Kastrup) should be considered. Like the `supertabular` environment, it shares some features with the `table` environment. In particular it uses the same counter, `table`, and has a similar `\caption` command. The `\listoftables` command lists tables produced by either the `table` or `longtable` environment.

The main difference between the `supertabular` and `longtable` environments is that the latter saves the information about the width of each `longtable` environment in the auxiliary `.aux` file. It then uses this information on a subsequent run to identify the widest column widths needed for the table in question. The use of the `.aux` file means that care should be taken when using the `longtable` in conjunction with the `\nofiles` command. One effect of `\nofiles` is to suppress the writing of the `.aux` file, so this command should not be used until after the final edits of that table have been made and the package has recorded the optimal column widths in the auxiliary file.

Use of the `.aux` file

To compare the two packages, Example 5-4-1 on page 257 is repeated here, but now uses `longtable` rather than `supertabular`. You can see that the width of the table is identical on both pages (the left and right parts of the picture). Note that in `longtable`, most of the table specification is *within* the `longtable`

environment; in `supertabular` the specification of the table headings occurs via commands executed *before* the `supertabular` environment.

```
\usepackage{longtable}
\begin{longtable}{lll}
  \caption{The ISOGRK3 entity set}\\
  \bfseries Entity&\bfseries Unicode Name&\bfseries Unicode\\ \hline
\endfirsthead
  \bfseries Entity&\bfseries Unicode Name&\bfseries Unicode\\ \hline
\endhead
  \hline \multicolumn{3}{r}{\emph{Continued on next page}}\\
\endfoot
  \hline
\endlastfoot
alpha & GREEK SMALL LETTER ALPHA & & 03B1\\
beta & GREEK SMALL LETTER BETA & & 03B2\\
chi & GREEK SMALL LETTER CHI & & 03C7\\
... text omitted ...

```

5-4-3

Page 1

Page 2

Table 1: The ISOGRK3 entity set

Entity	Unicode Name	Unicode
alpha	GREEK SMALL LETTER ALPHA	03B1
beta	GREEK SMALL LETTER BETA	03B2
chi	GREEK SMALL LETTER CHI	03C7
Delta	GREEK CAPITAL LETTER DELTA	0394
delta	GREEK SMALL LETTER DELTA	03B4
epsi	GREEK SMALL LETTER EPSILON	03B5
epsis	GREEK LUNATE EPSILON SYMBOL	03F5
epsiv	GREEK SMALL LETTER EPSILON	03B5
eta	GREEK SMALL LETTER ETA	03B7
Gamma	GREEK CAPITAL LETTER GAMMA	0393
gamma	GREEK SMALL LETTER GAMMA	03B3
gammad	GREEK SMALL LETTER DIGAMMA	03DD
iota	GREEK SMALL LETTER IOTA	03B9
kappa	GREEK SMALL LETTER KAPPA	03BA
kappav	GREEK KAPPA SYMBOL	03F0
Lambda	GREEK CAPITAL LETTER LAMDA	039B
lambda	GREEK SMALL LETTER LAMDA	03BB
mu	GREEK SMALL LETTER MU	03BC
nu	GREEK SMALL LETTER NU	03BD
Omega	GREEK CAPITAL LETTER OMEGA	03A9
omega	GREEK SMALL LETTER OMEGA	03C9
Phi	GREEK CAPITAL LETTER PHI	03A6
phis	GREEK PHI SYMBOL	03D5
phiv	GREEK SMALL LETTER PHI	03C6

*Continued on next page*

Entity	Unicode Name	Unicode
Pi	GREEK CAPITAL LETTER PI	03A0
pi	GREEK SMALL LETTER PI	03C0
piv	GREEK PI SYMBOL	03D6
Psi	GREEK CAPITAL LETTER PSI	03A8
psi	GREEK SMALL LETTER PSI	03C8
rho	GREEK SMALL LETTER RHO	03C1
rhow	GREEK RHO SYMBOL	03F1
Sigma	GREEK CAPITAL LETTER SIGMA	03A3
sigma	GREEK SMALL LETTER SIGMA	03C3
sigmav	GREEK SMALL LETTER FINAL SIGMA	03C2
tau	GREEK SMALL LETTER TAU	03C4
Theta	GREEK CAPITAL LETTER THETA	0398
thetas	GREEK SMALL LETTER THETA	03B8
thetav	GREEK THETA SYMBOL	03D1
Upsilon	GREEK UPSILON WITH HOOK SYMBOL	03D2
upsilon	GREEK SMALL LETTER UPSILON	03C5
Xi	GREEK CAPITAL LETTER XI	039E
xi	GREEK SMALL LETTER XI	03BE
zeta	GREEK SMALL LETTER ZETA	03B6

Page 1

Page 2

```
\begin{longtable}[align]{cols} rows \end{longtable}
```

The syntax of the `longtable` environment is modeled on that of the `tabular` environment. The main difference is that the optional `align` argument specifies *horizontal* alignment rather than vertical alignment as is the case with `tabular`.

The *align* argument may have the value [c], [l], or [r], to specify centering, left, or right alignment of the table, respectively. If this optional argument is omitted then the alignment of the table is controlled by the two length parameters, \LTleft and \LTright. They have default values of \fill, so by default tables will be centered.

*Horizontal alignment*

Any length can be specified for these two parameters, but at least one of them should be a rubber length so that it fills up the width of the page, unless rubber lengths are added between the columns using the \extracolsep command. For instance, a table can be set flush left using the definitions

```
\setlength\LTleft{0pt} \setlength\LTright{\fill}
```

or just by specifying \begin{longtable}[1].

You can, for example, use the \LTleft and \LTright parameters to typeset a multipage table filling the full width of the page. Example 5-4-2 on page 258, which used supertabular\*, may be typeset using the packages array and longtable and the declarations shown below:

```
\setlength\LTleft{0pt} \setlength\LTright{0pt}
\begin{longtable}{ll!{\extracolsep{\fill}}l}
```

In general, if \LTleft and \LTright are fixed lengths, the table will be set to the width of \textwidth - \LTleft - \LTright.

Before and after the table, longtable inserts vertical space controlled by the length parameters \LTpre and \LTpost. Both default to the length \bigskipamount, but may be changed using \setlength.

*Using parameters to control table width*

Each row in the table is ended with the \\ command. As in the standard tabular environment, the command \tabularnewline is also available; it is useful if \\ has been redefined by a command such as \raggedright. The star form \\\* may also be used which inhibits a page break at this linebreak. In a tabular environment, this star form is accepted but has the same effect as \\. Conversely, a \\ command may be immediately followed by a \newpage command, which forces a page break at that point.

*Vertical space around table*

If a table row is terminated with \kill rather than \\, then the row will not be typeset. Instead, the entries will be used when determining the widths of the table columns. This action is similar to that of the \kill command in the tabbing environment.

The main syntactic difference between the longtable package and the supertabular package is that in longtable, rows to be repeated on each page as the table head or foot are declared *within* the environment body, rather than before the environment as in supertabular. As shown in Example 5-4-3 on the preceding page, the table head and foot are specified by replacing the final \\ command by one of the commands listed below. Note that all of these commands, including those specifying the foot of the table, must come at the *start* of the en-

*Table row commands*

*Rows used as the table head and foot*

vironment. The command `\endhead` finishes the rows that will appear at the top of every page. The command `\endfirsthead` ends the declaration of rows for the start of the table. If this command is not used then the rows specified by `\endhead` will be used at the start of the table. Similarly, `\endfoot` finishes the rows that will appear at the bottom of every page, and `\endlastfoot`—if used—ends the rows to be displayed at the end of the table.

```
\caption*[short title]{full title}
```

The `\caption` command and its variant `\caption*` are essentially equivalent to writing a special `\multicolumn` entry

```
\multicolumn{n}{p{\LTcapwidth}}{...}
```

where *n* is the number of columns of the table. The width of the caption can be controlled by redefining the parameter `\LTcapwidth`. That is, you can write `\setlength{\LTcapwidth}{width}` in the document preamble. The default value is 4in. As with the `\caption` command in the `figure` and `table` environments, the optional argument specifies the text to appear in the list of tables if it is different from the text to appear in the caption.

When captions on later pages should differ from those on the first page, you should place the `\caption` command with the full text in the first heading, and put a subsidiary caption using `\caption[ ]` in the main heading, since (in this case) no entry is made in the list of tables. Alternatively, if the table number should not be repeated each time, you can use the `\caption*` command. As with the `table` environment, cross-referencing the table in the text is possible with the `\label` command.

By default, the caption is set in a style based on the caption style of the tables in standard L<sup>A</sup>T<sub>E</sub>X's article class. If the `caption` package (described in Section 6.5.1) is used, then it is easy to customize `longtable` and `table` captions, keeping the style of captions consistent between these two environments.

```
\usepackage{longtable,supertabular}
\usepackage[font=sl,labelfont=bf]{caption}
\begin{table}[t]\centering
\caption{A standard table}
\begin{tabular}{ccc}1&2&3\end{tabular}
\end{table}
```

**Table 1:** A standard table

1 2 3

```
\begin{longtable}{ccc}\caption{A longtable}\\
1&2&3\end{longtable}
```

**Table 2:** A `longtable`

1 2 3

```
\centering
\tablecaption{A supertabular}
\begin{supertabular}{ccc}1&2&3\\
\end{supertabular}
```

**Table 3:** A `supertabular`

1 2 3

You can use footnote commands inside the `longtable` environment. The footnote text appears at the bottom of each page. The footnote counter is not reset at the beginning of the table, but uses the standard footnote numbering employed in the rest of the document. If this result is not desired then you can set the footnote counter to zero before the start of each table, and then reset it at the end of the table if following footnotes must be numbered in the original sequence.

To enable  $\text{\TeX}$  to set very long multipage tables, it is necessary to break them up into smaller chunks so that  $\text{\TeX}$  does not have to keep everything in memory at one time. By default, `longtable` uses a value of 20 rows per chunk, which can be changed with a command such as `\setcounter{LTchunksize}{100}`. These chunks do not affect page breaking. When  $\text{\TeX}$  has a lot of memory available `LTchunksize` can be set to a big number, which usually means that `longtable` will be able to determine the final widths in fewer  $\text{\TeX}$  runs. On most modern  $\text{\TeX}$  installations `LTchunksize` can safely be increased to accommodate several pages of table in one chunk. Note that `LTchunksize` must be at least as large as the number of rows in each of the head or foot sections.

*Footnotes in  
longtable*

*Increase  
LTchunksize to  
reduce number of  
 $\text{\TeX}$  runs required*

### Problems with multipage tables

When a float occurs on the same page as the start of a multipage table, unexpected results can occur. Both packages have code that attempts to deal with this situation, but in some circumstances tables can float out of sequence. Placing a `\clearpage` command before the table, thereby forcing a page break and flushing out any floats, will usually correct the problem.

*Bad interaction  
of floating  
environments and  
multipage tables*

Neither the `supertabular` nor the `longtable` environment will make a page break after a line of text *within* a cell. Pages will be broken only between table rows (or at `\hline` commands). If your table consists of large multiple line cells set with the `p` preamble option, then  $\text{\LaTeX}$  may not be able to find a good page break and may leave unwanted white space at the bottom of the page.

*`p` column entries  
do not break*

The example below has room for six lines of text on each page but  $\text{\LaTeX}$  breaks the page between the two table rows, leaving page 1 short.

```
\usepackage{longtable}
\begin{longtable}{llp{43mm}}
entry 1.1 & entry 1.2 & entry 1.3, a long text entry taking several lines.\\
entry 2.1 & entry 2.2 & entry 2.3, a long text entry taking several lines
when set in a narrow column.
\end{longtable}
```

[5-4-5]

Page 1

entry 1.1 entry 1.2 entry 1.3, a long text  
entry taking several  
lines.

Page 2

entry 2.1 entry 2.2 entry 2.3, a long text  
entry taking several  
lines when set in a  
narrow column.

Page 1

Page 2

For some tables, the table rows form an important logical unit and the default behavior of not breaking within a row is desired. In other cases, it may be preferable to break the table manually to achieve a more pleasing page break. In the above example, we want to move the first two lines of page 2 to the bottom of page 1. Noting that  $\text{\TeX}$  broke the third column entry after the word “several”, we could end the table row at that point by using  $\backslash\backslash$ , insert blank entries in the first two columns of a new row, and place the remaining portion of the  $p$  entry in the final cell of this row. The first part of the split paragraph should be set with  $\setlength{\parfillskip}{0pt}$  so that the final line appears full width, just as it would be if it were set as the first two lines of a larger paragraph.

```
\usepackage{longtable}
\begin{longtable}{llp{43mm}}
entry 1.1 & entry 1.2 & entry 1.3, a long text entry taking several lines.\\
entry 2.1 & entry 2.2 & \setlength{\parfillskip}{0pt}%
& entry 2.3, a long text entry taking several\\
& & lines when set in a narrow column.
\end{longtable}
```

5-4-6

Page 1

Page 2

entry 1.1	entry 1.2	entry 1.3, a long text entry taking several lines.
entry 2.1	entry 2.2	entry 2.3, a long text entry taking several

lines when set in a narrow column.
---------------------------------------

Page 1

Page 2

## 5.5 Color in tables

The  $\text{\TeX}$  color commands provided by the `color` package are modeled on the font commands and may be used freely within tables. In particular, it is often convenient to use the `array` package preamble option `>` in order to apply a color to a whole column.

Day	Attendance	
Monday	57	\usepackage{array,color}
Tuesday	11	\begin{tabular}{>{\color{blue}\bfseries}lr}
Wednesday	96	Day & \textcolor{blue}{\bfseries} Attendance}\hline
Thursday	122	Monday& 57\\ Tuesday& 11\\
Friday	210	Wednesday& 96\\ Thursday& 122\\
Saturday	198	Friday& 210\\ Saturday& 198\\
Sunday	40	Sunday& 40
		\end{tabular}

5-5-1

It is perhaps more common to use color as a background to highlight certain rows or columns. In this case using the `\fcolorbox` command from the `color` package does not give the desired result, as typically the background should cover the full extent of the table cell. The `colortbl` package (by David Carlisle) provides several commands to provide colored backgrounds and rules in tables.

5-5-2

<b>Day</b>	<b>Attendance</b>
Monday	57
Tuesday	11
Wednesday	96
Thursday	122
Friday	210
Saturday	198
Sunday	40
<b>Total</b>	<b>724</b>

```
\usepackage{colortbl}
\begin{tabular}
    {>{\columncolor{blue}\color{white}\bfseries}lr}
    \rowcolor[gray]{0.8}
    \color{black} Day & \bfseries Attendance\\[2pt]
    Monday& 57 \\ Tuesday& 11 \\
    Wednesday& 96 \\ Thursday& 122 \\
    Friday& 210 \\ Saturday& 198 \\
    Sunday& 40 \\
    \cellcolor[gray]{0.8}\color{black}Total& 724
\end{tabular}
```

## 5.6 Customizing table rules and spacing

In this section we look at a number of packages that extend the `tabular` functionality by providing commands for drawing special table rules and fine-tuning the row spacing.

### 5.6.1 Colored table rules

The `colortbl` package extends the style parameters for table rules, allowing colors to be specified for rules and for the space between double rules. The declarations `\arrayrulecolor` and `\doublerulesepcolor` take the same argument forms as the `\color` command of the standard L<sup>A</sup>T<sub>E</sub>X `color` package.

Normally, these declarations would be used before a table, or in the document preamble, to set the color for all rules in a table. However, the rule color may be varied for individual rules using constructs very similar to the previous example.

5-6-1

A	B	C
X	Y	Z
100	10	1

```
\usepackage{colortbl} \setlength\arrayrulewidth{1pt}
\newcolumntype{B}{!{\color{blue}\vline}}
\newcommand\bhline
    {\arrayrulecolor{blue}\hline\arrayrulecolor{black}}
\newcommand\bcline[1]
    {\arrayrulecolor{blue}\cline{#1}\arrayrulecolor{black}}
\begin{tabular}{|cBc|c|}
\hline
A & B & C \\ \cline{1-1}\bcline{2-3}
X & Y & Z \\ \bhline
100 & 10 & 1 \\ \hline
\end{tabular}
```

### 5.6.2 Variable-width rules

Variable-width vertical rules may be constructed with the help of a `!{decl}` declaration and the basic `\TeX` command `\vrule` with a width argument. This command is used because it automatically fills the height of the column, whereas an explicit height must be specified for `\LaTeX`'s `\rule` command. To construct variable-width horizontal rules, it is again convenient to use a `\TeX` command, `\noalign`, to set the style parameter `\arrayrulewidth` so that it affects a single `\hline`, and then reset the rule width for the rest of the table.

In the example below, a new preamble option `I` is defined that produces a wide vertical rule. Similarly, a `\whline` command is defined that produces a wide horizontal rule.

```

\usepackage{array}
\newcolumntype{I}{!\vrule width 3pt}
\newlength\savedwidth
\newcommand\whline{\noalign{\global\savedwidth\arrayrulewidth
                           \global\arrayrulewidth 3pt}%
                  \hline
                  \noalign{\global\arrayrulewidth\savedwidth}}
\begin{tabular}{|c|c|c|} \hline
 A & B & C \\ \hline
 X & Y & Z \\ \hline
 100 & 10 & 1 \\ \hline \end{tabular}

```

5-6-2

### 5.6.3 hhline—Combining horizontal and vertical lines

The `hhline` package (by David Carlisle) introduces the command `\hhline`, which behaves like `\hline` except for its interaction with vertical lines.

\hhline{*decl*}

The declaration *decl* consists of a list of tokens with the following meanings:

- = A double \hline the width of a column.
  - A single \hline the width of a column.
  - ~ A column without \hline; a space the width of a column.  
  - | A \vline that “cuts” through a double (or single) \hline.
  - : A \vline that is broken by a double \hline.  
  - # A double \hline segment between two \vlines.
  - t The top rule of a double \hline segment.
  - b The bottom rule of a double \hline segment.
  - \* \*{3}{==#} expands to ==##==#, as in the \* form for the preamble.

If a double `\vline` is specified (|| or ::), then the `\hlines` produced by `\hhline` are broken. To obtain the effect of an `\hline` “cutting through” the double `\vline`, use a #.

The tokens t and b can be used between two vertical rules. For instance, |tb| produces the same lines as #, but is much less efficient. The main uses for these are to make constructions like |t: (top left corner) and :b| (bottom right corner).

If `\hhline` is used to make a single `\hline`, then the argument should only contain the tokens “-”, “~”, and “|” (and \* expressions).

An example using most of these features follows.

```
\usepackage{array, hhline}
\setlength{\arrayrulewidth}{.8pt}
\renewcommand{\arraystretch}{1.5}
\begin{tabular}{||c|c||c|c||c|c|c}
&&c&d&&&?
\hline
a&b&c&d&&&?
\hline
1&2&3&4&&&?
\hline
i&j&k&l&\multicolumn{1}{c}{?}&&?
\hline
w&x&y&z&&&?
\hline
\end{tabular}
```

a	b	c	d			?
1	2	3	4			?
i	j	k	l	&		?
w	x	y	z			?

[5-6-3]

The lines produced by `\hline` consist of a single (TeX primitive) `\hrule`. The lines produced by `\hhline` are made up of lots of small line segments. TeX will place these very accurately in the .dvi file, but the dvi driver used to view or print the output might not line up the segments exactly. If this effect causes a problem, you can try increasing `\arrayrulewidth` to reduce the effect.

#### 5.6.4 arydshln—Dashed rules

The `arydshln` package (by Hiroshi Nakashima) provides the ability to place dashed lines in tables. It is compatible with the `array` package, but must be loaded *after* `array` if both are used.

```
\hdashline[dash/gap]           \cdashline[colspec][dash/gap]
\firsthdashline[dash/gap]     \lastdashline[dash/gap]
```

The basic use of the package is very simple. A new preamble option “:” is introduced, together with two new commands `\hdashline` and `\cdashline`. These features may be used in the same way as the standard L<sup>A</sup>T<sub>E</sub>X “|” preamble option and `\hline` and `\cline` commands, except that dashed rather than solid lines are produced. If the `array` package is also loaded, then the commands `\firsthdashline`

and `\lasthdashline` are defined. They are dashed analogues of the `\firsthline` and `\lasthline` commands defined in that package.

A	B	C
X	Y	Z
100	10	1

```
\usepackage{array,arydshln}
\setlength\extrarowheight{4pt}% extra space on row top
\begin{tabular}{|c::c|c|}
\hline
A & B & C \\ \hline
X & Y & Z \\ \hline
100 & 10 & 1 \\ \hline
\end{tabular}
```

5-6-4

Each of the commands takes an optional argument that may be used to specify the style of rule to be constructed. For example, an optional argument of `[2pt/1pt]` would specify that the rule should use 2pt dashes separated by 1pt spaces. The `tabular` preamble syntax does not allow for optional arguments on preamble options, so the “`:`” option does not have an optional argument in which to specify the dash style. Instead, an additional preamble option “`;`” is defined that takes a mandatory argument of the form `dash/gap`, as demonstrated in the example below.

The default size of the dashes and gaps is `4pt`, which may be changed by setting the style parameters `\dashlinedash` and `\dashlinegap` via `\setlength`. This ability is shown in the example below.

A	B	C
X	Y	Z
100	10	1

```
\usepackage{array,arydshln}
\renewcommand\arraystretch{1.3333}% extra space evenly
                                % distributed
\setlength\dashlinedash{1pt}
\setlength\dashlinegap{1pt}
\begin{tabular}{;{5pt/2pt}c::c;c;{5pt/2pt}}
\hline
A & B & C \\ \hline
X & Y & Z \\ \hline
100 & 10 & 1 \\ \hline
\end{tabular}
```

5-6-5

The package may use any one of three methods for aligning the dashes within a table cell. The package may sometimes produce an overlarge gap at the edge of a table entry because there is not enough room to fit in the next “dash”. If this happens you might try specifying an alternative placement algorithm using the command `\ADLdrawingmode{m}`, where `m` may be 1 (the default), 2, or 3.

The package documentation contains details of the placement algorithms used in each of these cases, but in practice you can just experiment with your particular table and dash styles to see which setting of `\ADLdrawingmode` gives the most pleasing result.

### 5.6.5 tabls—Controlling row spacing

One of the difficulties of using L<sup>A</sup>T<sub>E</sub>X tables with irregular-sized entries is the challenge of obtaining a good spacing around large entries, especially in the presence of horizontal rules. The standard L<sup>A</sup>T<sub>E</sub>X command `\arraystretch` or the `\extrarowheight` parameter introduced by the `array` package may help in this case. Both, however, affect all the rows in the table. It is sometimes desirable to have a finer-grained control, an ability that is provided by the `tabls` package (by Donald Arseneau). Note that `tabls` is incompatible with the `array` package and its derivatives. The package introduces three new parameters:

`\tablinesep` The minimum space between text on successive lines of a table.

Negative values are treated as zero. The default is 1pt. If this parameter is set to 0pt, the code will not check the height of table entries to avoid touching text (which will emulate the default behavior of `tabular`).

`\arraylinesep` The equivalent to `\tablinesep` for the `array` environment.

`\extrarulesep` Extra space added above and below each `\hline` and `\cline`.

There will be space of at least  $\extrarulesep + 0.5\tablinesep$  between an `\hline` and text in the following table row. Negative values will reduce the space below the line, until the line is touching the text. Larger negative values will *not* cause the line to overprint the text. The default value is 3pt.

In addition, the `\hline` command is extended with an optional argument like that of `\backslash`. This argument specifies additional space to insert below the rule.

A	B	C
100	10	1

5-6-6

```
\usepackage{tabls} \setlength\tablinesep{2pt}
\begin{tabular}{|c|c|c|} \hline
\large A &\large B &\large C \\ \hline[5pt]
100 & 10 & 1 \\ [5pt] \hline
\end{tabular}
```

### 5.6.6 booktabs—Formal ruled tables

The vertical rules in a `tabular` environment are made up of a series of rule segments, one in each row of the table. Commands designed to improve vertical spacing between rows or around horizontal rules need to be carefully designed not to “break” any vertical rules by adding space between these rule segments. An alternative approach is taken by the `booktabs` package (by Simon Fear). It is designed to produce more formal tables according to a more traditional typographic style that uses horizontal rules of varying widths to separate table headings, but does not use any vertical rules. The `|` preamble option is not disabled when using this package, but its use is not supported and the extra commands for horizontal rules described below are not designed to work well in conjunction with vertical rules. Similarly, `booktabs` commands are not designed to support double rules as produced by the `||` or `\hline\hline`.

*Do not use vertical rules*

*Do not use double rules*

The `booktabs` commands may be used with the standard `tabular` environments, the extended versions provided by the `array` package, and in the `longtable` environment provided by the `longtable` package.

An example showing the most commonly used commands provided by the `booktabs` package is shown below.

\usepackage{booktabs}		
\begin{tabular}{@{}l l r@{}}		
\toprule		
\multicolumn{2}{c}{Item} & \multicolumn{1}{c}{Price/lb} \\		
Food	Category	\$
Apples	Fruit	1.50
Oranges	Fruit	2.00
Beef	Meat	4.50
Food & Category & \multicolumn{1}{c}{\\$}\\		
\midrule		
Apples & Fruit & 1.50 \\		
Oranges & Fruit & 2.00 \\		
Beef & Meat & 4.50 \\		
\bottomrule		
\end{tabular}		

[ 5-6-7 ]

\toprule [width]      \midrule [width]      \bottomrule [width]

The `booktabs` package provides the `\toprule`, `\midrule`, and `\bottomrule` commands. They are used in the same way as the standard `\hline` but have better vertical spacing, and widths specified by the length parameters `\heavyrulewidth` (for top and bottom rules) and `\lightrulewidth` (for mid-table rules). These parameters default to `0.08em` and `0.05em`, respectively (where the `em` is determined by the default document font at the point the package is loaded).

The spacing above and below the rules is determined by the length parameters: `\abovetopsep` (default `0pt`) is the space above top rules, `\aboverulesep` (default `0.4ex`) is the space above mid-table and bottom rules, `\belowrulesep` (default `0.65ex`) is the space below top and mid rules, and `\belowbottomsep` (default `0pt`) is the space below bottom rules.

If you need to control the widths of individual rules, all of these commands take an optional width argument. For example, `\midrule[0.5pt]` would produce a rule of width `0.5pt`.

When these commands are used inside a `longtable` environment, they may take an optional (`trim`) argument as described below for `\cmidrule`. This argument may be used to make the rules slightly less than the full width of the table.

\cmidrule [width] (trim) {col1-col2}

The `\cmidrule` command produces rules similar to those created with the standard L<sup>A</sup>T<sub>E</sub>X `\cline` command. The `col1-col2` argument specifies the columns over

which the rule should be drawn. Unlike the rules created by `\cline`, these rules do not, by default, extend all the way to the edges of the column. Thus, one may use `\cmidrule` to produce rules on adjacent columns without them touching, as shown in the example above.

If the optional *width* argument is not specified, the rule will be of the width specified by the `\cmidrulewidth` length parameter (default 0.03em).

By default, the rule extends all the way to the left, but is “trimmed” from the rightmost column by the length specified in the length parameter `\cmidrulekern`. The optional (*trim*) argument may contain one or more of the options `l lwd`, `r rwd`, where `l` and `r` indicate that the rule is to be trimmed from the left or right, respectively. Each `l` and `r` may optionally be followed by a width, in which case the rule is trimmed by this amount rather than by the default `\cmidrulekern`.

Normally, if one `\cmidrule` command immediately follows another, then the rules will be drawn across the specified columns on the same horizontal line. A command `\morecmidrules` is provided that may be used to terminate a row of mid-table rules. Following mid-table rules will then appear on a new line separated by the length `\cmidrulesep`, which by default is equal to `\doublerulesep`.

Each group of rules produced by `\cmidrule` is preceded and followed by a space of width `\midrulesep`, so this command generates the same spacing as `\midrule`. By default, however, the `\cmidrule` rules are lighter (thinner) than the rules produced by `\midrule`.

`\addlinespace [width]`

Extra space may be inserted between rows using `\addlinespace`. This command differs from using the optional argument to `\backslash`, as the former may also be used immediately before or after the rule commands.

If used in this position the command replaces the default spacing that would normally be produced by the rule. If the optional width argument is omitted it defaults to the length parameter `\defaultaddspace` (which defaults to 0.5em).

`\specialrule{width}{abovespace}{below space}`

Finally, if none of the other commands produces a suitable rule then the command `\specialrule` may be used. It takes three mandatory arguments that specify the width of the rule, and the space above and below the rule.

As the intention of the package is to produce “formal” tables with well-spaced lines of consistent thickness, the package author warns against overuse of the optional arguments and special commands to produce lines with individual characteristics. Nevertheless, these features may be useful in special circumstances.

The example on the following page shows the effect of many of these options as well as demonstrating that overuse of the commands will produce a very unpleasing layout.

```

\usepackage{booktabs}
\begin{tabular}{@{}llr@{}}
\toprule
& \multicolumn{2}{c}{Item} & \multicolumn{1}{c}{Price/lb} \\
\cmidrule(r){1-2}\cmidrule(l){3-3}
& a & b & c \\
\cmidrule(l{2pt}r{2pt}){1-2}\cmidrule(l{2pt}r{2pt}){3-3}
\morecmidrules
\cmidrule(l{2pt}r{2pt}){2-3}
\addlinespace[5pt]
& Food & Category & \multicolumn{1}{c}{\$} \\
\midrule
Food & Category & \$ & Apples & Fruit & 1.50 \\
& & & Oranges & Fruit & 2.00 \\
\addlinespace
Apples & Fruit & 1.50 & Beef & Meat & 4.50 \\
Oranges & Fruit & 2.00 & \specialrule{.5pt}{3pt}{3pt} \\
Beef & Meat & 4.50 & x & y & z \\
x & y & z & \bottomrule
\end{tabular}

```

5-6-8

## 5.7 Further extensions

Two other package files extend the `array` package with additional functionality. The first provides for table entries spanning more than one row. The second makes it easier to align decimal numbers in a column.

You can simulate a cell spanning a few rows vertically by putting the material in a zero-height box and raising it.

100	qqq	
	A	B
20000000	10	10

```

\begin{tabular}{|c|c|c|} \hline
& \multicolumn{2}{c}{qqq} \\ \cline{2-3}
\raisebox{1.5ex}[0cm][0cm]{100} & A & B \\ \hline
20000000 & 10 & 10 \\ \hline
\end{tabular}

```

5-7-1

Similarly, you can use a standard `tabular` preamble of the form `r@{.}1` to create two table columns and produce the effect of a column aligned on a decimal point, but then the input looks rather strange. For an alternative solution, see Section 5.7.2 on page 274.

1.2	\begin{tabular}{r@{.}1}
1.23	1 & 2 \\ 1 & 23 \\ 913 & 17
913.17	\end{tabular}

5-7-2

This strategy is not always convenient, because you have to be aware that the “column” is really two columns of the table. This consideration becomes important when counting columns for the `\multicolumn` or `\cline` commands. Also, you need to locally set `\extracolsep` to `0pt` if you use this construct in a `tabular*` environment, otherwise TeX may insert space after the decimal point to spread the table to the specified width.

### 5.7.1 `multirow`—Vertical alignment in tables

The `multirow` package (by Jerry Leichter) automates the procedure of constructing tables with columns spanning several rows by defining a `\multirow` command. Fine-tuning is possible by specifying optional arguments. This ability can be useful when any of the spanned rows are unusually large, when `\strut` commands are used asymmetrically about the centerline of spanned rows, or when descenders are not taken into account correctly. In these cases the vertical centering may not come out as desired, and the fixup argument `vmove` can then be used to introduce vertical shifts by hand.

```
\multirow{nrow} [njot] [width] [vmove] {contents}
```

Inside an array, this command is somewhat less useful because the lines have an extra `\jot` of space (a length, by default equal to `3pt`, that is used for opening up displays), which is not accounted for by `\multirow`. Fixing this problem (in general) is almost impossible. Nevertheless, a semiautomatic fix is to set the length parameter `\bigstrutjot` to `\jot`, and then use the second argument `njot` of `\multirow` with a value equal to half the number of rows spanned.

You have some ability to control the formatting within cells. Just before the text to be typeset is expanded, the `\multirowsetup` macro is automatically executed to set up any special environment. Initially, `\multirowsetup` contains just `\raggedright`, but it can be redefined with `\renewcommand`.

The `\multirow` command works in one or more columns, as shown in the example below.

	C2a	C4a	
	C2b	C4b	
	C2c	C4c	
	C2d	C4d	

```
\usepackage{multirow}
\begin{tabular}{|l|l|l|l|} \hline
\multirow{4}{14mm}{Text in column 1} & C2a & \multirow{4}{14mm}{Text in column 3} & \\
& C4a & \\ & C2b & & \\ & C4b & \\ & C2c & & \\ & C4c & \\ & C2d & & \\ & C4d & \\ \hline
\end{tabular}
```

You are now in a position to typeset the small example shown at the beginning of this section without having to use the `\raisebox` command. First, you must

change the alignment inside the `\multirow` paragraph to `\centering`. Next, you calculate the width of the text in the column, which is required by the `\multirow` command. If the column with the spanned rows has a fixed width, as in our other examples, this step is unnecessary.

```
\usepackage{multirow}
\renewcommand\multirowsetup{\centering}
\newlength\LL \settowidth\LL{100}
\begin{tabular}{|c|c|c|} \hline
\multirow{2}{\LL}{100} & \multicolumn{2}{c}{qqq} \\ \cline{2-3}
& A & B \\ \hline
20000000 & 10 & 10 \\ \hline
\end{tabular}
```

[ 5-7-4 ]

The effect of the optional vertical positioning parameter `vmove` can be seen below. Note the effect of the upward move by 3 mm of the lower third of the table.

Common text in column 1	Cell 1a	\usepackage{multirow}
	Cell 1b	\begin{tabular}{ l l } \hline
	Cell 1c	\multirow{4}{25mm}{Common text in column 1}
	Cell 1d	& Cell 1a \\ \cline{2-2} & Cell 1b \\ \cline{2-2}
Common text in column 1	Cell 2a	& Cell 1c \\ \cline{2-2} & Cell 1d \\ \hline
	Cell 2b	\multirow{4}{25mm}{-3mm}{Common text in column 1}
	Cell 2c	& Cell 2a \\ \cline{2-2} & Cell 2b \\ \cline{2-2}
	Cell 2d	& Cell 2c \\ \cline{2-2} & Cell 2d \\ \hline
Common text in column 1	Cell 3a	\multirow{4}{25mm}{3mm}{Common text in column 1}
	Cell 3b	& Cell 3a \\ \cline{2-2} & Cell 3b \\ \cline{2-2}
	Cell 3c	& Cell 3c \\ \cline{2-2} & Cell 3d \\ \hline
	Cell 3d	\end{tabular}

[ 5-7-5 ]

### 5.7.2 dcolumn—Decimal column alignments

The `dcolumn` package (by David Carlisle) provides a system for defining columns of entries in `array` or `tabular` environments that are to be aligned on a “decimal point”. Entries with no decimal part, those with no integer part, and blank entries are also dealt with correctly.

The package defines a “Decimal” tabular preamble option, `D`, that takes three arguments.

`D{inputsep}{outputsep}{decimal places}`

`inputsep` A single character, used as separator (or “decimal point”) in the source file (for example, “.” or “,”).

*outputsep* The separator to be used in the output. It can be the same as the first argument, but may also be any math mode expression, such as `\cdot`.

*decimal places* The maximum number of decimal places in the column. If this value is negative, any number of decimal places is allowed in the column, and all entries will be centered on the separator. Note that this choice can cause a column to be too wide (see the first two columns in the example below). Another possibility is to specify the number of digits *both* to the left and to the right of the decimal place, using an argument of the form `{left,right}` as described below.

If you do not want to use all three entries in the preamble, you can customize the preamble specifiers by using `\newcolumntype` as demonstrated below.

```
\newcolumntype{d}{[1]{D{.}}{\cdot}{#1}}
```

The newly defined “d” specifier takes a single argument specifying the number of decimal places. The decimal separator in the source file is the normal dot “.”, while the output uses the math mode “ $\cdot$ ”.

```
\newcolumntype{.}{D{.}{.}{-1}}
```

In this case the “.” specifier has no arguments: the normal dot is used in both input and output. The typeset entries should be centered on the dot.

```
\newcolumntype{,}{D{,}{,}{2}}
```

The “,” specifier defined here uses the comma “,” as a decimal separator in both input and output, and the typeset column should have (at most) two decimal places after the comma.

These definitions are used in the following example, in which the first column, with its negative value for *decimal places* (signaling that the decimal point should be in the center of the column), is wider than the second column, even though they both contain the same input material.

```
\usepackage{dcolumn}
\newcolumntype{d}{[1]{D{.}}{\cdot}{#1}}
\newcolumntype{.}{D{.}{.}{-1}}
\newcolumntype{,}{D{,}{,}{2}}
\begin{tabular}{|d{-1}|d{2}||.||}
  1.2      & 1.2      & 1.2      & 1.2      & 1.2      & \\
  1.23     & 1.23     & 12.5     & 300.2    & 1.23     & \\
  1121.2   & 1121.2   & 861.20   & 674.29   & 1121.2   & \\
  184      & 184      & 10       & 69       & 184      & \\
  .4       & .4       &           & ,4       & .4       & \\
                  &           &           & .4       &           & \\
                  &           &           &           &           & \\
\end{tabular}
```

If the table entries include only numerical data that must be aligned, the alignment forms shown in the above example should be sufficient. However, if the columns contain headings or other entries that will affect the width of the column, the positioning of the numbers within the column might not be as desired. In the example below, in the first column the numbers appear to be displaced toward the left of the column, although the decimal point is centered. In the second column the numbers are flush right under a centered heading, which is sometimes the desired effect but (especially if there are no table rules) can make the heading appear dissociated from the data. The final column shows the numbers aligned on the decimal point and centered as a block under the heading. This effect is achieved by using a third argument to the D preamble option of 4.2 specifying that at most four digits can appear to the left of the point, and two digits to the right of it.

wide heading	wide heading	wide heading
1000.20	1000.20	1000.20
123.45	123.45	123.45

```
\usepackage{dcolumn}
\begin{tabular}{|D..{-1}|D..{2}|D..{4.2}|}
\multicolumn{1}{|c|}{wide heading} &
\multicolumn{1}{c|}{wide heading} &
\multicolumn{1}{c|}{wide heading}\|[3pt]
1000.20 & 1000.20 & 1000.20 \\
123.45 & 123.45 & 123.45
\end{tabular}
```

5-7-7

The following is a variant of an example in the *L<sup>A</sup>T<sub>E</sub>X Manual* showing that D column alignments may be used for purposes other than aligning numerical data on a decimal point.

#### GG&A Hoofed Stock

Year	Price	Comments	Other
	low-high		
1971	97-245	Bad year for farmers in the West.	23,45
72	245-245	Light trading due to a heavy winter.	435,23
73	245-2001	No gnus was very good gnus this year.	387,56

```
\usepackage{dcolumn}
\newcolumntype{+}{D{/}{\mbox{--}}}{4}
\newcolumntype{,}{D{,}{,}}{2}
\begin{tabular}{|r||+|c|l|l|}
>{\raggedright}p{2.2cm},|} \hline
\multicolumn{4}{|c|}{GG&A Hoofed Stock}\hline
\hline
& \multicolumn{1}{c|}{Price} & \multicolumn{2}{c}{Year} \\
& \cline{2-2} \multicolumn{1}{c|}{Comments} & \multicolumn{1}{c|}{Other} & \hline
1971 & 97/245 & Bad year for farmers in the West. & 23,45 \\
72 & 245/245 & Light trading due to a heavy winter. & 435,23 \\
73 & 245/2001 & No gnus was very good gnus this year. & 387,56 \\
\end{tabular}
```

5-7-8

## 5.8 Footnotes in tabular material

As stated in Section 3.2.2 on page 112, footnotes appearing inside tabular material are not typeset by standard L<sup>A</sup>T<sub>E</sub>X. Only the environments `tabularx`, `longtable`, `mpsupertabular`, and `mpsupertabular*` will automatically typeset footnotes.

As you generally want your “table notes” to appear just below the table, you will have to tackle the problem yourself by managing the note marks and, for instance, by using `\multicolumn` commands at the bottom of your `tabular` environment to contain your table notes.

### 5.8.1 Using minipage footnotes with tables

If a `tabular` or `array` environment is used inside a `minipage` environment, standard footnote commands may be used inside the table. In this case these footnotes will be typeset at the bottom of the `minipage` environment, as explained in Section 3.2.1 on page 110.

In the example below note the redefinition of `\thefootnote` that allows us to make use of the `\footnotemark` command inside the `minipage` environment. Without this redefinition `\footnotemark` would have generated a footnote mark in the style of the footnotes for the main page, as explained in Section 3.2.2.

```

\begin{minipage}{\linewidth}
\renewcommand{\thefootnote}{\thempfootnote}
\begin{tabular}{l}
\multicolumn{2}{c}{\bfseries PostScript} \\
& \scriptsize Type 1 fonts \\ \hline
\multicolumn{2}{c}{\bfseries PostScript Type 1 fonts} \\
\hline
Couriera & cour, courb, courbi, couri \\
Charterb & bchb, bchbi, bchr, bchri \\
Nimbusc & unmr, unmrs \\
URW Antiquac & uaqrrc \\
URW Groteskc & ugqp \\
Utopiad & putb, putbi, putr, putri \\
\end{tabular}
\begin{array}{l}
\footnote{Donated by IBM.} \\
\footnote{Donated by Bitstream.} \\
\footnote{Donated by URW GmbH.} \\
\footnote{Donated by Adobe.}
\end{array}
\end{minipage}

```

Of course, this approach does not automatically limit the width of the footnotes to the width of the table, so a little iteration with the `minipage` width argument might be necessary to achieve the desired effect.

### 5.8.2 threeparttable—Setting table and notes together

Another way to typeset table notes is with the package `threeparttable`, written by Donald Arseneau. This package has the advantage that it indicates unambiguously that you are dealing with notes inside tables. Moreover, it gives you full control of the actual reference marks and offers the possibility of having a caption for your tabular material. With this package the table notes are automatically set in a box with width set equal to the width of the table.

*Table notes set to  
the width of the  
table*

Normally, the `threeparttable` environment would be contained within a `table` environment so that the table would float. However, `threeparttable` may also be used directly, in which case it constructs a nonfloating table similar to the nonfloating `table` environment set-up described in Example 6-3-4 on page 295.

```
\usepackage{threeparttable}
\begin{threeparttable}
\caption[Example of a \texttt{threeparttable} environment]{\textbf{PostScript Type 1 fonts}}
\begin{tabular}{@{}l@{}}
Courier\tnote{a} & cour, courb, courbi, couri \\
Charter\tnote{b} & bchb, bchbi, bchr, bchri \\
Nimbus\tnote{c} & unmrr, unmrs \\
URW Antiqua\tnote{c} & uaqrcc \\
URW Grotesk\tnote{c} & ugqp \\
Utopia\tnote{d} & putb, putbi, putr, putri\\
\end{tabular}
\begin{tablenotes}
\item[a]Donated by IBM.
\item[b]Donated by Bitstream.
\item[c]Donated by URW GmbH.
\item[d]Donated by Adobe.
\end{tablenotes}
\begin{tablenotes}[flushleft,online]
\item[a]Donated by IBM.
\item[b]Donated by Bitstream.
\item[c]Donated by URW GmbH.
\item[d]Donated by Adobe.
\end{tablenotes}
\begin{tablenotes}[para]
\item[]Donated by:
\item[a]IBM, \item[b]Bitstream,
\item[c]URW GmbH,
\item[d]Adobe.
\end{tablenotes}
\end{threeparttable}
```

Table 1: PostScript Type 1 fonts	
Courier <sup>a</sup>	cour, courb, courbi, couri
Charter <sup>b</sup>	bchb, bchbi, bchr, bchri
Nimbus <sup>c</sup>	unmr, unmrs
URW Antiqua <sup>c</sup>	uaqrcc
URW Grotesk <sup>c</sup>	ugqp
Utopia <sup>d</sup>	putb, putbi, putr, putri

<sup>a</sup> Donated by IBM.

<sup>b</sup> Donated by Bitstream.

<sup>c</sup> Donated by URW GmbH.

<sup>d</sup> Donated by Adobe.

a Donated by IBM.

b Donated by Bitstream.

c Donated by URW GmbH.

d Donated by Adobe.

Donated by: <sup>a</sup> IBM, <sup>b</sup> Bitstream,

<sup>c</sup> URW GmbH, <sup>d</sup> Adobe.

As its name suggests, the `threeparttable` environment consists of three parts. The **caption** consists of the usual `\caption` command (which may come before or after the table). The **table** may use one of the standard `tabular` or `tabular*` environments, the extended variants defined in the `array` package, or the `tabularx` environment defined in `tabularx`. Support for other tabular environments may be added in later releases, the package documentation lists the currently supported environments. The third part of a `threeparttable` is the text of the table **notes**, which consists of one or more `tablenotes` environments.

The `threeparttable` package offers several options to control the typesetting of the table notes:

`para` Notes are set within a paragraph, without forced line breaks.

`flushleft` No hanging indentation is applied to notes.

`online` Note labels are printed normal size, not as superscripts.

`normal` Normal default formatting is restored.

Each of these options may be used as a package option to set the default style for all such tables within the document. Alternatively, they may be used as shown in the example, on individual `tablenotes` environments.

In addition to these options the package has several commands that may be redefined to control the formatting in more specific ways than those provided by the package options. See the package documentation for details.

## 5.9 Applications

The following examples involve somewhat more complex placement requirements, allowing advanced functions such as the provision of nested tables. Here, we will put to work many of the features described in this chapter.

### 5.9.1 Managing tables with wide entries

Sometimes it is necessary to balance white space between narrow columns uniformly over the complete width of the table. For instance, the following table has a rather wide first row, followed by a series of narrow columns.

this-is-a-rather-long-row	<code>\begin{tabular}{ccc}</code>
C1 C2 C3	<code>\multicolumn{3}{c}{this-is-a-rather-long-row}\\"</code>
2.1 2.2 2.3	<code>C1 &amp;C2 &amp;C3 \\ 2.1&amp;2.2&amp;2.3 \\ 3.1&amp;3.2&amp;3.3</code>
3.1 3.2 3.3	<code>\end{tabular}</code>

You can put some rubber length in front of each column with the help of the `\extracolsep` command. The actual value of the rubber length is not important, as long as it can shrink enough to just fill the needed space. In this case you must, of course, specify a total width for the table. We could use `\linewidth` and make

the table full width, but here we can obtain a better result by precalculating the width of the wide entry and specifying it as the total width of the `tabular*`.

```
\usepackage{array}
\newlength\Mylen
\settowidth\Mylen{this-is-a-rather-long-row}
\addtolength\Mylen{2\tabcolsep}
\begin{tabular*}{\Mylen}%
    !{\extracolsep{4in minus 4in}}ccc
\multicolumn{3}{c}{this-is-a-rather-long-row}\\
C1 & C2 & C3 \\
2.1 & 2.2 & 2.3 \\
3.1 & 3.2 & 3.3
\end{tabular*}
```

5-9-2

To achieve correct alignment, we needed to take into account the column separation (`\tabcolsep`) on both sides of an entry. Alternatively, we could have suppressed the inter-column spaces at the left and right of the `tabular*` by using `\{ \}` expressions.

### 5.9.2 Tables inside tables

The example below shows how, with a little bit of extra effort, you can construct complex table layouts with L<sup>A</sup>T<sub>E</sub>X.

```
\firsthline \lasthline
```

The family of `tabular` environments allows vertical positioning with respect to the baseline of the text in which the environment appears. By default, the environment appears centered. This preference can be changed to align with the first or last line in the environment by supplying a `t` or `b` value to the optional position argument. Note that this approach does not work when the first or last element in the environment is an `\hline` command—in that case, the environment is aligned at the horizontal rule.

Tables with no hline commands used versus tables with some hline commands used.

```
\usepackage{array}
Tables \begin{tabular}[t]{l}
with no\\ hline \\
commands \\
used
\end{tabular}
versus tables
\begin{tabular}[t]{|l|} \hline
with some \\
hline \\
commands \\
\hline
\end{tabular} \hline
used.
```

5-9-3

To achieve proper alignments you can use the two commands `\firsthline` and `\lasthline`, which are special versions of `\hline` defined in the `array` pack-

age. These commands enable you to align the information in the tables properly as long as their first or last lines do not contain extremely large objects.

Tables with no hline commands used versus tables with some hline commands used.

5.9-4

```
\usepackage{array}
Tables \begin{tabular}[t]{l}
with no \\ hline \\
commands \\
used
\end{tabular}
versus tables
\begin{tabular}[t]{|l|} \firsthline
with some \\ hline \\
commands \\
\lasthline
\end{tabular} used.
```

```
\setlength\extratabsurround{dim}
```

The implementation of the two commands contains an extra dimension, `\extratabsurround`, to add space at the top and the bottom of such an environment. It is helpful for properly aligning nested tabular material, as shown in the next example.

```
\usepackage{array}
\setlength\extratabsurround{5pt}
\begin{tabular}{|c|} \hline
\emph{name} & \emph{telephone} \\ \hline
John & \begin{tabular}[t]{|cc|} \firsthline
\emph{day} & \multicolumn{1}{c}{\itshape telephone} \\ \hline
Wed & 5554434 \\ \hline
Mon & \begin{tabular}[t]{|cc|} \firsthline
\emph{time} & \emph{telephone} \\ \hline
8--10 & 5520104 \\ \hline
& 1--5 & 2425588 \\ \hline
\end{tabular} \\ \hline
\end{tabular} \\ \hline
Martin & \begin{tabular}[t]{|cp{4.5cm}|} \firsthline
\emph{telephone} & \multicolumn{1}{c}{\itshape instructions} \\ \hline
3356677 & Mary should answer forwarded message. \\ \hline
\end{tabular} \\ \hline
Peter & \begin{tabular}[t]{|cl|} \firsthline
\emph{month} & \multicolumn{1}{c}{\itshape telephone} \\ \hline
Sep--May & 5554434 \\ \hline
Jul--Aug & 2211456 \\ \hline
\end{tabular} \\ \hline
\end{tabular}
```

<i>name</i>	<i>telephone</i>	
John	<i>day</i>	<i>telephone</i>
	Wed	5554434
	Mon	<i>time</i> <i>telephone</i>
	8–10	5520104
	1–5	2425588
Martin	<i>telephone</i>	<i>instructions</i>
	3356677	Mary should answer forwarded message.
Peter	<i>month</i>	<i>telephone</i>
	Sep–May	5554434
	Jun	No telephone
	Jul–Aug	2211456

5-9-5

The *LATEX* code below shows how you can combine the various techniques and packages described earlier in this chapter. We used the package *tabularx* to generate a 12 column table in which columns 3 to 12 are of equal width. We used the package *multirow* to generate the stub head, “Prefix”, which spans two rows in column 1. To position the stub head properly, we calculated the width of the title beforehand.

```
\usepackage{array,tabularx,multirow}
\newlength\Tl \settowidth{\Tl}{Prefix} \setlength\tabcolsep{1mm}
\newcommand\T[1]{\$10^{#1}\$}
\begin{tabularx}{\linewidth}{|l|l|*{10}{>{\small X}|}} \hline
\multicolumn{12}{|c|}{\textbf{Prefixes used in the SI system of units}}\\\hline
\multicolumn{2}{|c|}{Factor} &
\T{24}&\T{21}&\T{18}&\T{15}&\T{12}&\T{9}&\T{6}&\T{3}&\T{2}&\T{} \\\hline
\multicolumn{2}{|l|}{\multirow{2}{\Tl}{Prefix}}&Name &
yotta &zetta &exa &peta &tera &giga &mega &kilo &hecto &deca \\
&Symbol &&&&&&&&&&\\\hline
... text omitted ...

```

5-9-6

Prefixes used in the SI system of units												
Factor	10 <sup>24</sup>	10 <sup>21</sup>	10 <sup>18</sup>	10 <sup>15</sup>	10 <sup>12</sup>	10 <sup>9</sup>	10 <sup>6</sup>	10 <sup>3</sup>	10 <sup>2</sup>	10		
Prefix	Name	yotta	zetta	exa	peta	tera	giga	mega	kilo	hecto	deca	
	Symbol	Y	Z	E	P	T	G	M	k	h	da	
Prefix	Symbol	y	z	a	f	p	n	$\mu$	m	c	d	
	Name	yocto	zepto	atto	femto	pico	nano	micro	milli	centi	deci	
Factor		10 <sup>-24</sup>	10 <sup>-21</sup>	10 <sup>-18</sup>	10 <sup>-15</sup>	10 <sup>-12</sup>	10 <sup>-9</sup>	10 <sup>-6</sup>	10 <sup>-3</sup>	10 <sup>-2</sup>	10 <sup>-1</sup>	

## CHAPTER 6

# Mastering Floats

Documents would be easier to read if all the material that belonged together was never split between pages. However, this is often technically impossible and TEX will, by default, split textual material between two pages to avoid partially filled pages. Nevertheless, when this outcome is not desired (as with figures and tables), the material must be “floated” to a convenient place, such as the bottom or the top of the current or next page, to prevent half-empty pages.

This chapter shows how “large chunks” of material can be kept conveniently on the same page by using a float object. We begin by introducing the parameters that define how LATEX typesets its basic `figure` and `table` float environments, and we describe some of the packages that make it easy to control float placement (Section 6.2). We then continue by explaining how you can define and use your own floating environments, or, conversely, use LATEX’s `\caption` mechanism to enter information into the list of figures and tables for nonfloating material (Section 6.3.1).

It is often visually pleasing to include a “picture” inside a paragraph, with the text wrapping around it. Various packages have been written to achieve this goal more or less easily; in Section 6.4 we look at two of them in some detail.

The final section addresses the problem of customizing captions. There is a recognized need to be able to typeset the description of the contents of figures and tables in many different ways. This includes specifying sub-figures and sub-tables, each with its own caption and label, inside a larger float.

Many float-related packages have been developed over the years and we cannot hope to mention them all here. In fact, the packages that we describe often feature quite a few more commands than we are able to illustrate. Our aim is to enable you to make an educated choice and to show how a certain function can be

obtained in a given framework. In each case consulting the original documentation will introduce you to the full possibilities of a given package.

## 6.1 Understanding float parameters

Floats are often problematic in the present version of L<sup>A</sup>T<sub>E</sub>X, because the system was developed at a time when documents contained considerably less graphical material than they do today. Placing floats (tables and figures) works relatively well as long as the space they occupy is not too large compared with the space taken up by the text. If a lot of floating material (pictures or tables) is present, however, then it is often the case that all material from a certain point onward floats to the end of the chapter or document. If this effect is not desired, you can periodically issue a `\clearpage` command, which will print all unprocessed floats. You can also try to fine-tune the float style parameters for a given document or use a package that allows you to always print a table or figure where it appears in the document. In the list below “float” stands for a table or a figure and a “float page” is a page that contains only floats and no text. Changes to most of the parameters will only take effect on the next page (not the current one).

`topnumber` Counter specifying the maximum number of floats allowed at the top of the page (the default number is 2). This can be changed with the `\setcounter` command.

`bottomnumber` Counter specifying the maximum number of floats allowed at the bottom of the page (the default number is 1). This can be changed with `\setcounter`.

`totalnumber` Counter specifying the maximum number of floats allowed on a single page (the default number is 3). This can be changed with `\setcounter`.

`\topfraction` Maximum fraction of the page that can be occupied by floats at the top of the page (e.g., 0.2 means 20% can be floats; the default value is 0.7). This can be changed with `\renewcommand`.

`\bottomfraction` Maximum fraction of the page that can be occupied by floats at the bottom of the page (the default value is 0.3). This can be changed with `\renewcommand`.

`\textfraction` Minimum fraction of a normal page that must be occupied by text (the default value is 0.2). This can be changed with `\renewcommand`.

`\floatpagefraction` Minimum fraction of a float page that must be occupied by floats, thus limiting the amount of blank space allowed on a float page (the default value is 0.5). This can be changed with `\renewcommand`.

`dbltopnumber` Analog of `topnumber` for double-column floats in two-column style (the default number is 2). This can be changed with `\setcounter`.

\dbltopfraction Analog of \topfraction for double-column floats on a two-column page (the default value is 0.7). This can be changed with \renewcommand.

\dblfloatpagefraction Analog of \floatpagefraction for a float page of double-column floats (the default value is 0.5). This can be changed with \renewcommand.

\floatsep Rubber length specifying the vertical space added between floats appearing at the top or the bottom of a page (the default is 12pt plus 2pt minus 2pt for 10pt and 11pt document sizes, and 14pt plus 2pt minus 4pt for 12pt document size). This can be changed with \setlength.

\textfloatsep Rubber length specifying the vertical space added between floats, appearing at the top or the bottom of a page, and the text (the default is 20pt plus 2pt minus 4pt). This can be changed with \setlength.

\intextsep Rubber length specifying the vertical space added below and above a float that is positioned in the middle of text when the h option is given (the default is similar to \floatsep). This can be changed with \setlength.

\dblfloatsep Rubber length that is the analog of \floatsep for double-width floats on a two-column page (the default is like \floatsep). This can be changed with \setlength.

\dbltextfloatsep Rubber length that is the analog of \textfloatsep for double-width floats on a two-column page (the default is like \textfloatsep on a text page, but is 8pt plus 2fil on a page that contains only floats). This can be changed with \setlength.

\topfigrule Command to produce a separating item (by default, a rule) between floats at the top of the page and the text. It is executed immediately before placing the \textfloatsep that separates the floats from the text. Like the \footnoterule, it must not occupy any vertical space.

\botfigrule Same as \topfigrule, but put after the \textfloatsep skip separating text from the floats at the bottom of the page.

\dblfigrule Similar to \topfigrule, but for double-column floats.

Changing the values of these parameters lets you modify the behavior of L<sup>A</sup>T<sub>E</sub>X's algorithm for placing floats. To obtain the optimal results, however, you should be aware of the subtle dependencies that exist between these parameters.

If you use the default values in a document you will observe that, with many floats, the formatted document will contain several float pages—that is, pages containing only floats. Often such pages contain a lot of white space. For example, you may see a page with a single float on it, occupying only half of the possible space,

*The problem of half-empty float pages*

so that it would look better if L<sup>A</sup>T<sub>E</sub>X had filled the remaining space with text. The reason for this behavior is that the algorithm is designed to try placing as many dangling floats as possible after the end of every page. The procedure creates as many float pages as it can until there are no more floats left to fill a float page. Float page production is controlled by the parameter `\floatpagefraction`, which specifies the minimum fraction of the page that must be occupied by float(s)—by default, half the page. In the standard settings every float is allowed to go on a float page (the default specifier is `tbp`), so this setting means that every float that is a tiny bit larger than half the page is allowed to go on a float page by itself. Thus, by enlarging its value, you can prevent half-empty float pages.

However, enlarging the value of `\floatpagefraction` makes it more difficult to produce float pages. As a result, some floats may be deferred, which in turn prevents other floats from being placed. For this reason it is often better to specify explicitly the allowed placements (for example, by saying `\begin{figure}[tb]`) for the float that creates the problem.

Another common reason for ending up with all floats at the end of your chapter is use of the bottom placement specifier, `[b]`. It indicates that the only acceptable place for a float is at the bottom of a page. If your float happens to be larger than `\bottomfraction` (which is by default quite small), then this float cannot be placed. This will also prevent all floats of the same type from being placed. The same problem arises if only `[h]` or `[t]` is specified and the float is too large for the remainder of the page or too large to fit `\topfraction`.

In calculating these fractions, L<sup>A</sup>T<sub>E</sub>X will take into account the separation (i.e., `\textfloatsep`) between floats and main text. By enlarging this value, you automatically reduce the maximum size a float is allowed to have to be considered as a candidate for placement at the top or bottom of the page.

In general, whenever a lot of your floats end up at the end of the chapter, look at the first ones to see whether their placement specifiers are preventing them from being properly placed.

## 6.2 Float placement control

*Flosts always after their call-out*

The float placement algorithm prefers to put floats at the top of the page, even if it means placing them before the actual reference. This outcome is not always acceptable but there is no easy cure for this problem short of substantially changing L<sup>A</sup>T<sub>E</sub>X's algorithm. The `flafter` package (by Frank Mittelbach) makes this change, thereby ensuring that floats are never placed before their references.

Sometimes, less drastic solutions might be preferred. For example, if the float belongs to a section that starts in the middle of a page but the float is positioned at the top of the page, the float will appear as if it belongs to the previous section. You might want to forbid this behavior while still allowing floats to be placed on the top of the page in other situations. For this purpose L<sup>A</sup>T<sub>E</sub>X offers you the following command.

\suppressfloats [*placement*]

The optional argument *placement* can be either *t* or *b*. If the command `\suppressfloats` is placed somewhere in the document, then on the current page any following floats for the areas specified by *placement* are deferred to a later page. If no *placement* parameter is given, all remaining floats on the current page are deferred. For example, if you want to prevent floats from moving backward over section boundaries, you can redefine your section commands in the following way:

```
\renewcommand\section{\suppressfloats[t]%
  \@startsection{section}{...}{...}{...} ... }
```

Possible arguments to `\@startsection` are discussed in Section 2.2.2.

Another way to influence the placement of floats in L<sup>A</sup>T<sub>E</sub>X is to specify a *!* in conjunction with the placement specifiers *h*, *t*, and *b*. The placement of floats on float pages is not affected by this approach. This means that for this float alone, restrictions given by the settings of the parameters described earlier (e.g., `\textfraction`) are ignored. Thus, such a float can be placed in the designated areas as long as neither of the following two restrictions is violated:

- The float fits on the current page; that is, its height plus the material already contributed to the page does not exceed `\textheight`.
- There are no deferred floats of the same type.

All other restrictions normally active (e.g., the number of floats allowed on a page) are ignored. For example, if you specify `[!b]` this float can be placed on the bottom of the page even if it is larger than the maximum size specified by `\bottomfraction`. Also, any `\suppressfloats` commands are ignored while processing this float.

The order of the given specifiers is irrelevant, and all specifiers should be given at most once. For example, `[bt]` is the same as `[tb]` and thus does *not* instruct L<sup>A</sup>T<sub>E</sub>X to try to place the float at the bottom and only then try to place it on the top. L<sup>A</sup>T<sub>E</sub>X always uses the following order of tests until an allowed placement is found:

1. If *!* is specified, ignore most restrictions as described above and continue.
2. If *h* is specified, try to place the float at the exact position. If this fails and no other position was specified, change the specifier to *t* (for a possible placement on the next page).
3. If *t* is specified, try to place it on the top of the current page.
4. If *b* is specified, try to place it on the bottom of the current page.
5. If *p* is specified, try to place it on a float page (or float column) when the current page (or column) has ended.
6. Steps 3 and 4 are repeated if necessary at the beginning of each subsequent page, followed by Step 5 at its end.

*Algorithm to determine allowed placement*

Sometimes you will find that L<sup>A</sup>T<sub>E</sub>X's float placement specifiers are too restrictive. You may want to place a float exactly at the spot where it occurs in the input file—that is, you do not want it to float at all. It is a common misunderstanding that specifying [h] means "here and nowhere else". Actually, that specifier merely directs L<sup>A</sup>T<sub>E</sub>X to *do its best* to place the float at the current position. If there is not enough room left on the page or if an inline placement is forbidden because of the settings of the style parameters (see Section 6.1), then L<sup>A</sup>T<sub>E</sub>X will ignore this request and try to place the float according to any other specifier given. Thus, if [ht] is specified, the float will appear on the top of some later page if it does not fit onto the current one. This situation can happen quite often if the floats you try to place in the middle of your text are moderately large and are thus likely to fall into positions where there is not enough space on the page for them. By ignoring an h and trying other placement specifiers, L<sup>A</sup>T<sub>E</sub>X avoids overly empty pages that would otherwise arise in such situations.

[h] does not mean "here"

In some cases you might prefer to leave large gaps on your pages. For this reason the package `float` provides you with an [H] specifier that means "put the float here"—period. It is described in Section 6.3.1.

### 6.2.1 `placeins`—Preventing floats from crossing a barrier

Donald Arseneau wrote the package `placeins` to enable you to prevent floats from moving past a certain point in the output document by introducing a `\FloatBarrier` command. With the `placeins` package, when such a command is encountered, all floats that are not yet placed will be transferred to the output stream. This approach is useful if you want to ensure that all floats that belong to a section are placed before the next section starts.

For example, you could redefine the sectioning command and introduce the `\FloatBarrier` command in its definition inside the `\@startsection` command (see Section 2.2.2), as shown here:

```
\makeatletter % needed if used in the preamble
\renewcommand\section{\@startsection
{section}{1}{0mm}%
{} % name, level, indent
{-\baselineskip} % beforeskip
{0.5\baselineskip} % afterskip
{\FloatBarrier\normalfont\Large\bfseries}}% style
\makeatother % needed if used in the preamble
```

The author of `placeins` anticipated that users might often want to output their floats before a new section starts, so his package provides the package option `section`, which automatically redefines `\section` to include the `\FloatBarrier` command. However, by itself this option forces all floats to appear *before* the next section material is typeset, since the `\FloatBarrier` prevents a float from a current section from appearing below the start of the new section, even if some material of the current section is present on the same page.

If you want to allow floats to pass the `\FloatBarrier` and appear at the bottom of a page (i.e., in a new section), specify the option below. To allow floats to pass it in the opposite direction and appear on the top of the page (i.e., in the previous section), specify the option above.

*Turning the barrier  
into a membrane*

When using the option `verbose` the package shows processing information on the terminal and in the transcript file.

### 6.2.2 `afterpage`—Taking control at the page boundary

The `afterpage` package (by David Carlisle) implements a command `\afterpage` that causes the commands specified in its argument to be expanded after the current page is output. Although its author considers it “a hack that not even always works” (for example, `\afterpage` will fail in `twocolumn` mode), it has a number of useful applications.

Sometimes L<sup>A</sup>T<sub>E</sub>X’s float positioning mechanism gets overloaded, and all floating figures and tables drift to the end of the document. You may flush out all the unprocessed floats by issuing a `\clearpage` command, but this tactic has the effect of making the current page end prematurely. The `afterpage` package allows you to issue the command `\afterpage{\clearpage}`. It will let the current page be filled with text (as usual), but then a `\clearpage` command will flush out all floats before the next text page begins.

*Preventing floats at  
the end of the  
document*

With the multipage `longtable` environment (see Section 5.4.2), you can experience problems when typesetting the text surrounding the long table, and it may be useful to “float” the `longtable`. However, because such tables can be several pages long, it may prove impossible to hold them in memory and float them in the same way that the `table` environment is floated. Nevertheless, if the table markup is in a separate file (say `ltfile.tex`) you can use one of the following commands:

```
\afterpage{\clearpage\input{ltfile}}
\afterpage{\clearpage\input{ltfile}\newpage}
```

*Floating multipage  
tables*

The first form lets text appear on the same page at the end of the `longtable`. The second ensures that the surrounding text starts again on a new page.

The `\afterpage` command can be combined with the `float` package and the `[H]` placement specifier, as explained at the end of Section 6.3.1.

### 6.2.3 `endfloat`—Placing figures and tables at the end

Some journals require figures and tables to be separated from the text and grouped at the end of a document. They may also want a list of figures and tables to precede them and potentially require markers indicating the original places occupied by the floats within the text. This can be achieved with the `endfloat` package (by James Darrell McCauley and Jeffrey Goldberg), which puts figures and tables

by themselves at the end of an article into sections titled “Figures” and “Tables”, respectively.

The `endfloat` package features a series of options to control the list of figures and tables, their section headings, and the markers left in the text. A list of available options follows.

`figlist/nofiglist` Produce (default) or suppress the list of figures.

`tablist/notablist` Produce (default) or suppress the list of tables.

`lists/nolists` Produce or suppress the list of figures and the list of tables (shorthand for the combination of the previous two option sets).

`fighead/nofighead` Produce or omit (default) a section heading before the collection of figures. The section headings text is given by `\figuresection` and defaults to the string “Figures”.

`tabhead/notabhead` Produce or omit (default) a section heading before the collection of tables. The section headings text is given by `\tablesection` and defaults to the string “Tables”.

`heads/noheads` Produce or omit a section heading before the collection of figures and before the collection of tables (shorthand for the combination of the previous two option sets).

`markers/nomarkers` Place (default) or omit markers in text.

`figuresfirst/tablesfirst` Put all figures before tables (default), or vice versa.

The package offers the hooks `\AtBeginFigures`, `\AtBeginTables`, and `\AtBeginDelayedFloats` to control the processing of the collected floats. For instance, the instruction `\AtBeginTables{\cleardoublepage}` ensures that the delayed tables will start on a recto page.

When the floats are finally typeset, the command `\efloatseparator` is executed after each float. By default, it is defined to be `\clearpage`, which forces one float per page. If necessary, it can be redefined with `\renewcommand`.

By default, the package indicates the original position of a float within the text by adding lines such as “[Figure 4 about here.]” at the approximate place. These notes can be turned off by specifying the `nomarkers` option when loading the package. The text and the formatting of the notes, which are defined via the commands `\figureplace` and `\tableplace`, can be changed with `\renewcommand`. For example, they might be adapted to a different language (the package does not support `babel` parameterization). A sample redefinition for French could look as follows:

```
\renewcommand\figureplace
  {\begin{center}[La figure~\thepostfig\ approx.\ ici.]\end{center}}
\renewcommand\tableplace
  {\begin{center}[La table~\theposttbl\ approx.\ ici.]\end{center}}
```

Within the replacement text `\thepostfig` and `\theposttbl` reference the current

figure or table number, respectively. Such redefinitions can, for example, be put in the package configuration file `endfloat.cfg` that, if present, is loaded automatically by the package (with the usual caveat of nonportability).

By default, the delayed floats are processed when the end of the document is reached. However, in some cases one might wish to process them at an earlier point—for example, to display them at the end of each chapter. For this purpose `endfloat` offers the command `\processdelayedfloats`, which will process all delayed floats up to the current point. The float numbering will continue by default, so to restart numbering one has to reset the corresponding counters (details are given in the package documentation).

The `endfloat` package file creates two extra files with the extensions `.fff` and `.ttt` for storing the figure and table floats, respectively. As the environment bodies are written verbatim to these files, it is important that the `\end` command, (e.g., `\end{figure}`), always appears on a line by itself (without any white space) in the source document; otherwise, it will not be recognized. For the same reason the standard environment names (i.e., `figure`, `table`, and their starred forms) will be recognized only if they are directly used in the document. If they are hidden inside other environments recognition of the environment `\end` tag will fail.

By default, nonstandard float environments, such as the `sidewaysfigure` and `sidewaystable` environments of the `rotating` package, are not supported. It is possible, however, to extend the `endfloat` package to recognize such environments as well. As an example the distribution contains the file `efxmpl.cfg`, which extends `endfloat` to cover the environments of the `rotating` package. To become operational it should be included (copied) into `endfloat.cfg` so that its code is automatically loaded.

*Premature output*

*Caveats*

## 6.3 Extensions to L<sup>A</sup>T<sub>E</sub>X's float concept

By default, L<sup>A</sup>T<sub>E</sub>X offers two types of horizontally oriented float environments, `figure` and `table`. For many documents these prove to be sufficient; in other cases additional features are needed. In this section we now look at packages that extend this basic tool set to cover more complex cases.

The `float` package offers ways to define new float types and also provides one way to prevent individual floats from floating at all. A different approach to the latter problem is given by the `caption` package.

The last two packages described in this section, `rotating` and `rotfloat`, allow the rotation of the float content, something that might be necessary for unusually large float objects.

### 6.3.1 float—Creating new float types

The `float` package by Anselm Lingnau improves the interface for defining floating objects such as figures and tables in L<sup>A</sup>T<sub>E</sub>X. It adds the notion of a “float style” that

governs the appearance of floats. New kinds of floats may be defined using the `\newfloat` command.

`\newfloat{type}{placement}{ext}[within]`

The `\newfloat` command takes four arguments, three mandatory and one optional, with the following meanings:

*type* “Type” of the new class of floats, such as `program`. Issuing a `\newfloat` declaration will make the environments *type* and *type\** available.

*placement* Default placement parameters for the given class of floats (combination of L<sup>A</sup>T<sub>E</sub>X’s `t`, `b`, `p`, and `h` specifiers or, alternatively, the `H` specifier).

*ext* File name extension of an auxiliary file to collect the captions for the new float class being defined.

*within* Optional argument specifying whether floats of this class will be numbered within some sectional unit of the document. For example, if the value of *within* is equal to `chapter`, the floats will be numbered within chapters (in standard L<sup>A</sup>T<sub>E</sub>X, this is the case for figures and tables in the `report` and `book` document classes).

*The style of the float class* The `\floatstyle` declaration sets a default float style that will be used for all float types that are subsequently defined using `\newfloat`, until another `\floatstyle` command is specified. Its argument is the name of a float style, and should be one of the following predefined styles:

`plain` The float style L<sup>A</sup>T<sub>E</sub>X usually applies to its floats—that is, nothing in particular. The only difference is that the caption is typeset below the body of the float, regardless of where it is given in the input markup.

`plaintop` Same style as the `plain` float style except that the caption is placed at the top of the float.

`boxed` The float body is surrounded by a box with the caption printed below.

`ruled` The float style is patterned after the table style of *Concrete Mathematics* [59]. The caption is printed at the top of the float, surrounded by rules; another rule finishes off the float.

The float styles define the general layout of the floats, including the formatting of the caption. For example, the `ruled` style sets the caption flush left without a colon, while other styles center the caption and add a colon after the number. One has to be careful when mixing different float styles in one document so as not to produce typographic monsters.

Even though the package does not offer a user-level interface for defining new float styles, it is fairly easy to add new named styles. For details refer to the package documentation in `floats.dtx`.

The next example shows the declarations for two “nonstandard” new float types, `Series` and `XMLexa`. The former are numbered inside sections and use a “boxed” style, and the latter are numbered independently and use a “ruled” style (typographically this combination is more than questionable).

The introductory string used by L<sup>A</sup>T<sub>E</sub>X in the captions of floats for a given *type* can be customized using the declaration `\floatname{type}{floatname}`. “XML Listing” is used for `XMLexa` floats in the example below. By default, a `\newfloat` command sets this string to its *type* argument if no other name is specified afterwards (shown with the `Series` float environment in the example).

## 1 New float environments

Some text for our page that might get reused over and over again.

### XML Listing 1 A simple XML file

```
<XMLphrase>Great fun!</XMLphrase>
```

Some text for our page that might get reused over and over again.

### XML Listing 2 Processing instruction

```
<?xml version="1.0"?>
```

Some text for our page that might get reused over and over again. Some text for our page that might get reused over and over again.

$$e = 1 + \sum_{k=1}^{\infty} \frac{1}{k!}$$

**Series 1.1:** Euler's constant

```
\usepackage{float}
\floatstyle{boxed}
\newfloat{Series}{b}{los}[section]
\floatstyle{ruled}
\newfloat{XMLexa}{H}{lox}
\floatname{XMLexa}{XML Listing}
\newcommand\xmlcode[1]{\texttt{\#1}}
\newcommand\sample{Some text for our page
that might get reused over and over again. }
\section{New float environments}
\sample
\begin{XMLexa} \caption{A simple XML file}
\xmlcode{<XMLphrase>Great fun!</XMLphrase>}
\end{XMLexa}
\sample
\begin{XMLexa}
\caption{Processing instruction}
\xmlcode{<?xml version='1.0'?>}
\end{XMLexa}
\sample
\begin{Series}
\caption{Euler's constant}
\,[e = 1 + \sum^{\infty}_{k=1} \frac{1}{k!}]
\end{Series}
\sample
```

[ 6-3-1 ]

The command `\listof{type}{title}` produces a list of all floats of a given class. It is the equivalent of L<sup>A</sup>T<sub>E</sub>X's built-in commands `\listoffigures` and `\listoftables`. The argument *type* specifies the type of the float as given in the `\newfloat` command. The argument *title* defines the text of the title to be used to head the list of the information associated with the float elements, as specified by the `\caption` commands.

*Listing the captions  
of a float class*

The following example is a repetition of Example 6-3-1 on the preceding page (source only partially shown) with two `\listof` commands added.

## XML Listings

- |   |                                  |
|---|----------------------------------|
| 1 | A simple XML file . . . . .      |
| 2 | Processing instruction . . . . . |

```
\usepackage{float}
% Float types ‘‘Series’’ and ‘‘XMLexa’’ and
% commands \xmlcode and \sample as defined
% in previous example
\listof{XMLexa}{XML Listings}
\listof{Series}{List of Series}
\section{New float environments}
\sample
\begin{XMLexa} \caption{A simple XML file}
\xmlcode{<XMLphrase>Great fun!</XMLphrase>}
\end{XMLexa}
... text omitted ...
```

6-3-2

## List of Series

- |     |                            |
|-----|----------------------------|
| 1.1 | Euler’s constant . . . . . |
|-----|----------------------------|

3

## 1 New float environments

Some text for our page which might get reused over and over again.

*Customizing LaTeX’s standard float types*

`\newfloat`, as they already exist when the `float` package is loaded. To solve this problem the package offers the declaration `\restylefloat{type}`, which selects the current float style (specified previously with a `\floatstyle` declaration) for floats of this *type*.

For the same reason there exists the `\floatplacement{type}{placement}` declaration, which can be used to change the default placement specifier for a given float *type* (e.g., `\floatplacement{table}{tp}`). In the following example, both `figure` and `table` have been customized (not necessarily for the better) to exhibit the usage of these declarations.



Figure 1: Sample figure

## 1 Customizing standard floats

Some text for our page that might get reused over and over again. Some text for our page that might get reused over and over again.

**Table 1** Sample table

AAAA	BBBB	123
CCC	DDD	45

```
\usepackage{graphicx, float}
\floatstyle{boxed} \restylefloat{figure}
\floatstyle{ruled} \restylefloat{table}
\floatplacement{table}{b}
% \sample as previously defined
\section{Customizing standard floats}
\sample
\begin{table}
\begin{tabular}{@{}llr}
AAAA&BBBB&123\\CCC&DDD&45\end{tabular}
\caption{Sample table}
\end{table}
\sample
\begin{figure} \centering
\includegraphics[width=12mm]{rosette.ps}
\caption{Sample figure}
\end{figure}
```

6-3-3

Modeled after David Carlisle’s `here` package, the `float` package adds the `[H]` placement specifier which means “place the float Here regardless of any surround-

ing conditions". It is available for all float types, including L<sup>A</sup>T<sub>E</sub>X's standard `figure` and `table` environments. The `[H]` qualifier must always be used on a stand-alone basis; e.g., `[Hbpt]` is illegal.

If there is not enough space left on the current page, the float will be printed at the top of the next page together with whatever follows, even if there is still room left on the current page. It is the authors' responsibility to place their `H` floats in such a way that no large patches of white space remain at the bottom of a page. Moreover, one must carefully check the order of floats when mixing standard and `[H]` placement parameters. Indeed, a float with a `[t]` specifier, for example, appearing before one with an `[H]` specifier in the input file might be incorrectly positioned after the latter in the typeset output, so that, for instance, Figure 4 would precede Figure 3.

All float placement specifiers are shown together in the following example.

6-3-4

<code>t</code>	Top of page
<code>b</code>	Bottom of page
<code>p</code>	Page of floats
<code>h</code>	Here, if possible
<code>H</code>	Here, always

Table 1: Float placement specifiers

With "h" instead of

```
\usepackage{float, array}
All float placement specifiers are
shown together in the following example.
\begin{table}[H]
\begin{tabular}{>{\ttfamily}c}
t & Top of page \\
b & Bottom of page \\
p & Page of floats \\
h & Here, if possible \\
H & Here, always
\end{tabular}
\caption{Float placement specifiers}
\end{table}
With "h" instead of the "H" specifier
this text would have appeared before the
table in the current example.
```

6

7

In combination with the `placeins` and `afterpage` packages described in Sections 6.2.1 and 6.2.2, respectively, an even finer control on the placement of floats is possible. Indeed, in some cases, although you specify the placement parameter as `[H]`, you do not really mean "at this point", but rather "somewhere close". This effect is achieved by using the `\afterpage` command:

```
\afterpage{\FloatBarrier\begin{figure}[H]\dots\end{figure}}
```

The `\FloatBarrier` command ensures that all dangling floats are placed first at a suitable point (due to `\afterpage` without producing a huge gap in the text), thereby solving the sequencing problem, described above. The `[H]` float is then immediately placed afterwards. If you use `\clearpage` instead of `\FloatBarrier`, it would come out on top of the next page instead.

### 6.3.2 `caption`—For nonfloating figures and tables

An alternative to specifying the `[H]` option with the various float environments, as described in the previous section, is to define captioning commands that typeset and are entered into the "List of Figures" or "List of Tables" just like L<sup>A</sup>T<sub>E</sub>X's

standard `figure` and `table` environments. This functionality is provided by the `caption` package (discussed in more detail in Section 6.5.1).

```
\captionof{type}[short-text]{text}      \captionof*[type]{text}
```

This command works analogously to L<sup>A</sup>T<sub>E</sub>X's `\caption` command, but takes an additional mandatory argument to denote the float *type* it should mimic. It can be used for any nonfloating material that should get a (numbered) caption whose text will also be added into the list of figures or list of tables. The starred form suppresses both the number and the "List of..." entry.

The following example shows a normal figure and its nonfloating variant used together. In such a case there is always the danger that a floating figure will travel past its nonfloating counterparts. In the example we force this situation by pushing the floating figure to the bottom of the page. As a result, the numbering gets out of sync. One has to watch out for this problem when mixing floating and nonfloating objects.

## List of Figures

2	Fake LOF entry . . . . .	1
1	Standard figure . . . . .	1

### 1 Various kinds of figures

Here we mix standard and nonfloating figures.

Figure II

Figure 2: Nonfloating figure

As Figure 1 is forced to the bottom with an optional `[b]` argument it passes Figure 2 and the numbering

Figure I

Figure 1: Standard figure

```
\usepackage{caption}
\listoffigures
\section{Various kinds of figures}
Here we mix standard and nonfloating figures.
\begin{figure}[b] \centering
\fbox{Figure I}
\caption{Standard figure} \label{fig:I}
\end{figure}
\begin{center}
\fbox{Figure II} \\
\captionof{figure}[Fake LOF entry]
{Nonfloating figure}
\label{fig:II}
\end{center}
As Figure \ref{fig:I} is forced to
the bottom with an optional \texttt{[b]}
argument it passes Figure \ref{fig:II}
and the numbering gets out of sync.
```

6-3-5

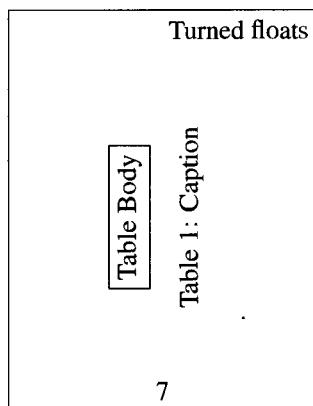
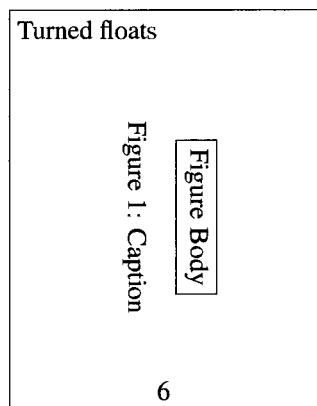
### 6.3.3 rotating—Rotating floats

Sometimes it is desirable to turn the contents of a float sideways, by either 90 or 270 degrees. As T<sub>E</sub>X is not directly capable of performing such an operation, it needs support from an output device driver. To be as device independent as possible, L<sup>A</sup>T<sub>E</sub>X encapsulates the necessary operations in the packages `graphics` and `graphicx` (see Section 10.2). One of the earliest packages that used this interface was the `rotating` package written by Sebastian Rahtz and Leonor Barroca.<sup>1</sup>

<sup>1</sup>In fact, its original release predates the development of the `graphics` interface. It was later reimplemented as an extension of this interface.

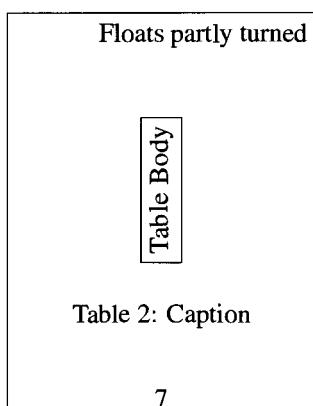
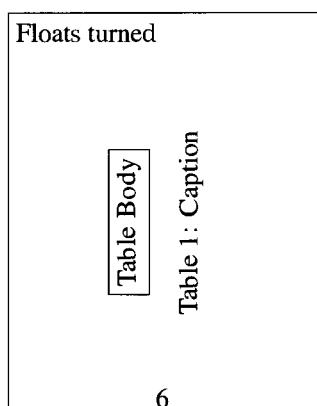
The rotating package implements two environments, `sidewaysfigure` and `sidewaystable`, for turning whole floats sideways. These environments automatically produce page-sized floats, or more exactly column-sized floats (if used in `twocolumn` mode). Starred forms of these environments, which span both columns in `twocolumn` mode, exist as well.

By default, the floats are turned in such a way that they can be read from the outside margin, as you can see in the next example. If you prefer your floats to be always turned in the same way, you can specify one of the package options `figuresright` or `figuresleft`.



```
\usepackage{rotating}
\usepackage{fancyhdr}
\pagestyle{fancy}
\fancyhead[RO,LE]{Turned floats}
\begin{sidewaysfigure}
  \centering \fbox{Figure Body}
  \caption{Caption}
\end{sidewaysfigure}
\begin{sidewaystable}
  \centering \fbox{Table Body}
  \caption{Caption}
\end{sidewaystable}
```

The package also defines a number of environments for rotating arbitrary objects, such as `turn` or `rotate` (to rotate material with or without leaving space for it); see Section 10.3.4. Directly relevant to floats is the `sideways` environment, which enables you to turn the float body while leaving the caption untouched. It is used in the following example, which also exhibits the result of the `figuresright` option (which, despite its name, acts on `sidewaysfigure` and `sidewaystable`).



```
\usepackage[figuresright]{rotating}
\usepackage{fancyhdr}
\pagestyle{fancy}
\fancyhead[LE]{FLOATS TURNED}
\fancyhead[RO]{FLOATS PARTLY TURNED}
\begin{sidewaystable} \centering
  \fbox{Table Body} \caption{Caption}
\end{sidewaystable}
\begin{table} \centering
  \begin{sideways}
    \fbox{Table Body}
  \end{sideways}
  \caption{Caption}
\end{table}
```

Instead of turning the whole float or the float body, it is sometimes more appropriate to turn only the caption. This ability is supported by the rotating package through the `\rotcaption` command. Unfortunately, the layout produced by this command is hard-wired but can be customized through the `caption` package whose features are discussed in Section 6.5.1.

### 6.3.4 `rotfloat`—Combining float and rotating

To extend the new float styles, as introduced by the `float` package, with the `sidewaysfigure` and `sidewaystable` environments defined in the rotating package, you can use Axel Sommerfeldt's `rotfloat` package. It allows you to build new floats, which are rotated by 90 or 270 degrees.

The `rotfloat` package offers identical options to the `rotating` package. Internally, for every float *type*, `rotfloat` defines an additional environment with the name `sidewaystype` and its corresponding starred form. For instance, when you write

```
\newfloat{XMLLexa}{tbp}{lox} \floatname{XMLLexa}{XML Listing}
```

four environments become available: `XMLLexa`, `XMLLexa*`, `sidewaysXMLLexa`, and `sidewaysXMLLexa*`. Similarly, the commands for redefining the `table` or `figure` environments, for example,

```
\floatstyle{boxed} \restylefloat{table}
```

will restyle not only the `table` and `table*` environments, but also the environments `sidewaystable` and `sidewaystable*`.

## 6.4 Inline floats

In `TEX`'s typesetting model, text is first broken into paragraphs on a vertically oriented galley (or scroll). Once enough material is collected in this way `TEX` invokes its output routine, which chops off the first part of the galley, attaches running headers and footers as specified, and outputs the result in the `.dvi` file. It then restarts collecting text and breaking it into paragraphs to refill the galley.

As a consequence of this processing model, it is relatively easy to implement a float mechanism in which floats span the full width of the page or at least the full width of individual columns. Unfortunately, it is nearly impossible to have floats that occupy only parts of a text column and have the text flow around them. The reason being that when the paragraphs are broken into lines, their final positions are not yet known. It is therefore impossible to direct the paragraph builder to leave holes for the float objects if a later part of the process will decide on their final placement. In contrast, placing floats at the top or the bottom of a page (or column) only directs the output routine to chop off less material from the assembled galley without otherwise manipulating the galley content.

Because of this processing model, the production of inline floats with text flowing around the float object has to take place during the paragraph-generating phase. The best outcome that packages can currently achieve is to ensure that the inline floats do not fall off the page (by measuring the amount of material already assembled on the galley to decide whether there is enough space to fit in the inline float with its surrounding paragraph(s)).

Such an algorithm is, for example, implemented by the `wrapfig` package. Because the package's inline floats only "float" very little in comparison to standard floats, mixing both types can result in the float numbering getting out of sequence.<sup>1</sup> Most relevant packages leave the placement decisions completely to the user because the automatic solution comes out wrong in many cases, so that it is not worth supplying it in the first place.

For this book we have chosen a total of three packages that are representative of what is available in this area. We have already discussed one such package (`picinpar`) in Section 3.1.14; two more are introduced here. The `wrapfig` package supports figures and tables and offers some support for automatic placement. The `picins` package allows precise control over the placement of inline figures and for this particular task can be quite interesting. Unlike other packages in this area, it does not support inline tables.

All packages have some problems so that it might be worthwhile to explore other possibilities such as `floatfl` by Mats Dahlgren (an extension of the `floatfig` package by Thomas Kneser), which works together with the `multicol` package. A good starting point to look for other packages is Graham Williams' *TeX online catalogue* [169].

#### 6.4.1 `wrapfig`—Wrapping text around a figure

The package `wrapfig` (by Donald Arseneau) defines the `wrapfigure` and `wraptable` environments. These environments allow one to typeset a narrow float at the edge of some text, and then make the text wrap around it. Both produce captions with the standard caption layout for figures and tables. Although the environments have some limited ability to "float", no provision is made to synchronize them with regular floats. Thus, one must be aware that they may be printed out of sequence with standard floats.

```
\begin{wrapfigure}[nlines]{placement}[overhang]{width}
```

The `wrapfigure` and `wraptable` environments have two mandatory and two optional arguments with the following meanings:

*nlines* (optional) The number of narrow lines needed for the float (normally calculated automatically). Each display equation counts as three lines.

<sup>1</sup>In theory, one could do better and properly synchronize both types, although the coding would probably be quite difficult.

*placement* Horizontal placement of the float, specified as *one* of the following letters: r or R (right side of the text), and l or L (left side of the text). There is no option for centering the float. For a two-sided document, the placement can alternatively be specified via i or I (inside edge) and o or O (outside edge). This refers to the inside and outside of the whole page, not to individual columns. In each case the uppercase variant allows the figure or table to float, while the lowercase variant puts it “exactly here”.

*overhang* (optional) Overhang of the float into the margin (default 0pt).

*width* Width of the figure or table. Specifying 0pt has a special meaning, such that the “natural width” will be used as the wrapping width. The caption is then typeset to the wrapping width. If the figure is wider than the space allotted, an “overfull box” will be generated and the figure or table contents can overwrite the wrapping text.

$\text{\LaTeX}$  will wrap surrounding text around the figure or table, leaving a gap of `\intextsep` at the top and bottom and `\columnsep` at the side, thereby producing a series of shortened text lines beside the figure. The size of the hole made in the text is the float width plus `\columnsep` minus the *overhang* value.

$\text{\LaTeX}$  calculates the number of short lines needed based on the height of the figure and the length `\intextsep`. This guess may be overridden by specifying the first optional argument (*nlines*), which is the desired number of shortened lines. It can be useful when the surrounding text contains extra vertical spacing that is not accounted for automatically.

Our first example shows a wrapped table, 4cm wide and placed at the left side of the paragraph. The package calculated a wrapping of 5 lines, which would have left a lot of empty space below the caption, so we explicitly selected 4 lines of wrapping instead. The figure is referenced using  $\text{\LaTeX}$ ’s standard `\label` and `\ref` commands.

Wrapped Table	Some text for our page that is reused over and over again.
Table 1: The Caption	Some text for our page that is reused over and over again. Reference to Table 1. Some text for our page that is reused over and over again.

```
\usepackage{wrapfig}
% \sample as before
\begin{wraptable}[4]{1}{4cm}
\centering\fbox{Wrapped Table}
\caption{The Caption}\label{T}
\end{wraptable}
\sample \sample Reference to Table~\ref{T}.
\sample
```

6-4-1

The `wrapfigure` and `wraptable` environments should not be used inside another environment (e.g., `list`). They do work in `twocolumn` page layout (provided the column width is wide enough to allow inline floats).

Generally  $\text{\LaTeX}$  will not be able to move `wrapfigure` and `wraptable` environments to their optimal places, so it is up to you to position them in the best fashion. It is best to wait to do so until just before printing your *final copy*, because

any changes to the document can ruin their careful positioning. Information about float processing by `wrapfig` is written to the log file if you specify the `verbose` option. Here are some rules for good placement:

- The environments should be placed so as to not run over a page boundary and must not be placed in special places like lists.
- Only ordinary text should have to flow past the figure but not a section title or large equations. Small equations are acceptable if they fit.
- It is convenient to place `\begin{wrapfigure}` or `\begin{wraptable}` just after a paragraph has ended. If you want to start in the middle of a paragraph, the environment must be placed between two words where there is a natural line break.

Our second example displays a figure that is set to its natural width (last argument `0pt`), but extends 20% into the left margin (specified by the optional argument). Instead of using the special unit `\width`, denoting the natural float width in this case, one can, of course, use some explicit dimension such as `30pt`. The effect of this choice can be clearly seen by looking at the way the paragraph text is typeset below the picture when the text wrapping ends. As the example also shows, wrapping continues even across paragraph boundaries if necessary.

The formatting of the caption can be influenced by combining `wrapfig` with packages like `caption`, although an option like `centerlast` may not be the appropriate choice in narrow measures.

The starting place for the `wrapfigure` environment was manually determined in the current example by first setting the text without the figure to find the linebreaks.

This is a “`wrapfigure`”.

**Figure 1:** An example of the `wrapfigure` environment

Some text for our page that is reused over and over again. Some text for our page that is reused over and over again. Some text for our page that is reused over and over again.

6-4-2

```
\usepackage{wrapfig}
\usepackage[labelfont={sf,bf},
           justification=centerlast]{caption}
% \sample as before
The starting place for the wrapfigure
environment was manually determined in
the current ex-
\begin{wrapfigure}[7]{1}[0.2\width]{0pt}
  \centering
  \fbox{This is a ``wrapfigure''.}
  \caption{An example of the
    \texttt{wrapfigure} environment}
\end{wrapfigure}
\sample \sample \sample
```

In the preceding example we specified an `overhang` length explicitly. The overhang width can also be specified globally for all `wrapfig` environments by setting the `\wrapoverhang` length with L<sup>A</sup>T<sub>E</sub>X's `\setlength` command to a non-zero

value. For example, to have all wrap figures and tables use the space reserved for marginal notes, you could write

```
\setlength \wrapoverhang{\marginparwidth}
\addtolength\wrapoverhang{\marginparsep}
```

New “wrapping” environments for additional float types (as defined via the `float` package) with the same interface and behavior as `wrapfigure` or `wraptable` may be easily added, or directly invoked, using the `wrapfloat` environment:

```
\newfloat{XML}{tbp}{lox}
\newenvironment{wrapXML}{\begin{wrapfloat}{XML}}{\end{wrapfloat}}
```

You can find other ways to fine-tune the behavior of `wrapfig` by reading the implementation notes at the end of the `wrapfig.sty` package file.

#### 6.4.2 picins—Placing pictures inside the text

The `picins` package (by Joachim Bleser and Edmund Lang) defines the `\parpic` command, which allows you to place a “picture” at the left or right of one or more paragraphs with the paragraph text flowing around the picture.

`\parpic{w,h}{x-o,y-o}[opt][pos]{pict}`

*w,h* (optional) Width and height of the picture. The text lines that flow around the picture are set in a paragraph whose lines are shorter than the text width by an amount *w*. The height *h* is used to calculate the number of lines of text that will flow in this manner.

If the argument is not specified, the actual picture size (“bounding box”) is used, if it can be calculated by L<sup>A</sup>T<sub>E</sub>X. Otherwise, an error results.

*x-o,y-o* (optional) The *x* and *y* offsets of the picture with respect to the upper-left corner of its bounding box (positive *x-o* yields a displacement to the right; positive *y-o* moves the picture downward). If the argument is absent, the picture is positioned using the *pos* specification.

*opt* (optional) Placement and box characteristics of picture, given as a pair of *one* positional and *one* frame specifier.

The *positional* specifiers are *l* (*left*) picture at left of paragraph and *r* (*right*) picture at right of paragraph.

The *frame* specifiers are *d* (*dash*) picture surrounded by dashed lines; *f* (*frame*) picture surrounded by full lines; *o* (*oval*) picture frame with rounded corners; *s* (*shadow*) picture surrounded by shadow box; and *x* (*box*) picture surrounded by “three-dimensional” box. When no option is specified, the picture is placed at the left of the paragraph.

*pos* (optional) Position of the picture inside its frame, given as one horizontal specifier, one vertical specifier, or a pair of horizontal and vertical specifiers. Possible *horizontal* specifiers are *l* (*left*) picture at left of frame and *r* (*right*) picture at right of frame. If no horizontal specifier is given, the picture is centered horizontally in its frame.

Possible *vertical* specifiers are *t* (*top*) picture at top of frame and *b* (*bottom*) picture at bottom of frame. If no vertical specifier is given, the picture is centered vertically in its frame.

If the offset argument *x-o,y-o* is present, the *pos* argument is ignored.

*pict* The source of the picture. It can be any L<sup>A</sup>T<sub>E</sub>X construct.

The following examples show various ways to place a picture inside a paragraph. We also introduce some other commands provided by the `picins` package to fine-tune the visual presentation of the typeset result.

We start by using `picins`'s default setting, where the width and height of the contents are automatically calculated. In that case the “picture” is placed at the left of the paragraph. This paragraph has a normal indentation: if this effect is not desired, one has to start it with `\noindent`. The second part of the example pulls in an Encapsulated PostScript (EPS) picture and lets text flow around it. In this case the natural dimensions of the picture are read from the `BoundingBox` comment in the EPS source file. We added a dashed frame for more clarity.

### BOX

Some text for our page that is reused over and over again. Some text for our page that is reused over and over again.



Some text for our page that is reused over and over again. Some text for our page that is reused over and over again.

Some text for our page that is reused over and over again. Some text for our page that is reused over and over again.

[ 6-4-3 ]

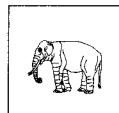
```
\usepackage{picins,graphicx}
\newcommand\sample{Some text for our page
    that is reused over and over again. }
\newcommand\FIG{\includegraphics
    [width=14mm]{cat}}
\parpic{\fbox{Large\sshape Box}}
\sample\sample\par
\parpic[d]{\FIG}
\noindent\sample\sample\sample\sample
```

We can specify the dimensions of the picture ourselves, so that L<sup>A</sup>T<sub>E</sub>X will use these parameters in its typesetting calculations, and will not try to use the intrinsic information associated with the source. If no offsets or position parameters are given, the content is centered (first picture). On the second picture we shift the content 2mm to the right and 14mm down. There the “dr” argument produces a dashed frame and places the picture to the right.

A `\picskip{nlines}` command instructs L<sup>A</sup>T<sub>E</sub>X to continue to typeset the paragraph for *nlines* lines at the given indentation (as though the picture extended downward for that many lines). A zero value for *nlines* means that the following lines no longer need to be indented and that a new paragraph must start. The

*Controlling the hole*

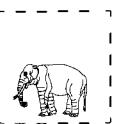
horizontal space between the paragraph text and the picture can be controlled through the `\pichskip` command.



Some text for our page that is reused over and over again. Some text for our page that is reused over and over again.

Here we prove that the “picture” can span more than a single paragraph.

Some text for our page that is reused over and over again. Some text for our page that is reused over and over again.



Without the explicit request in the source this paragraph would have only one shortened line, like the one surrounding the previous “picture”.

```
\usepackage{picins,graphicx}
\newcommand\FIG{\includegraphics
               [width=10mm]{elephant}}
% \sample as previously defined
\parpic(15mm,15mm)[f]{\FIG}\noindent
\sample\sample\par
Here we prove that the “picture” can
span more than a single paragraph.
\parpic(15mm,15mm)(2mm,14mm)[dr]{\FIG}%
\noindent\sample\sample\par
\pichskip{2}
Without the explicit request in the source
this paragraph would have only one
shortened line, like the one surrounding
the previous “picture”.
```

6-4-4

Perhaps the results produced by the offset in the previous example were somewhat surprising. For this reason the next example studies its effects in some detail. If we specify an offset of `0mm,0mm` the “picture” is placed with its reference point at the top-left corner of the area reserved for the picture. As most `LATEX` constructs produce a box with the reference point at the left of the bottom baseline, the “picture” is effectively placed outside the intended area—that is, in a completely different place than it would be without any offset at all.

Some text for  
our page that is  
reused over and  
over again. Some text for our  
page that is reused over and  
over again.

Some text for  
our page that is  
reused over and  
over again. Some text for our  
page that is reused over and  
over again.

Some text for  
our page that is  
reused over and  
over again. Some text for our  
page that is reused over and  
over again.

Some text for  
our page that is  
reused over and  
over again. Some text for our  
page that is reused over and  
over again.

```
\usepackage{picins}
% \sample as previously defined
\parpic(15mm,10mm)(0mm,0mm)
[dr]{\fbox{Box}}%
\sample\sample\par
\parpic(15mm,10mm)(2mm,5mm)
[dr]{\fbox{Box}}%
\sample\sample\par
\parpic(15mm,10mm)(4mm,10mm)
[dr]{\fbox{Box}}%
\sample\sample\par
\parpic(15mm,10mm)(6mm,15mm)
[dr]{\fbox{Box}}%
\sample\sample\par
```

6-4-5

You can use the `\parpic` inside list environments at any depth. This is in contrast to other packages in this area, which often restrict the placement of pictures within lists. The following example features an `itemize` list with embedded `\parpic` commands. It also shows how line thickness (`\linethickness`), length

of the dashes (`\dashlength`), and depth of the shade (`\shadowthickness`) and the 3-D effect (`\boxlength`) can all be controlled separately.

Some text for our page that is reused over and over again.

- Some text for our page that is reused over and over again. Some text for our page that is reused over and over again.
- Some text for our page that is reused over and over again.

6-4-6

again. Some text for our page that is reused over and over again.

-  Some text for our page that is reused over and over again. Some text for our page that is reused over and over again.

Some text for our page that is reused over and over again.

```
\usepackage{picins}
% \sample as previously defined
\sample
\begin{itemize} \item
\dashlength{2mm}
\linethickness{1mm}
\parpic(15mm,10mm)[dr]{BOX}
\sample\sample
\item
\shadowthickness{3mm}
\linethickness{.4pt}
\parpic(15mm,10mm)[sr]{BOX}
\sample\sample
\item
\boxlength{2mm}
\parpic(15mm,10mm)[x]{BOX}
\sample\sample
\end{itemize} \sample
```

One can generate numbered captions for the pictures that will appear in *L<sup>A</sup>T<sub>E</sub>X*'s "List of Figures". As the pictures do not float, one has to be careful when mixing them with ordinary floats to avoid out-of-sequence numbering. To specify a caption text you use the command `\piccaption`, which takes the same arguments as the standard `\caption` command but only stores them for use with the next `\parpic`.

For our first example we typeset the contents of a picture inside a framed shadow box, with the caption appearing outside the frame and below the picture. This corresponds to the default positioning for caption material. There is a space of 6mm between picture and text as specified with the `\pichskip` command.

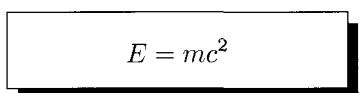


Figure 1: Einstein's formula.

6-4-7

text for our page that is reused over and over again.

```
\usepackage{picins}
\newcommand\FOR{\((\displaystyle E=mc^2)\)}
% \sample as before
\pichskip{6mm}
\piccaption{Einstein's formula.}
\parpic(45mm,10mm)[s]{\FOR}
\sample\sample
```

The default caption placement can be explicitly requested with the declaration `\piccaptionoutside`. The package offers three other placement options that can be selected `\piccaptioninside`, `\piccaptionside`, and `\piccaptiontopside`. Their effects are shown in the next example. Even though `picins` uses its own command to specify the caption text, it is possible to influence the caption formatting

by loading a package such as `caption`. We prove this by setting the caption label in bold sans serif font.

$$E = mc^2$$

**Figure 1:** Einstein's formula.

that is reused over and over again. Some text for our page that is reused over and over again.

$$E = mc^2$$

**Figure 2:** Einstein's formula.

Some text for our page that is reused over and over again.

**Figure 3:** Einstein's formula.

Some text for our page that is reused over and over again. Some text for our page that is reused over and over again.

$$E = mc^2$$

Some text for our page that is reused over and over again. Some text for our page that is reused over and over again.

```
\usepackage{picins}
\usepackage[labelfont={sf,bf}]{caption}
% \sample and \FOR as before
\piccaptioninside
\piccaption{Einstein's formula.}
\parpic(50mm,10mm)[s]{\FOR}
\sample\sample\sample

\piccaptionside
\piccaption{Einstein's formula.}
\parpic(30mm,10mm)[s]{\FOR}
\sample

\piccaptiontopside
\piccaption{Einstein's formula.}
\parpic(30mm,10mm)[sr]{\FOR}
\sample\sample
```

[ 6-4-8 ]

## 6.5 Controlling the float caption

When you want to explain what is shown in your floating environment (`figure` or `table` in standard L<sup>A</sup>T<sub>E</sub>X), you normally use a `\caption` command. After introducing the basic syntax and explaining the (low-level) interfaces available with standard L<sup>A</sup>T<sub>E</sub>X, this section describes the powerful `caption` package, which offers a large number of customization possibilities for adjusting the caption layout to your needs. As shown in the examples it can be combined with all other packages described in this chapter.

We then examine the `subfig` and `subfloat` packages, which introduce substructures for float objects. The section concludes with a discussion of the `sidecap` package (placing captions beside the float body) and the `fltpage` package (for generating full-page floats whose captions are placed on the opposite page).

`\caption[short-text]{text}`

This standard L<sup>A</sup>T<sub>E</sub>X command is only defined inside a float environment. It increments the counter associated with the float in question. If present, the optional argument *short-text* goes into the list of figures or tables. If only the mandatory argument *text* is specified, then it is used in those lists. If the caption is longer than one line, you are strongly advised to use the optional argument to provide

a short and informative description of your float. Otherwise, the list of figures and tables may become unreadable and it may be difficult to locate the necessary information. In fact, L<sup>A</sup>T<sub>E</sub>X allows multi-paragraph captions only if the *short-text* argument is present. Otherwise, you will get a “Runaway argument?” error.

The following example shows how standard L<sup>A</sup>T<sub>E</sub>X typesets captions. Compare this layout to the customization provided by the various packages discussed in the next sections. Note how the optional argument of the second \caption command defines what text appears for that figure in the “List of Figures”.

## List of Figures

1	Short caption text . . . . .	6	\section{Caption}
2	Short entry in lof . . . . .	6	

## 1 Caption

Figures 1 and 2 have captions.

A small Figure

Figure 1: Short caption text

A small Figure

Figure 2: Long caption text with some extra explanation that this figure is important even though it is small.

6-5-1

```
\listoffigures
\begin{figure}[ht]
\centerline{\fbox{\small A small Figure}}
\caption{Short caption text}\label{Fig1}
\end{figure}

\begin{figure}[ht]
\centerline{\fbox{\small A small Figure}}
\caption[Short entry in lof]{Long caption text with some extra
explanation that this figure is important
even though it is small.}\label{Fig2}
\end{figure}
```

Internally, \caption invokes the command \@makecaption{*label*}{*text*}. The *label* argument is the sequence number of the caption and some text like “Figure”; it is generated internally depending on the type of float. The *text* argument is passed on from the mandatory \caption argument; it is the text to be typeset. The default definition for the part responsible for the typesetting of a caption looks something like this:

```
\newcommand{\@makecaption}[2]{% #1 is e.g. Figure 1, #2 is caption text
  \vspace{\abovecaptionskip}%
  \sbox{\tempboxa{#1: #2}}%
  \ifthenelse{\lengthtest{\wd\tempboxa > \ linewidth}}{ test size
    {\noindent #1: #2\par}%
    \centering
    \makebox[\ linewidth]{c}{\usebox{\tempboxa}\par}%
  }{ several lines
    \centering
    \makebox[\ linewidth]{c}{\usebox{\tempboxa}\par}%
  }%
  \vspace{\belowcaptionskip}%
}
```

After an initial vertical space of size `\abovecaptionskip` (default often 10pt), the material is typeset in a temporary box `\@tempboxa`, and its width is compared to the line width. If the material fits on one line, the text is centered; if the material does not fit on a single line, it will be typeset as a paragraph with a width equal to the line width. Thereafter, a final vertical space of `\belowcaptionskip` (default typically 0pt) is added, finishing the typesetting. The actual implementation that you find in the standard classes uses lower-level commands to speed up the processing so it looks somewhat different.

You can, of course, define other ways of formatting your captions. You can even supply different commands for making captions for each of the different types of floats. For example, the command `\@makefigcaption` can be used instead of `\@makecaption` to format the captions for a `figure` environment.

```
\newcommand{\@makefigcaption}[2]{...}
\renewenvironment{figure}
  {\let\@makecaption\@makefigcaption \@float{figure}}
  {\end@float}
```

This approach requires fairly low-level programming and is not very flexible, so it is normally better to use a package like `caption` (described below) to do this work for you.

Rather than force you to write your own code for customizing captions, we invite you to read the following pages, which describe a few packages that offer various styles to typeset captions.

### 6.5.1 `caption`—Customizing your captions

Axel Sommerfeldt developed the `caption` package<sup>1</sup> to customize the captions in floating environments. It not only supports L<sup>A</sup>T<sub>E</sub>X's standard `figure` and `table` environments, but also interfaces correctly with the `\rotcaption` command and the `sidewaysfigure` and `sidewaystable` environments of the `rotating` package. It works equally well with most of the other packages described in this chapter (see the original documentation for a complete compatibility matrix).

Like the `geometry` package, the `caption` package uses the extended option concept (based on the `keyval` package), in which options can take values separated from the option name by an equals sign. In most cases there exists a default value for an option; thus, you can specify the option without a value to produce this default behavior.

The customization possibilities of the `caption` package cover (nearly) all aspects of formatting and placing captions, and we will introduce them below. For those users who need even more customization, the package offers an interface to add additional option values (representing special formatings). One can even

<sup>1</sup>The `caption` package is, in fact, a completely rewritten version of Axel's `caption2` package and makes the latter obsolete. Axel advises all users of `caption2` to upgrade to `caption` as soon as possible and, if needed, to modify their L<sup>A</sup>T<sub>E</sub>X sources accordingly.

add additional options, a functionality used, for example, by the `subfig` package described in Section 6.5.2.

The first set of options we examine here are those that influence the overall shape of the caption:

*Customizing the general shape*

`singlelinecheck` If the whole caption (including the label) fits on a single line, center<sup>1</sup> it (keyword `true`). With the keyword `false`, such captions are formatted identically to multiple-line captions.

`format` This option defines the overall shape of the caption (except when overwritten by the previous option). With the keyword `default`, you will get a typical “standard L<sup>A</sup>T<sub>E</sub>X” format that is, the label and the caption text are set as a single block. Absent any further customization by other options, the label and the text are separated by a colon and space, and the caption is set justified to full width.

As an alternative, the keyword `hang` specifies that the caption should be set with the label (and separation) to the left of the caption text. In other words, continuation lines are indented by the width of the label.

`margin, width` By default, the caption occupies the whole width of the column (or page). By specifying either a specific `width` or a `margin`, you can reduce the measure used for the caption. In either case the caption is centered in the remaining space. Thus, with the current implementation, it is not possible to specify different values for left and right (or inner and outer) margins.

`indentation` If set to a given dimension, this option specifies an additional indentation for continuation lines (e.g., on top of any indentation already produced by the `hang` keyword).



Figure 1: Short caption

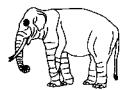


Figure 2: A caption that runs over  
more than one line

```
\usepackage{float,graphicx}
\usepackage[format=hang,margin=10pt]{caption}
\floastyle{boxed} \restylefloat{figure}
\begin{figure}[ht] \centering
\includegraphics[width=8mm]{elephant}
\includegraphics[width=10mm]{elephant}
\caption{Short caption}
\end{figure}
\begin{figure}[ht] \centering
\includegraphics[width=15mm]{elephant}
\caption{A caption that runs over more than one line}
\end{figure}
```

If you look at the previous example, you will notice that with this particular layout the space between box and caption appears very tight. Options for adjusting<sup>2</sup> such spaces are discussed on page 312. First, however, we look at options for

*Customizing the fonts*

<sup>1</sup>Or do something else with it.

<sup>2</sup>However, in some float styles, such as “boxed”, they are hard-wired and cannot be changed.

adjusting the fonts used within the caption, which are always working.

**font** This option defines the font characteristics for the whole caption (label and text) unless overwritten. This option can take a comma-separated list of keyword values to specify the font family (`rm`, `sf`, or `tt`), font series (`md` or `bf`), font shape (`up`, `it`, `s1`, or `sc`), or font size (`scriptsize`, `footnotesize`, `small`, `normalsize`, `large`, or `Large`). If more than one keyword is used, then the list must be surrounded by braces to hide the inner comma from being misinterpreted as separating one option from the next (see the example below).

Keywords for the same font attribute (e.g., the font shape) overwrite each other, but those for different attributes have the expected combined effect.

To set the font attributes to their default settings use the keyword `default`.

**labelfont** While the option `font` defines the overall font characteristics, this option specifies the (additional) attribute values to use for the caption label.

**textfont** This option is like `labelfont` but is used for the caption text. In the next example we use it to reset the font series from boldface to medium.



**Figure 1:** Short caption

A B C D E F G H I J K L M

**Table 1:** A caption that runs over more than one line

```
\usepackage{float,graphicx}
\usepackage[font={sf,bf},textfont=md]{caption}
\floatstyle{boxed} \restylefloat{table}
\begin{figure}[ht] \centering
\includegraphics[width=10mm]{Escher}
\caption{Short caption}
\end{figure}
\begin{table}[ht] \centering
\begin{array}{ccccccccccccc}
A & B & C & D & E & F & G & H & I & J & K & L & M
\end{array}
\caption{A caption that runs over more than one line}
\end{table}
```

6-5-3

*Customizing the label further* Another frequent requirement is the customization of the layout for the caption label, such as by replacing the default colon after the label by something else, or omitting it altogether. Also, the separation between label and text may require adjustments. Both can be achieved with the following options and their keywords.

**labelformat** With this option a format for the label can be selected. Out of the box the following keywords can be used: `simple` (label string, e.g., “Figure” and the number following each other and separated by a nonbreakable space), `parens` (number in parentheses), and `empty` (omit the label including the number altogether). The results of these keywords are shown in several examples in this chapter. Additional keywords for alternative formattings can be defined using the `\DeclareCaptionLabelFormat` declaration, as explained on page 313.

**labelsep** This option specifies the separation between the label and the text. Available keywords are `colon`, `period`, `space`, and `newline`, which have the expected meanings. New keywords producing other kinds of separations can be defined using the declaration `\DeclareCaptionLabelSeparator`; see the package documentation for more details.

**Figure 1****Figure 2.**

A small elephant

6-5-4

```
\usepackage{float,graphicx}
\floatstyle{boxed} \restylefloat{figure}
\usepackage[caption]
\DeclareCaptionLabelSeparator{period-newline}{.\newline}
\captionsetup{aboveskip=3pt,singlelinecheck=false,
             labelsep=period-newline,labelfont={small,bf}}
\begin{figure}[ht] \centering
\includegraphics[width=10mm]{Escher} \caption{}
\end{figure}
\begin{figure}[ht] \centering
\includegraphics[width=10mm]{elephant}
\caption{A small elephant}
\end{figure}
```

The actual formatting of the caption text within the general shape, such as the justification, can be customized using the following two options:

*Paragraph-related customizations*

**justification** This option specifies how the paragraph should be justified. The default is full justification (keyword `justified`). Using the keyword `centering` results in all lines being centered. The `raggedleft` and `raggedright` keywords produce unjustified settings with ragged margins at the indicated side.

If the `ragged2e` package is additionally loaded, you can use the keywords `Centering`, `RaggedLeft`, and `RaggedRight`, thereby employing the commands from that package that are described in Section 3.1.12.

Two other special justifications are available: `centerfirst` centers the first line and fully justifies the rest (with `\parfillskip` set to zero), whereas `centerlast` works the opposite way, centering the last line. Both shapes are sometimes requested for captions, but in most circumstances they produce questionable results.

Further specialized justification set-ups can be defined using the declaration `\DeclareCaptionJustification` as described in the documentation.

**parskip** This option controls the separation between paragraphs in multi-paragraph captions. It expects a dimension as its value. Recall that captions with several paragraphs are possible only if the optional `caption` argument is present!

# Bild

**Figure (1)** *A caption that runs over more than one line to show the effect of the centerfirst keyword.*

6-5-5

```
\usepackage[textfont={rm,it},labelfont={sf},
           labelformat=parens,labelsep=quad,
           justification=centerfirst,parskip=3pt]{caption}
\begin{figure}[ht] \centering
{\fontfamily{put}\fontsize{60}{60}\bfseries Bild}
\caption{A short caption text}
{A caption that runs over more than one line
  to show the effect of the centerfirst keyword.}
\end{figure}
```

*Customizing the spacing around the caption*

The final set of options deal with the position of the caption with respect to the float body. Note that none of these settings actually moves the caption in the particular place (you have to do that manually, or use a float style from the float package to do it for you). They only affect the space being inserted.

`aboveskip` Space between the caption and float body—for example, “above” the caption if caption is placed at the bottom. It typically defaults to 10pt.

`belowskip` Space on the opposite side of the caption—that is, away from the float body. It is 0pt in most standard classes.

`position` Specifies that the caption is placed above the float body (keyword `top`) or below the float body (keyword `bottom`). It does *not* place the caption there. That is still your task (or that of a package such as `float`).

*Be careful with the meanings of the options*

Note that the names `aboveskip` and `belowskip` give the wrong implications: they do *not* describe physical places, but rather are swapped if the caption is marked as being placed on the top. This is quite different from the parameters `\abovecaptionskip` and `\belowcaptionskip` in L<sup>A</sup>T<sub>E</sub>X’s default implementation of the `\caption` command (see page 307) which *do* describe their physical place in relation to the caption! For some float package styles setting these options may have no effect.

An option list as specified in the previous example may not be to everyone’s liking. In addition, it only allows us to customize the captions of all floats in the document regardless of their type. Sometimes, however, the captions for tables may need a different treatment than those for figures, for instance. In such a case the `\captionsetup` declaration will help.

`\captionsetup[type]{option-value-list}`

The `\captionsetup` declaration allows you to specify an *option-value-list* like the one possible when loading the package itself. The difference is that, if used with the optional *type* argument, this declaration specifies caption formatting for only this particular float type (e.g., `figure`) or any float type that has been set up with a `\newfloat` declaration from the `float` package.

`\DeclareCaptionStyle{name} [short-style] [long-style]`

Further assistance is available in the form of the `\DeclareCaptionStyle` declaration. It associates an option/value list with a name that can later be referred to as the value of a *style* option. The mandatory *long-style* argument is a list of option/value pairs that describe the formatting of a caption if the style *name* is selected. The optional *short-style* argument lists option/value pairs that are *also* executed whenever the caption is determined to be “short” (i.e., if it would fit on a single line).

It is possible to combine the *style* option with other options inside the argument of `\captionsetup`, as shown in the next example. There we select the

style default (predefined) for all floats except `figures` but overwrite its setting for `labelfont`. Note that the example is intended to show possibilities of the package—not good taste.



**Figure 1:** A long caption that runs over more than one line to show the effect of the style keyword.

A	B	C	D	E	F	G	H	I	J
---	---	---	---	---	---	---	---	---	---

**Table 1:** A long caption that runs over more than one line to show the effect of the style keyword.

6-5-6

```
\usepackage{caption,graphicx}
\DeclareCaptionStyle{italic}
  {labelfont={sf,bf},textfont={rm,it},indentation=18pt,
   labelsep=period,justification=raggedright}
\captionsetup[figure]{style=italic}
\captionsetup[style=default,labelfont={sf,bf}]
\begin{figure}[ht]
  \centering \includegraphics{cat}
  \caption{A long caption that runs over more than one line to show the effect of the style keyword.}
\end{figure}
\begin{table}[ht]
  \centering \fbox{A B C D E F G H I J}
  \caption{A long caption that runs over more than one line to show the effect of the style keyword.}
\end{table}
```

```
\DeclareCaptionLabelFormat{name}{code}
```

This declaration defines or redefines a `labelformat` keyword *name* to generate *code* to format the label, where *code* takes two arguments: #1 (a string like “Figure”) and #2 (the float number). Thus, to produce parentheses around the whole label, you can define your own `parens` keyword as follows:

```
\DeclareCaptionLabelFormat{parens}{(#1\nobreakspace#2)}
```

While this approach would work well in all examples seen so far, the above definition nevertheless contains a potential pitfall: if #1 is empty for some reason (e.g., if you changed `\figurename` to produce nothing), the above definition would put a space in front of the number. To account for situations like this the `caption` package offers the `\bothIfFirst` command.

```
\bothIfFirst{first}{second} \bothIfSecond{first}{second}
```

The `\bothIfFirst` command tests whether *first* is non-empty and, if so, typesets both *first* and *second*. Otherwise, it typesets nothing. With its help the above declaration can be improved as follows:

```
\DeclareCaptionLabelFormat{parens}
{(\bothIfFirst{#1}{\nobreakspace}#2)}
```

As a second example, suppose you want your caption labels to look like this: "(4) Figure". You could set up a new format, named `parensfirst`, and later assign it to the `labelformat`:

```
\DeclareCaptionLabelFormat{parensfirst}
  {(#2)\bothIfSecond{\nobreakspace}{#1}}
\captionsetup{labelformat=parensfirst}
```

In a similar fashion you can add new keywords for use with the `labelsep` using the `\DeclareCaptionLabelSeparator` declaration.

```
\DeclareCaptionLabelSeparator{name}{code}
```

After a `\DeclareCaptionLabelSeparator` the keyword `name` refers to `code` and can be used as the value to the `labelsep` option. For example, if you want to have a separation of one quad between the label and the text that should be allowed to stretch slightly, you can define

```
\DeclareCaptionLabelSeparator{widespace}{\hspace{1em plus .3em}}
```

and then use it as `labelsep=widespace` in the argument of `\captionsetup` or `\DeclareCaptionStyle`.

*Providing new  
caption shapes and  
justifications*

In addition to customizing the label format, you can define your own general caption shapes using `\DeclareCaptionFormat`, or specialized justification settings using `\DeclareCaptionJustification`. These are more specialized extensions and their internal coding is a bit more difficult, so we will not show an example here. If necessary, consult the package documentation.

*External  
configuration files*

Such declarations can be made in the preamble of your documents. Alternatively, if you are using the same settings over and over again, you can place them in a configuration file (e.g., `mycaption.cfg`) and then load this configuration as follows:

```
\usepackage[config=mycaption]{caption}
```

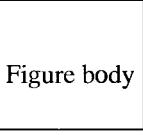
*Continuing captions  
across floats*

While it is possible to combine the `config` option with other options, it is probably clearer to specify additional modifications through a `\captionsetup` declaration in the preamble.

Sometimes figures or tables are so large that they will not fit on a single page. For such tables, the `longtable` or `supertabular` package may provide a solution. For multipage figures, however, no packages for automated splitting are available.

In the past a general solution to this problem was provided through the `captcont` package written by Steven Cochran, which supports the retention of a caption number across several float environments. Nowadays this functionality is readily available with the `caption` package. It provides the command `\ContinuedFloat`, to be used before issuing the `\caption` command if the current caption number should be retained.

If you prefer that the continued caption not to appear in the “List of...” list, use `\caption` with an empty optional argument (see Example 6-5-13 on page 321), or `\caption*`, which suppresses LOF entry and caption number.

<b>List of Figures</b> <table border="0"> <tr><td>1</td><td>Huge . . .</td><td>6</td></tr> <tr><td>1</td><td>Huge (cont.)</td><td>7</td></tr> </table> <p>A figure placed at the page</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;">Figure body</div> <p>Figure 1: Huge</p>	1	Huge . . .	6	1	Huge (cont.)	7	 <p>Figure 1: Huge (cont.)</p> <p>bottom and continued at the top of the next page.</p>	<pre>\usepackage{caption} \listoffigures \medskip \begin{figure}(!b] \centering \fbox{Figure body} \caption{Huge} \end{figure} A figure placed at the page bottom and continued at the top of the next page. \begin{figure}(!t] \ContinuedFloat \centering \fbox{\rule[-.5cm]{0pt}{1.5cm}}% Figure body \caption{Huge (cont.)} \end{figure}</pre>
1	Huge . . .	6						
1	Huge (cont.)	7						
6-5-7	6	7						

The `caption` package collaborates smoothly with the other packages described in this chapter, as can be observed in the various examples. Note that in some cases this package has to be loaded *after* the packages whose captioning style one wants to modify.

### 6.5.2 subfig—Substructuring floats

The `subfig` package (by Steven Cochran) allows the manipulation and reference of small, “sub” figures and tables by simplifying the positioning, captioning, and labeling of such objects within a single float environment. If desired, sub-captions associated with these sub-floats can appear in the corresponding list of floats (e.g., the list of figures). In addition, a global caption can be present.

The package is based on the `caption` package, discussed in the previous section, and makes use of all its features for customizing the layout of captions.<sup>1</sup> The main user command to identify a sub-float object within a float is `\subfloat`.

`\subfloat[list-entry] [caption] {object}`

The mandatory *object* argument specifies the sub-float content, the optional *caption* argument denotes the caption text for this object, and, if necessary, the optional *list-entry* argument specifies an alternate form to be used in the list of figures (or tables). If no optional argument is provided, no caption (and no caption

<sup>1</sup>An earlier version of this package was known as `subfigure`. It had a number of customization possibilities in common with the `caption2` package by Axel Sommerfeldt, but differed in some important details. When `caption2` was upgraded, the author of this book persuaded Steven to base a new version of his code on the emerging `caption` package. The results are described in this section.

label) is produced. If you wish to get only an (alpha)numeric label, use an empty *caption* argument.

An empty *list-entry* signifies that for this instance the caption text should not be inserted in the “List of...”. This special feature is relevant only if the sub-float captions should be listed there in the first place: see page 320 for information on creating this set-up.

Our first example shows a figure that features two `\subfloat` components. To reference them, you must associate labels with each of these `\subfloat` commands (be careful to put the `\label` commands *inside* the braces enclosing the contents of the `\subfloat`). We also place a `\label` following the `\caption` command to identify the enclosing `figure` environment, so that outside the environment we can refer to each of the components separately.

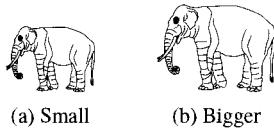


Figure 1: Two elephants

```
\usepackage{subfig} \usepackage{graphicx}
\begin{figure} \centering
\subfloat[Small]{\includegraphics[width=12mm]{elephant}\label{sf1}}
\qquad
\subfloat[Bigger]{\includegraphics[width=16mm]{elephant}\label{sf2}}
\caption{Two elephants}\label{elephants}
\end{figure}
```

Figure 1 contains sub-figure 1a, which is smaller than sub-figure 1b.

Figure~\ref{elephants} contains sub-figure~\ref{sf1}, which is smaller than sub-figure~\ref{sf2}.

6-5-8

Because the `subfig` package is based on `caption`, it is possible to influence the caption layouts for sub-floats using the options offered by the latter package. If it is not already loaded, `subfig` loads the `caption` package *without* any options. This means you have to either load `caption` first (as we did in the example below) or customize it after loading `subfig` by using a `\captionsetup` declaration.

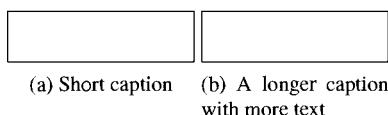


Figure 1: Default sub-figures

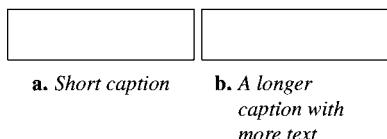
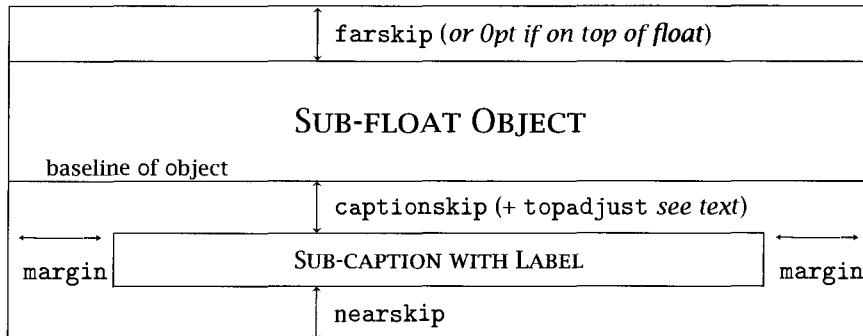


Figure 2: Customized sub-figures

```
\usepackage[font=sf]{caption}
\usepackage{subfig}
\newcommand\LCap{A longer caption with more text}
\newcommand\FIG{\fbox{\parbox{.4\textwidth}{\strut}}}
\begin{figure}[ht] \centering
\subfloat[Short caption]{\FIG} \subfloat[\LCap]{\FIG}
\caption{Default sub-figures}
\end{figure}
\captionsetup[subfloat]{format=hang, textfont=it,
labelfont={rm,bf}, labelformat=simple, labelsep=period,
margin=5pt, justification=raggedright}
\begin{figure}[ht] \centering
\subfloat[Short caption]{\FIG} \subfloat[\LCap]{\FIG}
\caption{Customized sub-figures}
\end{figure}
```

6-5-9

Figure 6.1: Spacing layout of the `subfig` package

As you can see, options for customizing the caption layouts can be set on various levels. Some default settings are already in place when the `subfig` package is loaded. Most noticeably, a setting of `font=footnotesize` for all sub-float captions accounts for the fact that our setting of `sf` when loading the `caption` package has no effect on the sub-captions. Another default that can be deduced is the use of parens with the `labelformat` option. But most other changes to the main caption layout are inherited by the sub-floats.

To overwrite such defaults, you can use any of the `caption` options when loading the `subfig` package, or you can specify them with a `\captionsetup` declaration using the `type` “`subfloat`” (as shown in the example). This will change all subsequent sub-float captions uniformly until they are overwritten by a further declaration.

Finally, if you want to customize sub-float captions just for a particular `(type)` of float (e.g., for all `figures`) you can do so by using `sub<type>` instead of `subfloat` in the `\captionsetup` declaration.

The `subfig` package offers a number of customization possibilities through a set of additional options (not available with the `caption` package) that expect a dimension as their value. They define the space produced around a sub-float. Assuming the default caption position below the object (i.e., `position=bottom`), we get a layout like that shown in Figure 6.1.

`farskip` Specifies the space left on the side of the sub-float that is opposite the main float caption (e.g., on top if the main caption is at the bottom of the float). This space is ignored if it is the first object in the float body. The default value if not modified is 10pt.

`nearskip` Specifies the space left on the side of the sub-float nearer the main caption to separate the sub-float object and its caption from surrounding material. It defaults to 0pt.

`captionskip` Specifies the vertical space that separates the sub-float object and its caption (default 4pt). If there is no caption, this space is not added.

*The default setting  
of the subfig  
package*

*Customizing all  
sub-captions*

*Customizing  
sub-captions by type*

*Spacing around  
sub-floats*

`topadjust` Not applicable with `position=bottom` on the sub-float level. If the sub-caption is placed above the sub-float object (i.e., Figure 6.1 flipped upside down using `position=top`) this space is added to the `captionskip` used to separate caption and sub-float body.

The caption is set to the width of the sub-float object reduced on both sides by the value specified with the `margin` option already provided by `caption` package.

If the caption is placed above the sub-float object (i.e., using `position=top` for the sub-float), then `captionskip` is increased by `topadjust` to allow for adjusting the separation between the caption and the object in this case. Also, note that the position of `farskip` and `nearskip` depends on the placement of the main caption. When it comes first (i.e., `position=top` at the float-level) `farskip` and `nearskip` swap places.

*Labeling the sub-captions* Internally, `\subfloat` uses a counter to keep track of the sub-floats within the current float and to produce a label for the caption from it. The counter name is `sub<type>`, where `type` is the current float type (e.g., the counter used for labeling sub-figures is called `subfigure`). Its representation is defined by `\thesub<type>` and defaults to `\alph{sub<type>}`. These counters are incremented for each sub-float regardless of whether a caption was printed.

A somewhat more complex layout applying several of the above options has been used in the following example. It introduces three sub-tables, two on top of a third. Due to the option settings the table captions appear above the tables in small slanted type. Single-line captions are set flush left; multiple-line captions are set ragged right with hanging indentation. To show further customization possibilities, the `\thesubtable` command (which generates the “number” for a sub-float of type `table`) is redefined to produce two-level caption numbers on the sub-tables. Each of the `\subfloat` commands, as well as the enclosing `table` environment, is identified by a strategically positioned `\label` command. They allow us to address the components individually.

Table 1: Three sub-tables	
(1.1) First	(1.2) Second
Table 1	Table 2
(1.3) Third table with a much longer caption	

Table 3  
Table 1 contains sub-tables (1.1) to (1.3). But don't use now: 11.3 (see text).

```
\usepackage{subfig}
\captionsetup[table]{position=top, aboveskip=5pt}
\captionsetup[subtable]{singlelinecheck=false,
format=hang, font={sl, small},
justification=raggedright}
\renewcommand\thesubtable{\thetable.\arabic{subtable}}
\newcommand\tAB[2]{\fbox{\parbox{\textwidth}{Table #1}}}
\begin{table}
\caption{Three sub-tables}\label{tbl}
\subfloat[First]{\tAB{1}{.4}\label{tbl1}}\hfill
\subfloat[Second]{\tAB{2}{.4}\label{tbl2}}\\
\subfloat[Third table with a much longer caption]{\tAB{3}{.8}\label{tbl3}}
\end{table}
Table \ref{tbl} contains sub-tables \subref{tbl1} to \subref{tbl3}. But don't use now: \ref{tbl3} (see text).
6-5-10
```

The references to the individual sub-tables in the previous example were created using the `\subref` command, which returns the reference formatted according to the `listofformat` (see page 320). This avoids any problem created by our redefinition of the `\thetable`, which would cause the `\ref` command to produce numbers like "11.3", because it combines the table number "1" with the sub-table number (e.g., "1.3").

The starred version of this command, `\subref*`, returns only the plain sub-float number (e.g., the value of `\thesubtable`), if needed to construct more complex references, such as "Figure 1(a-c)".

Sometimes one wants to label sub-floats but omit textual captions. This is, for example, common practice when showing a set of pictures or photographs: the main caption explains the significance of individual sub-floats. It can easily be achieved by using an empty optional argument on the `\subfloat` command, which results in a labeled sub-float. The next example shows this type of layout.

*Captionless  
sub-floats*

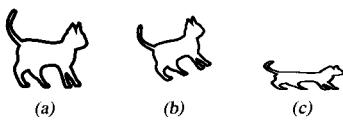


Figure 1: A group of cats: (a) the first cat, (b) a climbing one, and (c) one that is stretched.

6.5.11

```
\usepackage{graphicx}
\usepackage[font=\scriptsize, sl, captionskip=3pt]{subfig}
\newcommand\FIG[1]{\includegraphics[#1]{cat}}
\begin{figure} \centering
\subfloat[\FIG{width=3pc}\label{a}]{} \quad
\subfloat[\FIG{angle=20,width=3pc}\label{b}]{} \quad
\subfloat[\FIG{height=1pc,width=3pc}\label{c}]{} \\
\caption[A group of cats]{A group of cats: \subref{a} the first cat, \subref{b} a climbing one, and \subref{c} one that is stretched.}
\end{figure}
```

It is also possible to fine-tune individual floats, if their sub-floats have unusual forms or excess white space. In Example 6-5-8 on page 316, we could, for example, *Manual fine-tuning* move the main caption closer to the sub-captions by adding the line

```
\captionsetup[subfloat]{nearskip=-3pt}
```

at the top of the float body. This command would apply to the current float only and cancel part of the `aboveskip` added above the main caption.

```
\usepackage[subfig]{graphicx}
\begin{figure} \centering
\captionsetup[subfloat]{nearskip=-3pt}
\subfloat[Small] {\includegraphics[width=12mm]{elephant}\label{sf1}} \\
\qquad \subfloat[Bigger] {\includegraphics[width=16mm]{elephant}\label{sf2}} \\
\caption{Two elephants}\label{elephants}
\end{figure}
```

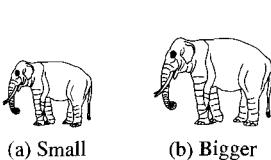


Figure 1: Two elephants

6.5.12

So far, we have discussed only sub-floats in `figure` or `table` environments. If you have added additional float types, you may want to be able to substructure them as well. This can be achieved with the `\newsubfloat` declaration.

`\newsubfloat [option-value-list] {float-type}`

A prerequisite for using `\newsubfloat` is that there must already exist the environments to produce the given `float-type`—for example, environments declared with `\newfloat` from the `float` package. In that case `\newsubfloat` will set up `\subfloat` to be usable within their float bodies (e.g., by declaring the counter `\sub<float-type>` to produce their labels). In the optional `option-value-list` argument, you can specify layout options that should apply only to this particular type of sub-float.

*Producing list of ... entries* The sub-float captions are automatically entered into the external file holding the data for the corresponding “List of ...” list. Such files have the extension `.lof` (a list of figures), `.lot` (list of tables), or the extension specified as the third argument to `\newfloat`.

The sub-float captions will not show up in these lists because only top-level float captions are typeset by default. To change this behavior, you have to set the counter’s `extdepth` to 2 (where `ext` is the extension of the corresponding “List of ...” file). For example, to make sub-figures captions appear you would use `\setcounter{lofdepth}{2}`, and for sub-tables you would change the value of `lotdepth`.

As explained in Section 2.3.2 the layout of such entries can be customized by redefining `\l@subfigure`, `\l@subtable`, and similar commands; the command name consists of float type prefixed by `l@sub`. However, `subfig` already offers three options that influence the entries in this list and they probably provide enough flexibility in most circumstances.

`listofindent` The indentation for the sub-float caption inside the contents list. Its default value is `3.8em`.

`listofnumwidth` The width reserved for the label in the contents list. Its default is `2.5em`.

`listofformat` The format used for the label of the sub-float entry when displayed in the contents list. Possible keywords are `empty`, `simple`, `parens`, `subsimple`, and `subparens` (default). Additional formatings can be declared using the `\DeclareCaptionListOfFormat` command; for details, see the package documentation.

The typeset result is also used by the `\subref` command, so changing the value of this option will affect references created by this command.

The next example shows how the sub-floats appear in the contents listings. We set `lofdepth` to make them appear and extend `listofindent` to `5em` so that they are slightly indented. We also use a continuation float to prove that sub-float numbering continues as well. To suppress the “List of ...” entry for the continu-

ation float we use an empty optional argument on the `\caption` command—the special feature provided by the `caption` package for such situations. Alternatively, we could have used `\caption*` to suppress both the caption number and the entry in the list of figures.

## List of Figures

1	Three figures . . . . .	1
1(a)	First . . . . .	1
1(b)	Second . . . . .	1
1(c)	Third . . . . .	2

Figure I

Figure II

(a) First                    (b) Second

Figure 1: Three figures

```
\usepackage[nearskip=-3pt,captionskip=5pt]{subfig}
\captionsetup[subfloat]{listofindent=5em,
listofformat=parens}
\setcounter{lofdepth}{2}
\listoffigures \medskip
\begin{figure}[!ht]\centering
\subfloat[First]{\fbox{Figure I}} \quad \qquad
\subfloat[Second]{\fbox{Figure II}}
\caption{Three figures}
\end{figure}
\pagebreak % <-- for illustration
\begin{figure}[!ht] \centering \ContinuedFloat
\subfloat[Third]{\fbox{Figure III}}
\caption[]{Three figures (cont.)}
\end{figure}
```

6-5-13

Like the `caption` package, `subfig` supports the use of external configuration files that contain your favorite settings using the option `config`. For example,

*External configuration files*

```
\usepackage[config=xcaption]{subfig}
```

loads the file `xcaption.cfg`.

While it is possible to combine the `config` option with other options, a clearer approach is to specify additional modifications through a `\captionsetup` declaration in the preamble.

### 6.5.3 subfloat—Sub-numbering floats

The `subfloat` package, developed by Harald Harders, can generate sub-numbers for figures or tables (analogous to the `subequations` environment of the `amsmath` package). While the `subfig` package sub-numbers objects inside one float, the `subfloat` package allows sub-numbering of the main captions of separate floats.

Figures (tables) for which sub-numbers are to be generated should be included inside a `subfigures` (`subtables`) environment. Alternatively, they can be placed between the commands `\subfiguresbegin` and `\subfiguresend` (`\subtablesbegin` and `\subtablesend`). While the environments must obey the basic nesting rules with respect to other environments, the commands can be placed anywhere. This flexibility can be helpful in unusual circumstances—for example, when sub-figures and sub-tables are intermixed.

The example that follows shows three figures. The first two are inside a `subfigures` environment, so they use sub-numbering (“1a” and “1b”). Both these labels are correctly handled by L<sup>A</sup>T<sub>E</sub>X’s `\listoffigures` and `\ref` commands.

## List of Figures

- |    |                |   |
|----|----------------|---|
| 1a | First figure . | 1 |
| 1b | Second figure  | 1 |
| 2  | Third figure   | 2 |

**Figure I**

Figure 1a: First figure

**Figure II**

Figure 1b: Second figure

Figures 1a and 1b in this

example are sub-numbered, while Figure 2 is not.

**Figure III**

Figure 2: Third figure

```
\usepackage{subfloat}
\listoffigures \medskip
\begin{subfigures}
\begin{figure} [!ht]
\centering\fbox{Figure I}
\caption{First figure}\label{FI}
\end{figure}
\begin{figure} [!ht]
\centering\fbox{Figure II}
\caption{Second figure}\label{FII}
\end{figure}
\end{subfigures}
\end{subfigures}
\begin{subfigures}
\begin{figure} [!ht]
\centering\fbox{Figure III}
\caption{Third figure}\label{FIII}
\end{figure}
\end{subfigures}
```

6-5-14

As in the previous example, the default caption label combines an Arabic numeral for the main figure with a lowercase letter to differentiate between the individual sub-figures. This label can be customized by redefining the command `\thesubfloatfigure`. Within its definition the command `\themainfigure` can be used to produce the main figure number<sup>1</sup> and the counter `subfloatfigure` to refer to the number of the sub-figure. Thus, to number sub-figures as “2.1”, “2.2”, and so on, one can define

```
\renewcommand\thesubfloatfigure{\themainfigure.\arabic{subfloatfigure}}
```

The same possibilities can be realized for tables by using the macros `\thesubfloattable` and `\themaintable`, and the counter `subfloattable`.

To enable users to automatically refer to the total number of sub-figures with the same main figure number, the package offers the option `countmax`. When it is used, the floats within a `subfigures` (`subtables`) environment are counted and the number is made available in the counter `subfloatfiguremax` (`subfloattablemax`). One could, for example, define

```
\renewcommand\thesubfloatfigure{\themainfigure
(\arabic{subfloatfigure}/\arabic{subfloatfiguremax})}
```

to produce caption labels such as “2(1/3)”, “2(2/3)”, and “2(3/3)” when the second set of figures consists of three sub-figures. This counting is implemented as a two-

<sup>1</sup>For technical reasons the command `\thefigure` is not usable within sub-figures. The “alias” `\themainfigure` is provided for this purpose.

pass system that uses the `\label` and `\ref` mechanism internally—which means that it is expensive in terms of resources and time. For this reason the default is not to count.

#### 6.5.4 `sidecap`—Place captions sideways

In their `sidecap` package Hubert Gäßlein and Rolf Niepraschk introduce two new environments, `SCfigure` and `SCtable`. They are analogous to L<sup>A</sup>T<sub>E</sub>X's `figure` and `table`, but typeset their captions at the side of the float in a `minipage` of a customizable width.

The package supports a number of options to influence the caption placement and formatting.

`outercaption/innercaption` The caption is typeset on the outer (default) or inner side of the page, respectively, i.e., varying between verso and recto pages.

`leftcaption/rightcaption` The caption is always typeset on the left or right side of the page, respectively.

`wide` The caption or float may extend into the margin if necessary.

`margincaption` The caption is set in the margin, with the float body appearing above the text. If this option is selected, the positioning of the float body with respect to the galley margins can be defined by using `innerbody`, `outerbody`, `centerbody`, `leftbody`, or `rightbody`.

`raggedright/raggedleft/ragged` The caption text is not justified. With small measures, this option often leads to better results. With `ragged` the unjustified margin varies between verso and recto pages, so this is best used with `innercaption`, `outercaption`, or `margincaption`. Martin Schröder's `ragged2e` package is used, when available on the system.

If the `sidecap` package is combined with the `caption` package, you have the choice of specifying the justification with the above options or through the `justification` option of the `caption` package. Only `ragged` is unique, as `caption` offers no way to vary the justification between pages.

```
\begin{SCfigure}[rel-width] [float-spec] <L-R material> \end{SCfigure}
\begin{SCtable}[rel-width] [float-spec] <L-R material> \end{SCtable}
```

The environments `SCfigure` and `SCtable` (and their starred versions for spanning two columns) take two *optional* arguments. The `rel-width` argument defines the width of the caption relative to the width of the table or figure body (default 1.0). A large value (e.g., 20) reserves the maximal width available on the page. The second argument, `float-spec`, is L<sup>A</sup>T<sub>E</sub>X's standard float positional argument (e.g., `[htb]`). In contrast to standard L<sup>A</sup>T<sub>E</sub>X floats, the float body is assumed to be horizontal material (necessary to be able to measure it). If you require vertical material at this point, use a `minipage` environment inside the body.

The first example shows a table and a figure with their captions set beside them. For the table the defaults have been used, resulting in a caption that occupies the same amount of space as the table. The figure is set with the caption twice as wide as the figure body. With the defaults the caption would have been typeset on two lines even though ample space is available. Except for the justification, the actual caption layout has been customized using the `caption` package.

AAA	BBB
CCC	DDD
EEE	FFF

**Table 1.** A  
small table with  
a rather long  
caption text

Figure I

Figure 1. A small figure

Paragraph text showing how floats are horizontally aligned with respect to the galley.

```
\usepackage[ragged]{sidecap}
\usepackage[labelfont={sf,bf},textfont=it,
labelsep=period]{caption}
Paragraph text showing how floats are
horizontally aligned with respect to the galley.
\begin{SCtable} \caption{A small table with a
rather long caption text}
\begin{tabular}{|l|l|} AAA & BBB \\
CCC & DDD \\ EEE & FFF \end{tabular}
\end{SCtable}
\begin{SCfigure}[2] \caption{A small figure}
\framebox[.3\linewidth][c]{Figure I}
\end{SCfigure}
```

6-5-1

In addition to its options, the `sidecap` package offers some parameters to influence the formatting. The size of the separation between the body and the caption can be changed by redefining `\sidecaptionsep` (using `\renewcommand`). The default is to use the value of the parameter `\marginparsep`. Instead of repeatedly specifying an optional argument to the environments, you can set the (default) relation between the float body and the caption size by redefining `\sidecaptionrelwidth`. For tables, the caption is aligned at the top; for figures, it is aligned at the bottom. This default can be changed by using a declaration like `\sidecaptionvpos{table}{b}`, where the second argument should be any one of: `t`, `c`, or `b`.

Changing the  
default settings

The next example uses all three customization possibilities, and the floats are allowed to extend into the margin (option `wide`). In fact, because of the chosen value for `\sidecaptionrelwidth`, they are forced to use all space available.

AAA	BBB
CCC	DDD
EEE	FFF

Table 1: A small table with a  
rather long caption text

Text showing how the float is horizontally aligned with respect to the galley.

```
\usepackage[wide]{sidecap}
\renewcommand\sidecaptionsep{15pt}
\renewcommand\sidecaptionrelwidth{20}
\sidecaptionvpos{table}{c}
Text showing how the float is horizontally
aligned with respect to the galley.
\begin{SCtable} \caption{A small table with
a rather long caption text}
\begin{tabular}{|l|l|} AAA & BBB \\
CCC & DDD \\ EEE & FFF \end{tabular}
\end{SCtable}
```

6-5-1

The package tries hard to produce a reasonable alignment between the float body and the caption text. In most cases, such as when the body consists of a `tabular` environment, it will produce satisfactory results. However, if the body contains straight text, perhaps as part of a `minipage` environment, you may have to help the alignment along by specifying a `\strut`, as shown in the next example. The second `\strut` on the last line is actually not necessary for a top-aligned caption but would be needed if the caption is bottom-aligned.

The example demonstrates the `ragged` option showing that it results in a ragged left setting if the caption appears in the left margin.

Table 1:	A misaligned caption	Some text for our page that is reused over and over again.
	caption	Some text for our page that is reused over and over again.

Table 2:	An aligned caption	Some text for our page that is reused over and over again.
	caption	Some text for our page that is reused over and over again.

```
\usepackage[margincaption,ragged]{sidecap}
% \sample as defined earlier
\begin{SCTable} \caption{A misaligned caption}
\begin{minipage}{\linewidth}
\sample \sample
\end{minipage}\end{SCTable}
\begin{SCTable} \caption{An aligned caption}
\begin{minipage}{\linewidth}
\strut \sample \sample \unskip\strut
\end{minipage}\end{SCTable}
```

6-5-17

### 6.5.5 `fltpage`—Captions on a separate page

When dealing with large figures or tables, sometimes insufficient room is left on the page to typeset the caption. Sebastian Gross's `fltpage` package addresses this problem by defining the environments `FPfigure` and `FPtable`. They are similar to `figure` and `table`, respectively, but typeset the caption for a full-page figure or table on the opposite page in `twoside` mode, or on the preceding or following page in `oneside` mode.

The package behavior is controlled by a number of options that specify the placement of the caption in relation to the float body (options in parentheses are alias option names):

- `closeFloats` The full-page floats are placed on the *next* possible page. In `twoside` mode the caption is placed on the bottom of the opposite page; in `oneside` mode it is always placed on the page before the float body.
- `rightFloats (CaptionBefore)` The float body always appears on a recto page and the caption on the previous page.
- `leftFloats (CaptionAfterwards)` The float body always appears on a verso page and the caption on the following page.

The “isolated” caption that refers to a full-page float is separated from the remaining text on the page by a horizontal rule. This rule can be suppressed by specifying the `noSeparatorLine` option. Moreover, to make the connection

between the caption and the float, you can let the package add hints like “Table xx. (on the facing page)” by specifying the option `variorref`. In that case the `variorref` package is used to produce such texts in the document language.<sup>1</sup>

We next construct a simple example demonstrating the principles underlying the `fltpage` package. In the example we construct an artificial full-page table by putting a frame containing an invisible rule (of zero width) inside a box with dimensions that are a small fraction smaller than the page dimensions.<sup>2</sup> The figure caption is typeset at the bottom of the page opposite the float material. Because we load the `variorref` package and specify the `variorref` option, the text “(*on the next page*)” is inserted automatically by the `fltpage` package.

## List of Figures

- |   |                    |   |
|---|--------------------|---|
| 1 | A full-page figure | 6 |
|---|--------------------|---|

### 1 Full-page floats

Figure 1 is a full-page float whose caption and body are on separate pages.

A full-page figure

Figure 1 (*on the next page*):  
Caption for a full-page float for  
which there was no room on  
the same page

6

7

```
\usepackage[twoside, variorref,
            closeFloats]{fltpage}
\listoffigures
\section{Full-page floats}
Figure~\ref{FP1} is a
full-page float whose caption
and body are on separate pages.
\begin{FPfigure}
\setlength\fboxsep{0pt}
\framebox[.97\linewidth][c]
{\rule[-3cm]{0pt}{.97\textheight}%
 A full-page figure}
\caption[A full-page figure]
{Caption for a full-page float
 for which there was no room
 on the same page}\label{FP1}
\end{FPfigure}
```

: 6-5-1-

Caveats Unfortunately this package is no longer being developed. Thus, it is, for example, impossible to use it for float types other than `figure` and `table` (e.g., those that can be defined with the `float` package). Furthermore, problems may potentially arise if floats appear to close to each other in the source (the content of the second might overwrite the first). Nevertheless, if used with care, it provides a solution to the difficult problem of handling large floats that currently has no counterpart in any other package available.

<sup>1</sup>This feature may not work if the layout of the caption is customized by the `caption` package.

<sup>2</sup>This step is needed to avoid generating overfull boxes due to the width of the `\framebox` rules. The separation `\fboxsep` between the frame and the inner material is also set to zero points.

## C H A P T E R 7

# Fonts and Encodings

### 7.1 Introduction

Half of the job of (L<sup>A</sup>) $\text{\TeX}$  as a typesetting system is to process the source document and to calculate from it the characters' positions on the output page. But (L<sup>A</sup>) $\text{\TeX}$  has only a primitive knowledge about these characters, which it basically regards as black boxes having a width, height, and depth. For each font these dimensions are stored in a separate external file, the so-called  $\text{\TeX}$  font metric or `.tfm` file.

The character shapes that correspond to such a `.tfm` file come into play at a later stage, after (L<sup>A</sup>) $\text{\TeX}$  has produced its `.dvi` file. Character placement information in the `.dvi` file and information about character shapes present in the `.pk` file or in outline descriptions (e.g., PostScript) are combined by a driver program that produces the character image on the output medium. Usually one driver program is needed for every output medium—for screen representation, a low-resolution laser printer, or other device. With  $\text{\TeX}$  variants such as pdf $\text{\TeX}$  or V $\text{\TeX}$  that bypass the production of `.dvi` output and instead directly generate PDF or PostScript output, the situation is slightly different (but, as far as L<sup>A</sup> $\text{\TeX}$  is concerned, similar). In that case the character shapes are “added” when the underlying formatter produces the final output format. That is, the driver program is internal, but the basic concepts are identical.

#### 7.1.1 The history of L<sup>A</sup> $\text{\TeX}$ 's font selection scheme (NFSS)

When  $\text{\TeX}$  was developed in 1979, only a dozen fonts were set up for use with the program: the “Almost Computer Modern” fonts, developed by Donald Knuth along with  $\text{\TeX}$ . With only this restricted set of fonts being available, a straightforward

approach for accessing them was used: a few control sequences were defined that changed from one external font to another.

This situation had not greatly changed five years later, when L<sup>A</sup>T<sub>E</sub>X was first released. Only the names of the fonts supplied with (L)T<sub>E</sub>X had changed, from Almost Computer Modern to Computer Modern, which was merely a slightly improved version of the former. So it was quite natural that L<sup>A</sup>T<sub>E</sub>X's font selection scheme followed the plain T<sub>E</sub>X concept with the addition of size-changing commands that allowed typesetting in 10 predefined sizes.

As a result L<sup>A</sup>T<sub>E</sub>X's font selection was far from general. For instance, when defining a heading command to produce a bolder font (by using a \bf command in its definition), the use of, say, \sf (for a sans serif font) inside that same heading did not produce a bold sans serif font but rather a medium-weight sans serif font (the bold attribute was ignored). Similarly, when, say, \bf was used inside emphasized text, the result was not a bold italic font, as normally desired, but rather a plain Roman bold font.

This behavior was caused by the fact that all the font-changing commands, such as \bf, referred to a fixed external font. As a consequence, rather than requesting an attribute change of the current font, they replaced the current font with another. Of course, L<sup>A</sup>T<sub>E</sub>X enhanced the plain T<sub>E</sub>X mechanism to a certain extent by providing a set of size-changing commands. Nevertheless, the underlying concept of the original release had a major drawback: the correspondence tables were hard-wired into L<sup>A</sup>T<sub>E</sub>X, so that changing the fonts was a difficult, if not impossible, task.

Since that time low-priced laser printers have become available and simultaneously a large number of font families from PostScript and other type formats have appeared. The number of fonts in METAFONT source format (freely available to every (L)T<sub>E</sub>X installation) has also increased drastically. But, unfortunately, there was no easy and standard method for integrating these new fonts into L<sup>A</sup>T<sub>E</sub>X—typesetting with L<sup>A</sup>T<sub>E</sub>X meant typesetting in Computer Modern on almost all installations. Of course, individual fonts could be loaded using the \newfont command, but this capability cannot be called integration: it requires a great deal of user intervention, because the additional fonts do not change size under the control of size commands, and it was extremely complicated to typeset a whole document in a font family.

There have been a few efforts to integrate other fonts into L<sup>A</sup>T<sub>E</sub>X. Typically, they involved exchanging one hard-wired font table with another. Thus, the resulting L<sup>A</sup>T<sub>E</sub>X variant was as inflexible as the original one, as this approach merely forced the use of a different set of fonts.

This unsatisfactory situation was finally resolved in 1989 with the release of the New Font Selection Scheme (NFSS) [128, 130] written by Frank Mittelbach and Rainer Schöpf, which became widely known after it was successfully used in *AMS-L<sup>A</sup>T<sub>E</sub>X* (see Chapter 8). This system contains a generic concept for varying font attributes individually and for integrating new font families easily into an existing L<sup>A</sup>T<sub>E</sub>X system. The concept is based on five attributes that can be defined independently to access different fonts, font characteristics, or font families. To

implement it, some of the  $\text{\LaTeX}$  commands were redefined and some new commands were added.

Later, a prototype version for scalable fonts was coded by Mark Purtill. Starting from his work, Frank Mittelbach designed and implemented NFSS2 integrating work by Sebastian Rahtz (on PostScript fonts) and several others. This version became the standard  $\text{\LaTeX}$  font selection scheme in 1994, when the current  $\text{\LaTeX}$  version ( $\text{\LaTeX} 2\epsilon$ ) was released.

This font selection scheme has now been in worldwide use for more than a decade and the code has proven to be stable and successful, though some people feel that extensions would be useful. The  $\text{\LaTeX}$  Project Team would welcome such experimental extensions in the form of external packages, which at a later stage might be consolidated into a successor of the base font selection mechanism.

### 7.1.2 Input and output encodings

As one of the side effects of being able to access more fonts, it became apparent that two related areas in  $\text{\TeX}$  made hard-wired selections no longer appropriate: the areas of input and output (or font) encodings.

If we press a key on a keyboard (usually) some 8-bit number will be generated representing a certain character. An input encoding describes which character corresponds to which number. When using different national keyboards or different operating systems, the correspondence between character and number may vary widely. For example, on the German keyboard that the author used to write this text, the key labeled “ä” will generate the 8-bit number “228” when used with Linux or Windows, but it generates “132” when used with MS-DOS.

When your document is stored in a computer file, information that remains about the characters consists of only these 8-bit numbers; the information about the input encoding used is not explicitly stored. Thus, if you transfer a file to a different environment, such as, from the United States to the United Kingdom, you might find that the dollar signs in your document are suddenly interpreted as pound symbols when viewing your file with some program (editor) that makes the wrong assumption about the encoding used to write the file.

To help with input encoding problems, in 1994–1995 the  $\text{\LaTeX}$  Project Team developed the `inputenc` package. It enables users to explicitly declare the input encoding used for documents or parts of documents. This mechanism allows you to safely transfer documents from one  $\text{\LaTeX}$  installation to another and to achieve identical printed results.<sup>1</sup>

The `inputenc` package works by interpreting the 8-bit numbers present in the file (representing the characters) and mapping them to an “internal  $\text{\LaTeX}$  representation”, which uniquely (albeit on a somewhat ad hoc basis) covers all characters representable in  $\text{\LaTeX}$ . For further processing, such as writing to some auxiliary

*The input encoding concept*

<sup>1</sup>Other solutions to this problem exist. For example, some people advertise the use of translation tables hard-wired into the program  $\text{\TeX}$  itself. This works as long as all people exchanging documents use a  $\text{\TeX}$  system with the same hard-wired tables but fails otherwise.

file,  $\text{\LaTeX}$  exclusively uses this internal representation, thereby avoiding any possible misinterpretation.

However, at some point  $\text{\LaTeX}$  has to associate these internal character representations with glyphs (i.e., character shapes in certain fonts) so another mapping must take place.  $\text{\TeX}$ 's fonts contain a maximum of 256 glyphs. These glyphs are not addressed by name, but rather by (8-bit) numbers representing the positions of the glyphs in the font (i.e., we have to map from a large unique naming space into several small ones). And it probably does not come as a large surprise to hear that these glyph positions again vary widely.

Thus, even after preserving the meaning of our dollar sign from the external file to the internals of  $\text{\LaTeX}$ , we might still end up with the wrong shape on paper if we happen to select a font for printing that contains an unexpected glyph in the position (slot) we assumed was occupied by a dollar sign.<sup>1</sup> It is one of the tasks of NFSS to ensure either that any  $\text{\LaTeX}$  internal character representation is properly rendered or, if that is impossible for some reason, that the user receives a proper error message.

If fonts contain accented characters as individual glyphs, rather than only base characters plus accents (from which  $\text{\TeX}$  then has to build up the accented glyphs internally), then it is preferable to use these glyphs because they typically have a better appearance. There is also a technical reason for this preference: the `\accent` primitive of  $\text{\TeX}$  will suppress hyphenation. This defect might be acceptable if such words are occurring only infrequently, as when typesetting English. However, when dealing with, say, a French text in which all words with accents are never hyphenated, line breaking soon becomes a nightmare.

To cater to the different possibilities, a command such as `\'e` ( $\text{\LaTeX}$ 's internal representation for the character e-acute, é) sometimes has to initiate some complicated actions involving the `\accent` primitive. In other cases it merely informs the paragraph builder that it wants the glyph from a certain slot in the current font.

All this is achieved in  $\text{\LaTeX}$  through the concept of output encodings, which map the  $\text{\LaTeX}$  internal character representations to appropriate glyph positions or to glyph-building actions depending on the actual glyphs available in the font used for typesetting. Although the output encoding concept was fully introduced with NFSS2, it took several years to finally settle on its current implementation (the internals were rewritten several times while the developers were gaining more insight into the problems in this area).

\* \* \*

The following sections describe release 2 of NFSS, which was completed at the end of 1992 and became part of standard  $\text{\LaTeX}$  in 1994. As far as the user interface is concerned, it is intended for integration into  $\text{\LaTeX}3$ .

We start by discussing font characteristics in general and introduce the major attributes used in  $\text{\LaTeX}$  for orthogonal font switching. We then describe the use of

<sup>1</sup>The example of the \$ turning into a £ sign is not artificial: some of the original  $\text{\TeX}$  fonts show this strangeness, and Knuth [82, p.339] even advocates typesetting a pound symbol using `\it\$`.

*Made-up  
accented  
characters prevent  
hyphenation*

*The output encoding  
concept*

the high-level interface—that is, the commands a user normally has to deal with. This includes commands used in normal text (Section 7.3), special features for use in mathematical formulas (Section 7.4), and an overview of basic support packages for NFSS—those being distributed together with L<sup>A</sup>T<sub>E</sub>X (Section 7.5). It also covers the packages and commands provided to deal with the encoding issues mentioned earlier.

One of the important advantages of L<sup>A</sup>T<sub>E</sub>X's font selection scheme is the ease with which new fonts for use in the main text can be integrated. Besides the Computer Modern families, which are used by default, one can easily use other font families by adding the appropriate package in the preamble. Of course, for successful processing and printing the corresponding font files (e.g., the .tfm and .pk, Type 1, or TrueType files) must be installed on the system. The next three sections deal with major and minor font packages. Section 7.6 discusses PSNFSS, the standard PostScript support for L<sup>A</sup>T<sub>E</sub>X, which is part of the required set of packages available with any L<sup>A</sup>T<sub>E</sub>X distribution.

This is followed by a collection of other interesting packages for adjusting the document body fonts (Section 7.7) and by an introduction to the L<sup>A</sup>T<sub>E</sub>X world of symbols (Section 7.8). All packages described are available free of charge, and most (if not all) are part of a modern L<sup>A</sup>T<sub>E</sub>X distribution. Some pointers to commercial font support are given as well.

The final part of this chapter describes the low-level interfaces that are useful when defining complex new commands and that are important when new fonts are to be made available in L<sup>A</sup>T<sub>E</sub>X. Here you will find low-level commands for changing individual font attributes (Section 7.9), commands for setting up new fonts with L<sup>A</sup>T<sub>E</sub>X (Section 7.10), and a discussion of L<sup>A</sup>T<sub>E</sub>X's encoding models for text and math (Section 7.11). The chapter concludes with a section devoted to compatibility questions that arise with very old L<sup>A</sup>T<sub>E</sub>X documents.

## 7.2 Understanding font characteristics

There are many design principles that divide fonts into individual overlapping classes. Knowledge of these characteristics often proves helpful when deciding which font family to use in a special context (for further reading see, for example, the books [28, 41, 116] or the article [52]).

### 7.2.1 Monospaced and proportional fonts

Fonts can be either monospaced or proportionally spaced. In a monospaced font, each individual character takes up the same horizontal space regardless of its shape. In contrast, characters in a proportionally spaced font take up different amounts of space depending on their shape. In Figure 7.1 on the following page, you can see that the “i” of the monospaced font occupies the same space as the “m”, while it is noticeably narrower in the proportional font. As a result, proportional fonts (also called typographical fonts) normally allow more words to be

i i i i i i i m m m m m m m m (monospaced)	i i i i i i i i m m m m m m m m m m (proportionally spaced)
--	---

Figure 7.1: Major font characteristics

placed on a page and are more readable than monospaced fonts. The extra spaces around individual characters of monospaced fonts make it more difficult for the eye to recognize word boundaries and thus make monospaced text less readable.

However, monospaced fonts do have their uses. Within the proper context, they enhance the quality of the printed document. For example, in tables or computer listings where proper alignment of information is important, a monospaced font is a natural choice. In computer science books, it is common practice to display computer programs in a monospaced font to make them easily distinguishable from surrounding explanations.

But the use of monospaced fonts goes beyond marking portions of a document as special. One can even consider choosing a monospaced font as the base font for a complete document. Such a font has the flavor of the manual or electric typewriter engine; it looks hand-made when used with unjustified paragraphs and therefore may be better suited to certain situations than a more professional-looking typographical font. Keep in mind, however, that monospaced fonts look very poor when lines are justified. (See Section 3.1.11 to learn how to turn off justification.)

### 7.2.2 Serifed and sans serif fonts

Another useful classification is based on the presence or absence of serifs. Serifs are the tiny strokes at the extremities of character shapes (see Figure 7.2). Originally they were produced by the chisel, when Roman capitals were engraved into stone. For this reason, serifed fonts are often referred to as “Roman” fonts.

Serifed fonts traditionally have been used for long texts because, it was argued, they are more readable. It was long thought that serifed letters give the eye more clues for identification. This is certainly true if only parts of the characters are visible, but for fully visible text recent research has shown that reading speed is not substantially affected by the absence of serifs [150].



Figure 7.2: Comparison of serifed and sans serif letters

A	B	C	a	b	c	x	y	z
A	B	C	a	b	c	x	y	z
<i>A</i>	<i>B</i>	<i>C</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>x</i>	<i>y</i>	<i>z</i>

Figure 7.3: Comparison between upright and italic shapes

### 7.2.3 Font families and their attributes

Besides the crude classifications of serifed versus sans serif and monospaced versus proportional, fonts are grouped into font families. Members of a font family share common design principles and are distinguished by variations in size, weight, width, and shape.

#### Font shapes

An important attribute when classifying a member of a font family is its shape. Of course, sometimes it is a matter of personal judgment whether a set of fonts with different shapes constitutes one or several families. For example, Donald Knuth called his collection of 31 Computer Modern fonts a family [86], yet they form a meta-family of many families in the traditional sense.<sup>1</sup>

Although there is no uniform naming convention for font shapes, this is unimportant as long as one sticks to a particular scheme within L<sup>A</sup>T<sub>E</sub>X.

Nearly every font family has one shape called the “upright” shape.<sup>2</sup> For example, in the font family used in this book (Lucida Bright), the font that you are now reading is in the upright shape. *The upright shape*

Another important shape that is present in most families is the “italic” shape, which looks *like this* in the Lucida Bright family. Italic characters are slanted to the right and the individual letters generally are drawn differently from their upright counterparts, as illustrated in Figure 7.3. The first line in that figure shows letters from the Computer Modern Serif family in upright shape, and the third line shows the same letters in italic shape. For better comparison, the second line gives the italic letters without the usual slant—that is, the letters are artificially shown in an upright position. *The italic shape*

Font families without serifs often lack a proper italic shape; instead, they have a “slanted” shape in which the characters slant to the right but are otherwise identical to their upright counterparts. The terms “sloped” and “oblique” are also commonly used for this shape. *The slanted or oblique shape*

<sup>1</sup>METAFONT, as a design tool, allows the production of completely different fonts from the same source description, so it is not surprising that in 1989 another family was created [92] based on the sources for the Computer Modern fonts. This family, Concrete Roman, was obtained merely by varying some METAFONT parameters in the source files; but since the result was so different, Knuth decided to give this family a different name.

<sup>2</sup>Sometimes you will also hear the term “Roman” shape. This is due to the fact that until recently typesetting was nearly always done using serifed fonts. Thus, “Roman” was considered to be the opposite of “italic” by many people. So be aware that in some books this term actually refers to the upright shape and not to a serifed font family.



Figure 7.4: Comparison between caps and small caps

*The small caps shape* Another common variant is the “small caps” shape, in which the lowercase letters are represented as capitals with a reduced height, as shown in Figure 7.4. If such a shape is not available for a specific family, typographers sometimes use upright capitals from smaller sizes,<sup>1</sup> but this practice does not produce the same quality as a well-designed small caps font. Real small caps have different widths and weight than capital letters from the same font that have been reduced to the height of designed small caps (you can clearly see that the strokes in the faked capitals in Figure 7.4 are much too thin).

*Taking small capitals*

It is an open argument whether one should consider “small caps” to be a shape or whether this would be better modeled as another independent axis. In the latter interpretation, fonts have a “case” attribute, which could be either mixed case (the normal case), all caps, small caps, or all lowercase. For certain font families this would certainly be the better solution, but currently the L<sup>A</sup>T<sub>E</sub>X font selection supports only four axes modeling small caps as a shape.<sup>2</sup>

There are a few other, less important shapes. Some families contain fonts in which the inner parts of the letters are drawn in a special fashion, most importantly perhaps the “outline” shapes, in which the inner parts of the letters are kept empty. For display purposes, some families also contain fonts that could be classified as “shaded”—that is, where the letters appear three-dimensional. Examples are shown in Figure 7.5 on the facing page.

Special variants of the Computer Modern meta-family have been produced by setting the METAFONT parameters to special values. For example, there is “upright italic”, a shape in which the individual letters are drawn in italic fashion but without the usual slant (see the second line in Figure 7.3 on the previous page). This shape was devised for purposes of showing the abilities of METAFONT as a tool for meta-design, but some users might take a fancy to such an unusual shape.

### Weight and width

Fonts of a certain shape within a family may differ in “weight”. This characteristic refers to the thickness of the strokes used to draw the individual shapes. Once again, the commonly used names are not completely uniform, but it is relatively

<sup>1</sup>A good rule of thumb is to use capitals from a font that is about half a point larger than the x-height of the original font unless the x-height is very small. See discussion in Section 7.10.3 on page 428 for a way to determine the x-height of any font used with T<sub>E</sub>X.

<sup>2</sup>In some cases small caps fonts are in fact modeled as extra families to enable the combination of, say, small caps italic.



Figure 7.5: Outline and shaded shapes

easy to arrive at a consistent classification. Some font manufacturers, for example, call the font weights intended to be used for normal text “book”, while others call them “medium”. For thin strokes the name “light” is commonplace, while thicker strokes are usually called “bold”. In larger font families, finer distinctions are often necessary, so that we sometimes find a range starting with “ultra light”, going through “extra light”, “light”, “semi light”, and so on, and ending with “ultra bold” at the other end. Conversely, often only a few weights are present in some families. For example, the Computer Modern Roman family has only two weights, “medium” and “bold”.

Another equally important attribute of a font is its “width”—the amount of expansion or contraction with respect to the normal or medium width in the family. Computer Modern Roman has bold fonts in “**medium width**” and “**extended width**”. One application for condensed fonts is in titles and headings, where medium-width fonts, when used at large sizes, would consume too much space. Some typesetting systems can even condense fonts automatically to fit a given measure—for example, to exactly fill a particular line in a heading. This capability is not directly possible with (L)TeX, but in any case the results are often aesthetically questionable.

### Font sizes

Font sizes are traditionally measured in printer points (pt). There are 72.27 points to an inch.<sup>1</sup> The font size is not an absolute measure of any particular characteristic, but rather a value chosen by the font designer to guide the user. For example, in a 10pt font, letters of the alphabet are usually less than 10pt tall, and only characters such as parentheses have approximately this height.

Two fonts of the same size may not blend well with one another because the appearance of a font depends on many factors, such as the height of the lowercase letters (the x-height), the stroke width, and the depth of the descenders (the part of the letters below the baseline, as in the letter q).

In the (L)TeX world, fonts are often available in sizes that are powers of 1.2—that is, in a geometric progression [82, p.17]. This arrangement was chosen because it makes it easy to produce an enlarged master copy that later can be photographically reduced, thereby effectively enlarging the final output resolution. For example, if an A5 brochure is to be produced, one could print it with magnifica-

<sup>1</sup>PostScript uses a slightly different measurement system in which 72 points equal an inch. These units, sometimes referred to as “big points”, are available in TeX as bp.

Ten point type is different from magnified five point type

Figure 7.6: Scaled and designed fonts (Computer Modern)

tion of  $1.44 \approx \sqrt{2}$  on A4 paper. Photographic reduction from the 300 dpi (dots per inch) output of a normal laser printer would produce an effective output resolution of 432dpi and thus would give higher quality than is normally possible with such a laser printer.

However, this geometric ratio scheme used by (L<sup>A</sup>T<sub>E</sub>X fonts produced with the METAFONT program is not common in the professional world, where usual point sizes are 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 24, 30, and 36. Yet not all fonts are available in all these sizes, and sometimes additional sizes are offered—such as display sizes for large headings and tiny sizes for subscripts and superscripts. The requirement for fixed sizes had its origin in the technology used. Fonts cast in metal had to exist (at a particular size) or you could not print in that size. In today's digitalized world, fonts are usually vectorized and thus can be scaled at will. As a result, many commercial font families nowadays are provided in only a single design size.

The use of magnified or reduced fonts instead of fonts designed for a specific size often gives somewhat less satisfactory results, because to the human eye fonts do not scale in a linear fashion. The characters in handcrafted fonts of larger sizes usually are narrower than fonts magnified from a smaller size of the same family. While it is acceptable to scale fonts within a small size range if necessary, one should use fonts designed for the desired size whenever possible. The difference between fonts scaled to a particular size and those designed for that size is shown in Figure 7.6, though admittedly the variations are often less noticeable.

#### 7.2.4 Font encodings

As mentioned in the chapter introduction, T<sub>E</sub>X refers to the glyphs of a font by addressing them via 8-bit numbers. Such a mapping is called a font encoding. As far as L<sup>A</sup>T<sub>E</sub>X is concerned, two fonts having the same font encoding are supposed to be interchangeable in the sense that given the same input they produce the “same” glyphs on the printed page. To illustrate what happens if we use a font with an encoding not suitable for our input, here is the first sentence of this section again (using the Zapf Dingbats font):

```
◊▲ ○*■▼*□■*** *■ ▼*** *-*○□▼*□ *■▼□□*◆*▼*□■§ TEX
□***□▲ ▼□ ▼*** *●□*▲ □* ● □■▼ ◊! ○***□*▲▲*■* ▼***○
❖*● ✕▲❖*▼ ■◆○○*□▲❖
```

The result is an interesting puzzle, but nothing that we want to see in ordinary documents.

By classifying fonts according to their font encodings it is possible to modify other font characteristics, such as font family or font series, and still ensure that the typeset result will stay comprehensible.

The fonts that were originally distributed with TeX have only 128 glyphs per font and therefore do not include any accented characters as individual glyphs. Instead, all such glyphs have to be constructed using the \accent primitive of TeX or by similar methods. As a result any word containing diacritics cannot be automatically hyphenated by L<sup>A</sup>T<sub>E</sub>X and kerning (correction of spacing between certain letters in the font) cannot be automatically applied. The encoding of these fonts is called OT1. Although it remains the default encoding for L<sup>A</sup>T<sub>E</sub>X, it is not advisable to use OT1 for languages other than English.

As an alternative encoding, the TeX user community defined a 256-character encoding called T1 that enables TeX to typeset correctly (with proper hyphenation and kerning) in more than 30 languages based on the Latin alphabet (see Section 7.5.1 on page 353 for further details). The use of the T1 encoding is, therefore, highly recommended. Nowadays nearly all font families amenable to use with L<sup>A</sup>T<sub>E</sub>X are available in this encoding; in fact, some are *only* available in the T1 encoding. Specifying \usepackage[T1]{fontenc} after the \documentclass command, makes T1 become the default encoding. Section 7.5.3 contains a more detailed discussion of the **fontenc** package. For more on font encodings refer to page 415 and Section 7.11 on page 440.

## 7.3 Using fonts in text

When you are writing a L<sup>A</sup>T<sub>E</sub>X document, appropriate fonts are normally chosen automatically by the (logical) markup tags used to structure the document. For example, the font attributes for a section heading, such as large size and bold weight, are defined by the document class and applied when a \section command is used, so that you seldom need to specify font attributes yourself.

However, occasionally it becomes necessary to specify font attributes directly. One common reason is the desire to change the overall font attributes, by choosing, for example, a different font family for the main text. This alteration often can be done by simply specifying an appropriate package (see Sections 7.6 and 7.7 for descriptions of such packages).

Another use for explicit font attributes can be to mark certain portions of the document as special—for example, to denote acronyms, example, or company names. For instance, in this book, names of packages are formatted in a sans serif font. This formatting could be achieved by surrounding the names with \textsf{..}, but it is much better practice to define a new command (say, \LPack) for this purpose so that additional information is included in the source document. By defining individual commands for logically different things—even those that are currently being typeset in the same way—it is easier to change the formatting later in a consistent way.

Last, but not least, in some cases you may want to override a decision taken by the document class. For example, you might want to typeset a table in a smaller size to make it fit on a page. This desire is legitimate, as document classes can format documents automatically only to a certain extent. Hand-formatting—like the insertion of page breaks—is thus often necessary to create the final version. Unfortunately, explicit formatting makes further use of the document (if changes are made) difficult and error prone. Therefore, as with all visual formatting commands, you should try to minimize the direct use of font-changing commands in a document.

### 7.3.1 Standard L<sup>A</sup>T<sub>E</sub>X font commands

The font used for the main text of a document is called the “main font”, “body font”, or “normal font”. It is automatically selected at the beginning of the document and in certain constructs, such as footnotes, and figures. Certain logical markup tags, such as section headings, automatically switch to a different typeface or size, depending on the document class. These changes happen behind the scenes, and the only action required of the author is to introduce the correct logical markup in the document. However, sometimes it might be desirable to manually highlight individual parts of the text, by choosing an appropriate typeface; this is done with the commands described below.

Most font-changing commands come in two forms: a command with one argument, such as `\textbf{...}`, and a declarative form, such as `\bfseries`. The declarations do not take arguments but rather instruct L<sup>A</sup>T<sub>E</sub>X that from now on (up to the end of the current group of braces or environments) it should behave in a special way. Thus, you should not write something like `\bfseries{...}`, as this would make everything bold from this point until the end of the current environment.

To change the fonts for individual words or short phrases within your document you should make use of the font commands with one argument. The declarative forms are often better in the definition of new environments or commands. For longer passages in your document, you can also use the environment form of the declaration (the declarative name without the preceding backslash), as shown in the following example:

Some words in this sentence are <b>typeset in bold letters.</b> <b>The bold typeface</b> continues here.	Some words in this sentence are <code>\begin{bfseries}typeset in bold letters.</code> <code>\end{bfseries}</code> continues here.
---	---

[7-3-1]

In fact, the font commands with one argument do not allow paragraph breaks in their arguments. Section 7.3.3 on page 344 contains a detailed comparison of the command and declarative forms and their advantages and disadvantages in specific cases.

### The main document font

To switch to the main document font you can use the command `\textnormal` or the declaration `\normalfont`. They are typically used only in the definition of commands or environments when it is important to define commands that always typeset in the same font regardless of the surrounding conditions. For example, the command to typeset the command names in this book is defined roughly as follows:

```
\newcommand{\Lcs}[1]{\normalfont\ttfamily\textbackslash#1}%
\index{\normalfont\ttfamily\textbackslash#1}}
```

Using `\normalfont` prevents the command names coming out *like \this in certain places*.

### Standard font families

By default, L<sup>A</sup>T<sub>E</sub>X maintains three font families that can be selected with short command sequences. These families are a serifed text font, accessed with the command `\textrm`; a sans serif text font, accessed by `\textsf`; and a typewriter font (usually monospaced), accessed by `\texttt`. The declaration forms of these commands are `\rmfamily`, `\sffamily`, and `\ttfamily`, respectively.

The names of the external font families accessed by these commands depend on the document class but can be changed by packages or in the preamble (see Section 7.3.5). As an installation default, the serifed font family is Computer Modern Roman, the sans serif family is Computer Modern Sans, and the typewriter family is Computer Modern Typewriter. If you use a different set-up, take care to define these default families so that the fonts can be mixed freely without visual clashes. Also, make sure that the external fonts are available in the correct resolution for the targeted output device.

In this book, the serifed font family is Lucida Bright, the sans serif family is Lucida Sans, and the typewriter family is European Modern Typewriter. These have been chosen by simply<sup>1</sup> loading the package `lucidabr` and afterwards redefining `\ttdefault` to produce `emtt`; see Section 7.3.5 for more details on changing the default text fonts.

In most document classes, the serifed font, accessed by `\textrm`, is also the main font of the document, so the command `\textrm` is not used often. But if a document designer has chosen a sans serif font as the main typeface, then `\textrm` would be the alternative serifed font family.

<sup>1</sup>Somewhat more truthful: for the second edition of this book the Lucida fonts were scaled down slightly, while the European Modern Typewriter was scaled up to match the x-height of both families using specially designed `\DeclareFontShape` declarations.

### Standard font series

Another attribute of a typeface that can be changed is the *series*. In L<sup>A</sup>T<sub>E</sub>X the series is a combination of two attributes: width and weight (boldness). L<sup>A</sup>T<sub>E</sub>X provides two commands for changing the series: `\textmd` and `\textbf`. The corresponding declarations are `\mdseries` and `\bfseries`, respectively. The first command selects a font with medium values for the width and the weight, while the latter switches to a bolder series. The actual values depend on the document class and its options or subsequent packages. As a default for the Computer Modern families, `\textbf` switches to a bold extended version of the current typeface, while `\textmd` returns to the medium width and medium weight version of the current typeface.

If finer control over the series attribute is desired, it is best to define additional high-level user commands with the help of the lower-level `\fontseries` declaration described in Section 7.9.1. Some packages that make large font families available for use with L<sup>A</sup>T<sub>E</sub>X provide such extra commands.

### Standard font shapes

A third font attribute that may be changed independently of the others is the *shape* of the current typeface. The default shape for most documents is the upright shape. It can be accessed, if necessary, with the command `\textup` or the declaration `\upshape`.

Probably the most important commands for changing the shape are `\textit` and `\textsc`, which switch to an *italic* or CAPS AND SMALL CAPS font shape, respectively. The corresponding declarations are `\itshape` and `\scshape`.

An alternative to `\textit` is the `\textit{sl}` command (its declaration form is `\itshape{sl}`), which switches to the slanted shape. A font family often contains only an italic or a slanted shape, yet Computer Modern Roman contains both.

At the point where one switches from slanted to upright, the characters usually come too close together, especially if the last slanted character has an ascender. The proper amount of extra white space that should be added at this boundary is called the “italic correction”. The value of this adjustment depends on the individual character shape and is stored in the `.tfm` file. The italic correction is automatically added by the font commands with arguments but it must be inserted manually using `\vphantom` when declarations are employed. For an upright font, the italic correction of the characters is usually zero or very small, but there are some exceptions. (In Computer Modern, to typeset a bold ‘f’ in single quotes, you should say ‘`\bfseries f\vphantom{f}`’ or ‘`\textbf{f}\vphantom{f}`’, lest you get a bold ‘f’ in some fonts.) In slanted or italic fonts, the italic correction is usually positive, with the actual value depending on the shape of the character. The correct usage of shape-changing declarations that switch to slanted shapes is shown in the next example.

When switching back from *italic* or *slanted* shapes to an upright font one should add the *italic correction*, except when a small punctuation character follows.

7-3-2

```
\raggedright
When switching back from {\itshape italic\}/} or
{\slshape slanted\}/} shapes to an upright font
one should add the {\itshape italic correction\},
except when a small punctuation character follows.
```

If you use the command forms with one argument instead, the italic correction is added automatically. This topic is further discussed in Section 7.3.3.

Small capitals are sometimes used in headings or to format names. For the latter case you can, for example, define the command `\name` with the definition

```
\newcommand\name[1]{\textsc{#1}}
```

or, using two declarations:

```
\newcommand\name[1]{{\normalfont\scshape #1}}
```

The first definition simply switches to the desired shape, while the second form initially resets all font attributes to their defaults. Which approach is preferable depends on the available fonts and the type of document. With Computer Modern only the Roman and typewriter families contain a small caps shape, so the second definition might be preferred in certain applications because it will use small caps (though serifed) even in a `\sffamily` context. The first command would result in a request for a medium series, small caps, shaped font in the Computer Modern Sans family. Because this font is not available, L<sup>A</sup>T<sub>E</sub>X would try to find a substitute by first changing the shape attribute to its default, with the result that you would not get small caps. (See Section 7.9.3 for further information about substitutions.)

Another interesting use of the `\scshape` declaration is in the definition of an acronym tag:

```
\newcommand\acro[1]{{\scshape\MakeLowercase{#1}}}
```

This definition makes use of the L<sup>A</sup>T<sub>E</sub>X command `\MakeLowercase`, which changes all characters within its argument to lowercase (in contrast to the T<sub>E</sub>X primitive `\lowercase`, this command also changes characters referred to by commands, such as `\OE`, to lowercase). As a result, all characters in the argument of `\acro` will be changed to lowercase and therefore typeset with small capitals.

Another slightly special shape command available in L<sup>A</sup>T<sub>E</sub>X is the `\emph` command. This command denotes emphasis in normal text; the corresponding declaration is `\em`. Traditionally, emphasized words in text are set in italic; if emphasis is desired in an already italicized portion of the text, one usually returns to the upright font. The `\emph` command supports this convention by switching to the `\itshape` shape if the current font is upright, and to the `\upshape` shape if the current font is already slanted (i.e., if the shape is `\itshape` or `\slshape`). Thus,

<code>\tiny</code>	Size	<code>\normalsize</code>	Size	<code>\huge</code>	<b>Size</b>
<code>\scriptsize</code>	Size	<code>\large</code>	Size		
<code>\footnotesize</code>	Size	<code>\Large</code>	Size		
<code>\small</code>	Size	<code>\LARGE</code>	Size	<code>\Huge</code>	<b>Size</b>

*The actual sizes shown above are those specially tailored for use in this book*

Table 7.1: Standard size-changing commands

the user does not have to worry about the current state of the text when using the `\emph` command or the `\em` declaration.

*Nevertheless, one has to be careful about the proper use of italic corrections on both ends of the emphasized text.* It is therefore better to use the `\emph` command, which *automatically* takes care of the italic correction on both sides.

{\em Nevertheless, one has to be careful about the\`/ {\em proper\`/} use of italic corrections on both ends of the emphasized text}. It is therefore better to use the `\verb=\emph=` command, which `\emph{automatically}` takes care of the italic correction on both sides.

7-3-3

Using the upright shape for nested emphasis is not always very noticeable. A common typographic recommendation is, therefore, to use small capitals for the inner emphasis. This practice is not directly supported by standard L<sup>A</sup>T<sub>E</sub>X but can be achieved through the command `\eminnershape`, made available by the fixltx2e package.

```
\usepackage{fixltx2e}
\renewcommand\eminnershape{\scshape}
{\em Nevertheless, one has to be careful about
the\`/ {\em proper\`/} use of italic corrections
on both ends of the emphasized text}.
```

7-3-4

*Nevertheless, one has to be careful about the PROPER use of italic corrections on both ends of the emphasized text.*

Note that underlining for emphasis is considered bad practice in the publishing world. Underlining is used only when the output device can't do highlighting in another way—for example, when using a typewriter. Sections 3.1.6 and 3.1.7 discuss packages that change `\em` to produce underlining.

### Standard font sizes

L<sup>A</sup>T<sub>E</sub>X has 10 size-changing commands (see Table 7.1). Since size changes are normally used only in the definition of commands, they have no corresponding command forms with one argument. The names of the commands have been retained from L<sup>A</sup>T<sub>E</sub>X 2.09 but in today's L<sup>A</sup>T<sub>E</sub>X their functionality has changed slightly. In L<sup>A</sup>T<sub>E</sub>X 2<sub>E</sub> such a command changes only the size of the current font, with all other attributes staying the same; in L<sup>A</sup>T<sub>E</sub>X 2.09 a size-changing command also automatically switched back to the main document font.

The size selected by these commands depends on the settings in the document class file and possibly on options (e.g., `11pt`) specified with it. In general, `\normalsize` corresponds to the main size of the document, and the size-changing commands form an ordered sequence starting with `\tiny` as the smallest and going up to `\Huge` as the largest size. Sometimes more than one command refers to the same real size; for example, when a large `\normalsize` is chosen, `\Huge` can be the same as `\huge`. In any event, the order is always honored.

The size-related commands for the main text sizes (i.e., `\normalsize`, `\small`, and `\footnotesize`) typically influence the spacing around lists and displays as well. Thus, to change their behavior, one should not simply replace their definition by a call to `\fontsize`, but instead start from their original definitions, as documented in `classes.dtx`.

Unfortunately, there is currently no relative size-changing command in L<sup>A</sup>T<sub>E</sub>X—for example, there is no command for requesting a size 2pt larger than the current one. This issue is partially resolved with the `relsize` package described in Section 3.1.4 on page 83.

### 7.3.2 Combining standard font commands

As already shown, the standard font-changing commands and declarations can be combined. The result is the selection of a typeface that matches the combination of all font attributes. For example:

One can typeset a text **in a large sans serif bold typeface** but note the unchanged leading! L<sup>A</sup>T<sub>E</sub>X uses the value in force at the *end* of the paragraph!

One can typeset a text  
{\sffamily\bfseries\large  
in a large sans serif bold typeface}  
but note the unchanged leading!  
\LaTeX{} uses the value in force at  
the \emph{end} of the paragraph!

What happens behind the scenes is that the `\sffamily` command switches to the sans serif default family, then `\bfseries` switches to the default bold series in this family, and finally `\large` selects a large size but leaves all other font attributes unchanged (the leading appears to be unchanged because the scope of `\large` ends before the end of the paragraph). Font metric files (i.e., `.tfm` files) are loaded for all intermediate typefaces, even if these fonts are never used. In the preceding example, they would be “sans serif medium 10pt” after the `\sffamily`, then “sans serif bold extended 10pt” after the `\bfseries`, then “sans serif bold extended 14pt”, which is the font that is finally used. Thus, such high-level commands can force L<sup>A</sup>T<sub>E</sub>X’s font selection to unnecessarily load fonts that are never used. This normally does not matter, except for a small loss of processing speed when a given combination is used for the first time. However, if you have many different combinations of this type, you should consider defining them in terms of the primitive font-changing declarations (see Section 7.9).

<i>Command</i>	<i>Corresponds to</i>	<i>Action</i>
<code>\textrm{...}</code>	<code>{\rmfamily...}</code>	Typeset text in Roman family
<code>\textsf{...}</code>	<code>{\sffamily...}</code>	Typeset text in sans serif family
<code>\texttt{...}</code>	<code>{\ttfamily...}</code>	Typeset text in typewriter family
<code>\textmd{...}</code>	<code>{\mdseries...}</code>	Typeset text in medium series
<code>\textbf{...}</code>	<code>{\bfseries...}</code>	Typeset text in <b>bold</b> series
<code>\textup{...}</code>	<code>{\upshape...}</code>	Typeset text in upright shape
<code>\textit{...}</code>	<code>{\itshape...}</code>	Typeset text in <i>italic</i> shape
<code>\textsl{...}</code>	<code>{\slshape...}</code>	Typeset text in <i>slanted</i> shape
<code>\textsc{...}</code>	<code>{\scshape...}</code>	Typeset text in SMALL CAPS shape
<code>\emph{...}</code>	<code>{\em...}</code>	Typeset text <i>emphasized</i>
<code>\textnormal{...}</code>	<code>{\normalfont...}</code>	Typeset text in the document font

Table 7.2: Standard font-changing commands and declarations

### 7.3.3 Font commands versus declarations

We have already seen some examples of font commands that have arguments and change font attributes. These font-changing commands with arguments all start with `\text...` (except for the `\emph` command) to emphasize that they are intended for use in normal text and to make them easily memorizable. Using such commands instead of the declarative forms has the advantage of maintaining consistency with other L<sup>A</sup>T<sub>E</sub>X constructs. They are intended for typesetting short pieces of text in a specific family, series, or shape. Table 7.2 shows the effects of these commands.

A further advantage of these commands is that they automatically insert any necessary italic correction on either side of their argument. As a consequence, one no longer has to worry about forgetting the italic correction when changing fonts.

Only in a very few situations is this additional space wrong. For example, most typographers recommend omitting the italic correction if a small punctuation character, like a comma, directly follows the font change. As the amount of correction required is partly a matter of taste, you can define in which situations the italic correction should be suppressed. This is done by specifying the characters that should cancel a preceding italic correction in the list `\nocorrlist`.<sup>1</sup> The default definition for this command is

```
\newcommand{\nocorrlist}{,,}
```

It is best to declare the most often used characters first, as it will make the processing slightly faster.

<sup>1</sup> Any package that changes the `\catcode` of a character inside `\nocorrlist` must redeclare the list. Otherwise, the changed character will no longer be recognized by the suppression algorithm.

In addition to the global customization, it is possible to suppress the italic correction in individual instances. For this purpose, the command `\nocorr` is provided. Note that you have to put `\nocorr` on the left or right end inside the argument of the `\text{...}` commands, depending on which side of the text you wish to suppress the italic correction.

*When using the  $\text{\LaTeX}$  high-level font commands, the proper use of italic corrections is automatically taken care of. Only sometimes one has to help  $\text{\LaTeX}$  by adding a `\nocorr` command.*

7-3-6

`\emph{When using the \LaTeX{} high-level font commands, the \emph{proper} use of italic corrections is automatically taken care of. Only \emph{sometimes} one has to help \LaTeX{} by adding a \verb=\nocorr= command.}`

In contrast, the use of the declaration forms is often more appropriate when you define your own commands or environments.

- This environment produces boldface items.
- It is defined in terms of  $\text{\LaTeX}$ 's `itemize` environment and NFSS declarations.

7-3-7

```
\newenvironment{bfitemize}{\begin{itemize}%
  \normalfont\bfseries\raggedright\end{itemize}}%
\begin{bfitemize}
\item This environment produces boldface items.
\item It is defined in terms of \LaTeX's
      \texttt{itemize} environment and NFSS declarations.
\end{bfitemize}
```

### 7.3.4 Accessing all characters of a font

Sometimes it is impossible to enter a character directly from the keyboard, even though the character exists in the font. Therefore, many useful characters are accessible via command names like `\ss` or `\AE`, which produce “ß” and “Æ”, respectively. Some characters can also be implicitly generated from sequences of letters (this is a property of fonts) like `ffi`, which produces “ffi”, and `---`, which produces “—” in the standard  $\text{\TeX}$  fonts.

In addition, the command `\symbol` allows you to access any character in a font by giving its number in the current encoding scheme as either a decimal, octal (preceded by ‘`\`’), or hexadecimal (preceded by “`\x`”) number.

In the Cork font encoding (T1), characters like `\P`, `\$`, and `\_` are included and can be accessed with the `\symbol` command.

7-3-8

```
\fontencoding{T1}\selectfont
In the Cork font encoding (\texttt{T1}),
characters like \symbol{"DE}, \symbol{'237},
and \symbol{32} are included and can be
accessed with the \verb=\symbol= command.
```

The numbers corresponding to the characters in any font can be obtained by using the program `nfssfont.tex`, described in Section 7.5.7 on page 369.

<i>Hook</i>	<i>Default value</i>	<i>Description</i>
\encodingdefault	OT1	Encoding scheme for “main font”
\familydefault	\rmdefault	Family selected for “main font”
\seriesdefault	m	Series selected for “main font”
\shapedefault	n	Shape selected for “main font”
\rmdefault	cmr	Family selected by \rmfamily and \textrm
\sfdefault	cmss	Family selected by \sffamily and \textsf
\ttdefault	cmtt	Family selected by \ttfamily and \texttt
\bfdefault	bx	Series selected by \bfseries and \textbf
\mddefault	m	Series selected by \mdseries and \textmd
\itdefault	it	Shape selected by \itshape and \textit
\sldefault	sl	Shape selected by \slshape and \textsl
\scdefault	sc	Shape selected by \scshape and \textsc
\updefault	n	Shape selected by \upshape and \textup

Table 7.3: Font attribute defaults

### 7.3.5 Changing the default text fonts

To make it easier to modify the overall appearance of a document, *L<sup>A</sup>T<sub>E</sub>X* provides a set of built-in hooks that modify the behavior of the high-level font-changing commands discussed in the previous sections. These hooks are shown in Table 7.3. The values of these hooks can be set in package files or in the preamble of a document by using \renewcommand. Suitable values for these commands can be found by looking through the font tables in this chapter.

For example, by writing in the preamble

```
\renewcommand\familydefault{cmss}
```

a whole document would come out in Computer Modern Sans, because this redefinition changes the font family for the main font used by *L<sup>A</sup>T<sub>E</sub>X*. More exactly, the main document font is determined by the values of \encodingdefault, \familydefault, \seriesdefault, and \shapedefault. Thus, you have to make sure that these commands are defined in such a way that their combination points to an existing font shape in *L<sup>A</sup>T<sub>E</sub>X*'s internal tables.

The default value stored in \encodingdefault currently is OT1, which means *Suboptimal encoding default* that *L<sup>A</sup>T<sub>E</sub>X* assumes that most fonts use the original T<sub>E</sub>X encoding. This is actually a compatibility setting: in most circumstances it is better to use the T1 encoding because it contains many additional glyphs that are not available with OT1 and allows proper hyphenation for words with accented characters (see Section 7.5.1). Nowadays, some fonts are made available only in T1; that is, they do not support OT1 at all.

One also has to be aware that not every font encoding is suitable for use as a document-encoding default. A prerequisite is that the encoding must include most of the visible ASCII letters in their standard positions; see the discussion in Section 7.11 on page 440 for details. The `\encodingdefault` can be changed by loading the `fontenc` package with one or more options; see Section 7.5.3. For more information on font encodings refer to Section 7.9.1.

Another example, this time involving a series-changing command, would be to define `\bfdefault` to produce `b` so that the `\bfseries` command will use **bold** instead of **bold extended**, which is the default under Computer Modern. However, there is some risk in using such a setting since, for example, in Computer Modern only the Roman family has bold variants with a medium width. Computer Modern Typewriter and Computer Modern Sans have only bold extended variants. Thus, without further adjustments, a request for a bold sans serif font (i.e., `\sffamily\bfseries`), for example, might force L<sup>A</sup>T<sub>E</sub>X to try font substitution, and finally select a medium-weight font. (This outcome can be avoided, as explained in Section 7.10.3, by specifying that the bold extended variants of the sans family should serve as substitutes for the bold medium ones.)

An example in which some default values are changed can be found in Section 7.10.8 on page 439, which covers setting up PostScript manually.

The initial setting of `\familydefault` means that changing `\rmdefault` will implicitly also change `\familydefault` to the new value, as long as no special setting for `\familydefault` is defined. However, if `\familydefault` is changed, `\rmdefault` is not affected.

 *Wrong bold  
default can lead  
to problems*

### 7.3.6 L<sup>A</sup>T<sub>E</sub>X 2.09 font commands

The two-letter font commands used in L<sup>A</sup>T<sub>E</sub>X 2.09, such as `\bf`, are no longer defined by L<sup>A</sup>T<sub>E</sub>X<sub>2ε</sub> directly. Instead, they are defined (if at all) in the L<sup>A</sup>T<sub>E</sub>X<sub>2ε</sub> class files. For compatibility reasons the standard classes provide definitions for these commands that emulate their behavior in L<sup>A</sup>T<sub>E</sub>X 2.09. However, it is legitimate for you to redefine them in a package or in the preamble according to your personal taste, something you should not do with basic font selection commands like `\bfseries`.

Because the old L<sup>A</sup>T<sub>E</sub>X 2.09 font commands are now allowed to be defined freely in a document class or by the user, they are no longer used within the code for L<sup>A</sup>T<sub>E</sub>X<sub>2ε</sub>. Instead, all internal references to fonts are created using either high- or low-level interfaces of L<sup>A</sup>T<sub>E</sub>X's font selection scheme. This convention should be followed by package and class developers to ensure a consistent behavior throughout.

 *Do not use \bf  
and friends*

## 7.4 Using fonts in math

Unlike the situation in text, automatic changes in font shapes are generally not desired in math formulas. For mathematicians, individual shapes convey specific

information. For example, bold upright letters may represent vectors. If the characters in a formula were to change because of surrounding conditions, the result would be incorrect. For this reason handling of fonts in mathematical formulas is different than that in text.

Characters in a formula can be loosely put into two classes: symbols and alphabet characters (including digits). Internally, (L<sup>A</sup>)<sub>E</sub>X distinguishes between eight types of math characters (to account for appropriate spacing), but for the discussion of fonts the division into two classes is generally adequate.

Some symbols, such as  $=$ , can be entered directly from the keyboard. The bulk of them, however, must be entered via a control sequence—for example,  $\leq$  stands for  $\leq$ . The other main group of characters in a formula, the alphabet characters, are entered directly from the keyboard.

More than 200 symbols are predefined in a standard (L<sup>A</sup>)<sub>E</sub>X system, allowing the user to typeset almost any desired formula. These symbols are scattered over several fonts, but they are accessed in such a way that the user does not have to be aware of their internal representations. If necessary, additional symbol fonts can be made accessible in a similar way; see Section 7.10.7.

The most important difference between symbols and alphabet characters is that symbols always have the same graphical representation within one formula, while it is possible for the user to change the appearance of the alphabet characters. We will call the commands that change the appearance of alphabet characters in a formula “math alphabet identifiers” and the fonts associated with these commands “math alphabets”. The alphabet identifiers are independent of surrounding font commands outside the formula, so a formula does not change if it is placed (for example) inside a theorem environment whose text is, by default, typeset in italics. This behavior is very important, because character shapes in a mathematical formula carry meanings that must not change because the formula is typeset in a different place in a document.

Some people who are familiar with the old method of font selection may be surprised by the fact that commands like  $\bfseries$  cannot be used in formulas. This is the price we must pay for the greater flexibility in choosing text font attributes—a flexibility that we do not want in a formula. We therefore need a different mechanism (math alphabet identifiers) for changing the typeface of certain alphabet characters in complicated formulas.

#### 7.4.1 Special math alphabet identifiers

One alphabet and a huge number of symbols are not sufficient for scientists to express their thoughts. They tend to use every available typeface to denote special concepts. Besides the use of foreign alphabets such as Greek letters, which usually are accessed as symbols— $\alpha$ ,  $\beta$ , and so on—we find sans serif letters for matrices, bold serif letters for vectors, and Fraktur fonts for groups, ideals, or fields. Others use calligraphic shapes to denote sets. The conventions are endless, and—even more importantly—they differ from one discipline to another.

<i>Command</i>	<i>Example</i>
<code>\mathcal</code>	<code>\$\mathcal{A}=a\$</code> $\mathcal{A} = a$
<code>\mathrm</code>	<code>\$\mathrm{max}_i\$</code> $\mathrm{max}_i$
<code>\mathbf</code>	<code>\$\sum x = \mathbf{v}\$</code> $\sum x = \mathbf{v}$
<code>\mathsf</code>	<code>\$\mathsf{G}_1^2\$</code> $\mathsf{G}_1^2$
<code>\mathtt</code>	<code>\$\mathtt{W}(a)\$</code> $W(a)$
<code>\mathnormal</code>	<code>\$\mathnormal{abc}=abc\$</code> $abc = abc$
<code>\mathit</code>	<code>\$differ\noteq\mathit{differ}\$</code> $differ \neq differ$

Table 7.4: Predefined math alphabet identifiers in L<sup>A</sup>T<sub>E</sub>X

For this reason L<sup>A</sup>T<sub>E</sub>X makes it possible to declare new math alphabet identifiers and associate them with any desired font shape group instead of relying only on a predefined set that cannot be extended. These identifiers are special commands for use in a formula that typeset any alphabet character in their argument in a specific typeface. (Symbols cannot be changed in this way.) These identifiers may use different typefaces in different formulas, as we will see in Section 7.4.3, but within one formula they always select the same typeface regardless of the surrounding conditions.

### Predefined alphabet identifiers

New math alphabet identifiers can be defined according to the user's needs, but L<sup>A</sup>T<sub>E</sub>X already has a few built in. These identifiers are shown in Table 7.4. As the last lines in the table show, the letters used in formulas are taken by default from the math alphabet `\mathnormal`. In contrast, the letters produced by `\mathit` have different spacing; thus this alphabet could be used to provide full-word variable names, which are common in some disciplines.

In L<sup>A</sup>T<sub>E</sub>X 2<sub>E</sub> math alphabet identifiers are commands with one argument, usually a single letter or a single word to be typeset in a special font.

Therefore, G can be computed as

$$G = A + \sum_{i=1}^n B_i \quad (1)$$

7-4-1

Therefore, `$\mathsf{G}$` can be computed as

```
\begin{equation}
\mathsf{G} = \mathcal{A} + 
\sum_{i=1}^n \mathcal{B}_i
\end{equation}
```

This procedure differs from the way font commands were used in L<sup>A</sup>T<sub>E</sub>X 2.09, where commands, such as `\rm`, would cause font changes ( $\dots\{\rm A\}\dots$ ). For the most important two-letter font-changing commands like `\rm`, `\sf`, `\bf`, `\it`, and `\tt`, the old syntax is still supported in the standard classes. For the others you can force the old behavior by specifying the package `oldlfont`; see Section 7.12.1. However, we suggest that you refrain from using such commands in new L<sup>A</sup>T<sub>E</sub>X documents.

As already mentioned, another difference between the old L<sup>A</sup>T<sub>E</sub>X 2.09 font selection scheme and NFSS is that text font declarations are no longer allowed in formulas, as they merely change some characteristic of the current font rather than switching to a specific font. Thus, if you write `\bfseries...` instead of `\mathbf{...}` in a formula, L<sup>A</sup>T<sub>E</sub>X will produce an error message.

The command names for the math alphabet identifiers are chosen to be descriptive rather than simple to type—they all start with `\math`. Therefore, if you use the commands more than occasionally in your document, you should consider defining some abbreviations in the preamble, such as the following:

```
\newcommand\mrm{\mathrm}
```

*No default math alphabet* You may wonder what the default math alphabet is—that is, from which alphabet the alphabet characters are selected if you do not specify an alphabet identifier explicitly, as in the formula `$x = 123$`. The answer is that no single default math alphabet exists. The (I)T<sub>E</sub>X system can be set up so that alphabetical characters are fetched from different alphabets as long as the user has not explicitly asked for a specific one, and this is normally the case, as the following example shows.

	<code>\begin{eqnarray}</code>	
<code>x == 12345</code>	(1)	<code>x &amp;= 12345 \\</code>
<code>x == 12345</code>	(2)	<code>\mathrm{x} &amp;= \mathrm{12345} \\</code>
<code>x == 12345</code>	(3)	<code>\mathnormal{x} &amp;= \mathnormal{12345}</code>
		<code>\end{eqnarray}</code>

7-4-2

As you can see, `\mathrm` does not change the digits and `\mathnormal` does not change the letters, so the default for digits in the normal set-up is the math alphabet associated with `\mathrm` and the default for letters is the one associated with `\mathnormal`.<sup>1</sup> This behavior can be controlled with the `\DeclareMathSymbol` command, which is explained in Section 7.10.7.

### Defining new alphabet identifiers

New math alphabet identifiers are defined with the `\DeclareMathAlphabet` command. Suppose that you want to make a slanted sans serif typeface available as a math alphabet. First you decide on a new command name, such as `\msfs`, to be used to select your math alphabet. Then you consult the font classification tables in this chapter (starting on page 354) to find a suitable font shape group to assign to this alphabet identifier. You will find that the Computer Modern Sans family, for example, consists of a medium series with upright and slanted shapes. If you decide to use the slanted shape of this family, you tell L<sup>A</sup>T<sub>E</sub>X using `\DeclareMathAlphabet`.

<sup>1</sup>It is a strange fact that the math font that corresponds to the `\mathnormal` alphabet actually contains old-style numerals. When the Computer Modern fonts were developed, space was a rare commodity, so Donald Knuth squeezed a number of “nonmathematical” glyphs into these fonts that are normally used only in text.

```
\DeclareMathAlphabet{cmd}{encoding}{family}{series}{shape}
```

This declaration has four arguments besides the identifier: the encoding scheme, the family, the series, and the shape of the font to be used. The alphabet identifier defined in the example will always switch to Computer Modern Sans medium slanted.

We demonstrate this with the formula

$$\sum A_i = a \tan \beta \quad (1)$$

```
\DeclareMathAlphabet{\msfsl}{OT1}{cmss}{m}{sl}
We demonstrate this with the formula
\begin{equation}
\sum \msfsl{A}_i = a \tan \beta
\end{equation}
```

It is also possible to redefine an existing math alphabet identifier in a package file or in the preamble of your document. For example, the declaration

```
\DeclareMathAlphabet{\mathsf}{OT1}{pag}{m}{n}
```

will override the default settings for the `\mathsf` alphabet identifier. After that, `\mathsf` will switch to Adobe Avant Garde in your formulas. There is, however, a subtle point: if the math alphabet in question is part of a symbol font that is already loaded by L<sup>A</sup>T<sub>E</sub>X for other reasons (e.g., `\mathcal`), it is better to use `\DeclareSymbolFontAlphabet` as it makes better use of T<sub>E</sub>X's somewhat limited resources for math; see page 435 for details.

### 7.4.2 Text font commands in math

As mentioned previously, text font declarations like `\rmfamily` cannot be used in math. However, the font-changing commands with arguments—for example, `\textrm`—can be used in both text and math. You can use these commands to temporarily exit the math context and typeset some text in the midst of your formula that logically belongs to the text surrounding the formula. Note that the font used to typeset this text will depend on surrounding conditions—that is, it will pick up the current values of encoding, family, series, and shape, as in the next example.

The result will be

$$x = 10 \textbf{ and thus } y = 12 \quad \sffamily \text{The result will be} \\ \text{[ x = 10 \textbf{ and thus } } y = 12 \text{ ]}$$

As you see, the Sans family was retained and the series was changed to bold. Perhaps more useful is the `\text` command, provided by the `amstext` package, which picks up the current values of encoding, family, series, and shape without changing any of them (see Section 8.6.1).

### 7.4.3 Mathematical formula versions

Besides allowing parts of a formula to be changed by using math alphabet identifiers, L<sup>A</sup>T<sub>E</sub>X lets you change the appearance of a formula as a whole. Formulas are typeset in a certain “math version”, and you can switch between math versions outside of math mode by using the command `\mathversion`, thereby changing the overall layout of the following formulas.

L<sup>A</sup>T<sub>E</sub>X knows about two math versions called “normal” and “bold”. Additional ones are sometimes provided in special packages. For example, the `mathtime` package (for the commercial MathTime fonts) sets up a math version called “heavy” to typeset formulas with ultra bold symbols as provided by the MathTime fonts.

As the name indicates, `\mathversion{normal}` is the default. In contrast, the `bold` version will produce bolder alphabet characters and symbols, though by default big operators, like `\sum`, are not changed. The following example shows the same formula first in the normal and then in the bold math version.<sup>1</sup>

$$\sum_{j=1}^z j = \frac{z(z+1)}{2} \quad (1)$$

$$\sum_{j=1}^z j = \frac{z(z+1)}{2} \quad (2)$$

```
\begin{equation}
\sum_{j=1}^z j = \frac{z(z+1)}{2}
\end{equation}
\mathversion{bold}
\begin{equation}
\sum_{j=1}^z j = \frac{z(z+1)}{2}
\end{equation}
```

7-4-5

Using `\mathversion` might be suitable in certain situations, such as in headings, but remember that changing the version means changing the appearance (and perhaps the meaning) of the entire formula. If you want to darken only some symbols or characters within one formula, you should not change the `\mathversion`. Instead, you should use the `\mathbf` alphabet identifier for characters and/or use the command `\bm` provided by the `bm` package; see Section 8.8.2.

If you change the math version with the `\mathversion` command, L<sup>A</sup>T<sub>E</sub>X looks in its internal tables to find where all the symbols for this new math version are located. It also may change all or some of the math alphabet identifiers and associate them with other font shapes in this version.

But what happens to math alphabet identifiers that you have defined yourself, such as the `\msfs1` from Example 7-4-3? As long as you declared them using only `\DeclareMathAlphabet`, they will stay the same in all math versions.

If the math alphabet identifier is to produce a different font in a special math version, you must inform L<sup>A</sup>T<sub>E</sub>X of that fact by using the `\SetMathAlphabet` command. For example, in the default set-up the `\mathsf` alphabet identifier is defined as follows:

```
\DeclareMathAlphabet{\mathsf}{OT1}{cmss}{m}{n}
\SetMathAlphabet{\mathsf}{bold}{OT1}{cmss}{bx}{n}
```

<sup>1</sup>For historical reasons L<sup>A</sup>T<sub>E</sub>X has two additional commands to switch to its standard math versions: `\boldmath` and `\unboldmath`.

The first line means that the default for `\mathsf` in all math versions is Computer Modern Sans medium. The second line states that the bold math version should use the font Computer Modern Sans bold extended instead.

```
\SetMathAlphabet{cmd}{version}{encoding}{family}{series}{shape}
```

From the previous example, you can see that `\SetMathAlphabet` takes six arguments: the first is the name of the math alphabet identifier, the second is the math version name for which you are defining a special set-up, and the other four are the encoding, family, series, and shape name with which you are associating it.

As noted earlier, you can redefine an existing math alphabet identifier by using `\DeclareMathAlphabet`. If you do so, all previous `\SetMathAlphabet` declarations for this identifier are removed from the internal tables of L<sup>A</sup>T<sub>E</sub>X. Thus, the identifier will come out the same in all math versions unless you add new `\SetMathAlphabet` declarations for it.

## 7.5 Standard L<sup>A</sup>T<sub>E</sub>X font support

This section opens with a short introduction to the standard text fonts distributed together with L<sup>A</sup>T<sub>E</sub>X: Computer Modern and European Computer Modern. It is followed by a discussion of L<sup>A</sup>T<sub>E</sub>X's standard support packages for input and font encodings. The section concludes by describing a package for tracing L<sup>A</sup>T<sub>E</sub>X's font processing and another package for displaying glyph charts (a package the author used extensively while preparing the later parts of this chapter).

### 7.5.1 Computer Modern—The L<sup>A</sup>T<sub>E</sub>X standard fonts

Along with T<sub>E</sub>X, Donald Knuth developed a family of fonts called Computer Modern; see Table 7.5 on the next page. Until the early 1990s, essentially only these fonts were usable with T<sub>E</sub>X and, consequently, with L<sup>A</sup>T<sub>E</sub>X. Each of these text fonts contains 128 glyphs (T<sub>E</sub>X was working with 7 bits originally), which does not leave room for including accented characters as individual glyphs. Thus, using these fonts means that accented characters have to be produced with the `\accent` primitive of T<sub>E</sub>X, which in turn means that automatic hyphenation of words with accented characters is impossible. While this restriction is acceptable with English documents that contain few foreign words, it is a major obstacle for other languages.

Not surprisingly, these deficiencies were of great concern to the T<sub>E</sub>X users in Europe and eventually led to a reimplementation of T<sub>E</sub>X in 1989 to support 8-bit characters internally and externally. At the T<sub>E</sub>X Users conference in Cork (1990), a standard 8-bit encoding for text fonts (T1) was developed that contains many diacritical characters (see Table 7.32 on page 449) and allows typesetting in more

*Original T<sub>E</sub>X font  
encoding*

*T1 a.k.a. "Cork"  
encoding*

Family	Series	Shape(s)	Example of Typeface
<i>Computer Modern Roman (T1, OT1, TS1)</i>			
cmr	m	n, it, sl, sc, ui	COMPUTER ROMAN SMALL CAPS
cmr	bx	n, it, sl	<i>Comp. Mod. Roman bold extended italic</i>
cmr	b	n	Computer Modern Roman bold upright
<i>Computer Modern Sans (T1, OT1, TS1)</i>			
cmss	m	n, sl	<i>Computer Modern Sans slanted</i>
cmss	bx	n	<b>Computer Modern Sans bold extended</b>
cmss	sbc	n	Computer Modern Sans semibold condensed
<i>Computer Modern Typewriter (T1, OT1, TS1)</i>			
cmtt	m	n, it, sl, sc	<i>Computer Modern Typewriter italic</i>
cmvtt	m	n, it	Proportional Computer Modern Typewriter
<i>Computer Modern Fibonacci (T1, OT1)</i>			
cmfib	m	n	Computer Modern Fibonacci
<i>Computer Modern Funny Roman (T1, OT1)</i>			
cmfr	m	n, it	Computer Modern Funny Roman
<i>Computer Modern Dunhill (T1, OT1)</i>			
cmdh	m	n	Computer Modern Dunhill

Table 7.5: Classification of the Computer Modern font families

than 30 languages based on the Latin alphabet. At the University of Bochum (under the direction of Norbert Schwarz) the Computer Modern font families were then reimplemented, and additional characters were designed, so that the resulting fonts completely conform to this encoding scheme. The first implementation of these fonts was released under the name “DC fonts”. Since then Jörg Knappen has finalized them and they are now distributed as “European Computer Modern Fonts”, often shortened to “EC fonts”.<sup>1</sup>

*EC fonts*

Both Computer Modern and the EC fonts are considered standard in L<sup>A</sup>T<sub>E</sub>X and must be available at any installation. Although originally developed with METAFONT, there are now free Type 1 PostScript replacements as well. For Computer Modern these were produced by Blue Sky Research; Y&Y added the L<sup>A</sup>T<sub>E</sub>X, AMS, and Euler fonts. The EC fonts have been recently converted from METAFONT sources

*PostScript Type 1 instances*

<sup>1</sup>Not to be confused with the European Modern Fonts™, a high-quality set of commercial fonts by Y&Y that are based on the Computer Modern design but have slightly different metrics [65].

to Type 1 PostScript by Vladimir Volovich. His implementation is called the CM-Super fonts package and, beside the EC fonts, it covers EC Concrete, EC Bright, *CM-Super fonts* and LH fonts (Cyrillic Computer Modern). In addition to the T1 encoding, the L<sup>A</sup>T<sub>E</sub>X standard encodings TS1, T2A, T2B, T2C, and X2 are supported by CM-Super. The CM-Super fonts have been automatically converted to the Type 1 format and although a sophisticated algorithm was used for this conversion, you cannot expect exactly the same quality as could be achieved by a manual conversion process.

Since the PostScript fonts have the same font metrics as their METAFONT counterparts they need no support package in the L<sup>A</sup>T<sub>E</sub>X document. Once installed they will be automatically used by the driver program (e.g., dvips) that converts the .dvi output to PostScript. The standard .fd files for Computer Modern provide only well-defined font sizes to avoid the generation of too many bit-mapped fonts. However, with PostScript the use of intermediate sizes (via \fontsize) is possible without any such side effect. The package fix-cm makes use of this feature.

Although the EC fonts were originally meant to be a drop-in extension (and replacement) for the 7-bit Computer Modern fonts, not all glyph shapes were kept in the end. For example, the German ß got a new design—a decision by the font designer that did not make everybody happy.

7-5-1

Computer Modern sharp s: ß	\fontencoding{OT1}\fontfamily{cmr}\selectfont Computer Modern sharp s: \ss \par
EC Modern sharp s: ß	\fontencoding{T1}\fontfamily{cmr}\selectfont EC Modern sharp s: \ss

With the CM-Super fonts this is no longer a problem: if one prefers the original CM glyph over the EC glyph, one can simply exchange `germandbls` with `germandbls.alt` in the file `cm-super-t1.enc`.<sup>1</sup>

However, these are not the only differences between the original Computer Modern fonts and the new EC fonts. The latter have many more individual designs for larger font sizes (while CM fonts were scaled linearly) and in this respect the fact that both really are different font families is quite noticeable.<sup>2</sup> The particular example that follows is perhaps the most glaring difference of that kind.

7-5-2

**The fox jumps**  
quickly over the fence!

**The fox jumps**  
quickly over the fence!

```
\fontencoding{OT1}\sffamily\bfseries
\Huge      The fox jumps \par
\normalsize quickly over the fence!\par
\fontencoding{T1}\sffamily\bfseries
\Huge      The fox jumps \par
\normalsize quickly over the fence!\par
```

<sup>1</sup>An even better solution is to use a different name for the modified encoding file and then change the references in the (dvips) mapping file to use the new name.

<sup>2</sup>The historical mistake was to pretend to NFSS that both are the same families (e.g., `cmr`, `cmss`), just encoded according to different font encodings. Unfortunately, this cannot be rectified without huge backward compatibility problems.

This issue is no problem if one likes the EC designs and uses T1 throughout. Otherwise, a number of approaches can be taken to resolve this problem. One is to employ a different set of font definitions that do not make use of *all* individual EC font designs, and that are closer to those of the traditional CM fonts, but with improved typographical quality. Such a solution is provided by Walter Schmidt's package `fix-cm`, which is distributed as part of the core L<sup>A</sup>T<sub>E</sub>X distribution. Load this package directly after the document class declaration (or even before using `\RequirePackage`), as it takes effect only for fonts not already loaded by L<sup>A</sup>T<sub>E</sub>X—and the document class might load fonts.

## The fox jumps

quickly over the fence!

## The fox jumps

quickly over the fence!

```
\usepackage{fix-cm}
\fontencoding{OT1}\sffamily\bfseries
\Huge The fox jumps \par
\normalsize quickly over the fence!\par
\fontencoding{T1}\sffamily\bfseries
\Huge The fox jumps \par
\normalsize quickly over the fence!
```

7-5-3

Another possible solution is to use the Almost European fonts (by Lars Engebretsen) or the EZ-fonts (by Robert Fuster), both of which are sets of virtual fonts built upon the Computer Modern fonts. They implement the T1 encoding with the exception of a small number of glyphs that simply cannot be obtained from the CM font material.

This approach has a number of disadvantages. For instance, these solutions do not support the companion symbol fonts, so the additional symbols provided by the `textcomp` package cannot be used at all. More importantly, the use of virtual fonts to build composite glyphs means that a resulting .pdf file would not be searchable for words containing diacritics, simply because instead of the accented character (as a single glyph) a complicated construction is placed in this file. In other words, the solutions help to make L<sup>A</sup>T<sub>E</sub>X believe that it deals with single glyphs (and thus allows proper hyphenation and kerning) but this information is lost again in the resulting output file, so further post-processing cannot be done properly.

However, as far as the selected fonts are concerned, the `ae` package shows the same result as `fix-cm`.

## The fox jumps

quickly over the fence!

```
\usepackage{ae}
\fontencoding{T1}\sffamily\bfseries
\Huge The fox jumps \par
\normalsize quickly over the fence!
```

7-5-4

Searching  
problems in .pdf  
documents

*Latin Modern on the  
horizon*

In 2002, three European T<sub>E</sub>X user groups (DANTE, GUTenberg, and NTG) initiated and funded a project to integrate all of the variants of the Computer Modern Roman typefaces into a single Latin Modern family of fonts. The project is being carried out by Bogusław Jackowski and Janusz Nowacki, and the first official

version of the Latin Modern fonts was presented at the DANTE meeting in 2003.

The **Latin Modern** fonts are carefully handcrafted PostScript Type 1 fonts based on the designs of Knuth's *Computer Modern* families. They contain all the glyphs needed to typeset Latin-based European languages. At the moment the T1 and TS1 encodings are supported. In a later step the project will address glyphs needed for typesetting Native American, Vietnamese, and Transliteration. Also planned are 8-bit math encodings (based on earlier work by CLASEN/VIETH and ZIEGLER [40,174]).

7-5-5

```
\usepackage{lmodern} \usepackage[T1]{fontenc}
The \textbf{Latin Modern} fonts are carefully handcrafted PostScript Type-1 fonts based on the designs of Knuth's \emph{Computer Modern} families. They contain all the glyphs needed to typeset Latin-based European languages. At the moment the \texttt{T1} and \texttt{TS1} encodings are supported. In a later step the project will address glyphs needed for typesetting Native American, Vietnamese, and Transliteration. Also planned are 8-bit math encodings (based on earlier work by \textsc{Clasen/Vieth} and \textsc{Ziegler}~[40,174]).
```

At the time of writing, the fonts were continuing to undergo further fine-tuning. For example, additional kerning pairs and language-dependent ligatures are being added. It is expected that a later version of the Latin Modern fonts will become the default fonts for L<sup>A</sup>T<sub>E</sub>X; for now, they can be used by loading the lmodern package and selecting the T1 encoding.

### 7.5.2 inputenc—Selecting the input encoding

If your computer allows you to write accented characters, either via single keystrokes or by some other input method (e.g., by pressing ‘ and then a to get à) and also displays them nicely in the editor...

Quand ils furent revenus un peu à eux, ils marchèrent vers  
Lisbonne ; il leur restait quelque argent, avec lequel ils  
espéraient se sauver de la faim après avoir échappé à la  
tempête      (Voltaire)

...then ideally you would use such a text directly with L<sup>A</sup>T<sub>E</sub>X instead of having to type ‘a, ^e, and so forth.

While with languages such as French and German the latter approach is still feasible, languages such as Russian and Greek really require the potential for direct input, as (nearly) every character in these languages has a command name as its internal L<sup>A</sup>T<sub>E</sub>X form. For example, the default Russian definition for \ref{textafter} contains the following text (meaning “on the next page”):

```
\cyrn\cyra\ \crys\cyl\cyre\cyrd\cyru\cyryu\cyrshch\cyre\cyrishrt
\ \crys\cyrt\cyrr\cyra\cyrn\cyri\cyrc\cyre
```

Clearly, no one wants to type text like this on a regular basis. Nevertheless, it has the advantage of being universally portable, meaning that it will be interpreted

correctly on any  $\text{\LaTeX}$  installation. On the other hand, typing on an appropriate keyboard

на следующей странице

is clearly preferable, provided it is possible to make  $\text{\LaTeX}$  understand this kind of input. The problem is that what is stored in a file on a computer is not the characters we see in the above sequence, but rather octets (numbers) representing the characters. In different circumstances (using a different encoding), the same octets might represent different characters.

How does  $\text{\LaTeX}$  determine which interpretation it should use? As long as everything happens on a single computer and all programs interpret octets in files (when reading or writing) in the same manner, everything is usually fine. In such a situation it may make sense to activate an automatic translation mechanism that is built into several recent  $\text{\TeX}$  implementations. If, however, any file produced on such a system is sent to a different computer, processing is likely to fail or, even worse, may appear to succeed, but will in fact produce wrong results by displaying incorrect characters.

To cope with this situation the `inputenc` package was created. Its main purpose is to tell  $\text{\LaTeX}$  the “encoding” used in the document or in a part of the document. This is done by loading the package with the encoding name as an option. For example:

```
\usepackage[cp1252]{inputenc} % Windows 1252 (Western Europe) code page
```

From that point onward  $\text{\LaTeX}$  knows how to interpret the octets in the remainder of the document on any installation,<sup>1</sup> regardless of the encoding used for other purposes on that computer.

A typical example is shown below. It is a short text written in the koi8-r encoding popular in Russia. The right side shows what the text looks like on a computer using a Latin 1 encoding (e.g., in Germany). The left side shows that  $\text{\LaTeX}$  was nevertheless able to interpret the text correctly because it was told which input encoding was being used.

Русский язык (The Russian language)

```
\usepackage[russian]{babel}
\usepackage[koi8-r]{inputenc}
ôóóóééé ÑÚÜÈ (The Russian language)
```

7-5-6

The list of encodings currently supported by `inputenc` is given below. The interface is well documented, and support for new encodings can be added easily. Thus, if the encoding used by your computer is not listed here, it is worth looking

<sup>1</sup>This statement is true only if the  $\text{\TeX}$  installation has not been set up to make some hard-wired transformation when reading from a file. As mentioned in the introduction to this chapter, many  $\text{\TeX}$  implementations have been extended to support such transformations, but if they are activated it is no longer possible to process documents in several languages in parallel.

into the `inputenc` package documentation<sup>1</sup> to see whether it was added recently. You can also search the Internet for encoding files for `inputenc` provided by other authors. For example, encodings related to the Cyrillic languages are distributed together with other font support packages for Cyrillic languages.

The ISO 8859 standard [67] defines a number of important single-byte encodings, of which those related to the Latin alphabet are supported by `inputenc`. For MS-DOS and Windows operating systems a number of single-byte encodings have been defined by IBM and Microsoft, of which a subset is currently supported. In addition, some encodings defined by other computer vendors are available. The perhaps somewhat ad hoc (and constantly growing) selection is mainly the result of contributions from the L<sup>A</sup>T<sub>E</sub>X user community.

`latin1` This is the ISO 8859-1 encoding (also known as Latin 1). It can represent most Western European languages, including Albanian, Catalan, Danish, Dutch, English, Faroese, Finnish, French, Galician, German, Icelandic, Irish, Italian, Norwegian, Portuguese, Spanish, and Swedish.

`latin2` The ISO Latin 2 encoding (ISO 8859-2) supports the Slavic languages of Central Europe that use the Latin alphabet. It can be used for the following languages: Croatian, Czech, German, Hungarian, Polish, Romanian, Slovak, and Slovenian.

`latin3` This character set (ISO 8859-3) is used for Esperanto, Galician, Maltese, and Turkish.

`latin4` The ISO Latin 4 encoding (ISO 8859-4) can represent languages such as Estonian, Latvian, and Lithuanian.

`latin5` The ISO Latin 5 encoding (ISO 8859-9) is closely related to Latin 1 and replaces the rarely used Icelandic letters from Latin 1 with Turkish letters.

`latin9` Latin 9 (or ISO 8859-15) is another small variation on Latin 1 that adds the euro currency sign as well as a few other characters, such as the œ ligature, that were missing for French and Finnish. It is becoming increasingly popular as a replacement for Latin 1.

`cp437` IBM 437 code page (MS-DOS Latin but containing many graphical characters to draw boxes).

`cp437de` IBM 437 code page but with a “ß” (German sharp s) in place of a  $\beta$  (Greek beta) as used with German keyboards.

`cp850` IBM 850 code page (MS-DOS multilingual  $\approx$  `latin1`).

`cp852` IBM 852 code page (MS-DOS multilingual  $\approx$  `latin2`).

`cp858` IBM 858 code page (IBM 850 with the euro symbol added).

`cp865` IBM 865 code page (MS-DOS Norway).

<sup>1</sup>Process `inputenc.dtx` with L<sup>A</sup>T<sub>E</sub>X.

`cp1250` Windows 1250 (Central and Eastern Europe) code page.  
`cp1252` Windows 1252 (Western Europe) code page.  
`cp1257` Windows 1257 (Baltic) code page.  
`ansinew` Windows 3.1 ANSI encoding; a synonym for `cp1252`.  
`decmulti` DEC Multinational Character Set encoding.  
`applemac` Macintosh (standard) encoding.  
`macce` Macintosh Central European code page.  
`next` Next Computer encoding.  
`utf8` Unicode's UTF-8 encoding support.

Most  $\text{\TeX}$  installations accept 8-bit characters by default. Nevertheless, without further adjustments, like those performed by `inputenc`, the results can be unpredictable: characters may vanish, or you might get whatever character is present in the current font at the octet location being referred to, which may or may not be the desired glyph. This behavior was the default for a long time, so it was not changed in  $\text{\LaTeX} 2\epsilon$  because some people rely on it. However, to ensure that such mistakes can be caught, `inputenc` offers the option `ascii`, which makes any character outside the range 32–126 illegal.

### `\inputencoding{encoding}`

Originally the `inputenc` package was written to describe the encoding used for a document as a whole—hence the use of options in the preamble. It is, however, possible to change the encoding in the middle of a document by using the command `\inputencoding`. This command takes the name of an encoding as its argument. Processing is rather computing intensive, as typically more than 120 characters are remapped each time. Nevertheless, we know of applications that change the encoding several times within a paragraph yet seem to work reasonably well.

When `inputenc` was written, most  $\text{\LaTeX}$  installations were on computers that *UTF-8 support* used single-byte encodings like the ones discussed in this section. Today, however, another encoding is becoming popular as systems start to provide support for Unicode: UTF-8. This variable-length encoding represents Unicode characters in one to four octets. Recently, some Linux distributions decided to use UTF-8 as the default encoding for the operating system, leaving their  $\text{\LaTeX}$  users baffled that files written using the keys on the keyboard were suddenly no longer accepted by  $\text{\LaTeX}$ . For this reason encoding support for UTF-8 was added to `inputenc` via the option `utf8`. Technically, it does not provide a full UTF-8 implementation. Only Unicode characters that have some representation in standard  $\text{\LaTeX}$  fonts are mapped (i.e.,

mainly Latin and Cyrillic character sets); all others will result in a suitable error message. In addition, Unicode combining characters are not supported, although that particular omission should not pose a problem in practice.

7-5-7

```
\usepackage[utf8]{inputenc}
\usepackage{textcomp} % for Latin interpretation
German umlauts in UTF-8: äöü
But interpreted as Latin 1: ÄÖÄ¶Ä¼
\par\inputencoding{latin1}% switch to Latin 1
But interpreted as Latin 1: ^c3^a4^c3^b6^c3^bc
```

UTF-8 has the property that ASCII characters represent themselves and most Latin characters are represented by two bytes. In the verbatim text of the example, the two-byte representations of the German umlauts in UTF-8 are shown in T<sub>E</sub>X's hexadecimal notation, that is with each octet preceded by `^`. In an editor that does not understand UTF-8, one would probably see them as similar to the output that is produced when they are interpreted as Latin 1 characters.

The UTF-8 support offered by `inputenc` at the moment<sup>1</sup> is restricted to the character subset of Unicode directly supported by the `inputenc` mapping options (e.g., `latin1`, `latin2`) as described on page 359. A package with more comprehensive UTF-8 support (including support for Chinese, Korean, and Japanese characters), though consequently more complex in its set-up, is the `ucs` package written by Dominique Unruh. You may want to give it a try if the `inputenc` solution does not cover your needs.

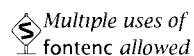
### 7.5.3 fontenc—Selecting font encodings

To be able to use a text font encoding with L<sup>A</sup>T<sub>E</sub>X, the encoding has to be loaded in the document class, a package, or in the document preamble. More precisely, the definitions to access the glyphs in fonts with a certain encoding have to be loaded. The canonical way to do this is via the `fontenc` package, which takes a comma-separated list of font encodings as a package option. The last of these encodings is automatically made the default document encoding. If Cyrillic encodings are loaded, the list of commands affected by `\MakeUppercase` and `\MakeLowercase` is automatically extended. For example,

```
\usepackage[T2A,T1]{fontenc}
```

will load all necessary definitions for the Cyrillic T2A and the T1 (Cork) encodings and set the latter to be the default document encoding.

In contrast to normal package behavior, one can load this package several times with different optional arguments to the `\usepackage` command. This is necessary to allow a document class to load a certain set of encodings and enable



<sup>1</sup>This is more of a resource problem than a technical one and thus may change.

the user to load still more encodings in the preamble. Loading encodings more than once is possible without side effects (other than potentially changing the document default font encoding).

If language support packages (e.g., those coming with the babel system) are used in the document, it is often the case that the necessary font encodings are already loaded by the support package.

#### 7.5.4 `textcomp`—Providing additional text symbols

When the T1 font encoding was defined in Cork, it was decided that this encoding should omit many standard text symbols such as † and instead include as many composite glyphs as possible. The rationale was that characters that are subject to hyphenation have to be present in the same font, while one can fetch other symbols without much penalty from additional fonts. These extra symbols have, therefore, been collected in a companion encoding.

In 1995, a first implementation of this encoding (TS1) was developed by Jörg Knappen [78, 79]. With the `textcomp` package, Sebastian Rahtz provided a L<sup>A</sup>T<sub>E</sub>X interface to it.

Unfortunately, just as with the T1 encoding, the encoding design for TS1 was prepared based on glyph availability in the T<sub>E</sub>X world without considering that the majority of commercial fonts provide different sets of glyphs. As a result, the full implementation of this encoding is available for very few font families, among them EC and CM Bright fonts. For most PostScript fonts implementations of the encoding also exist, but half of the glyphs are missing and produce square blobs of ink.<sup>1</sup> Table 7.6 on pages 363–364 shows the glyphs made available by `textcomp` and the commands to access them. Commands colored in blue indicate that the corresponding glyph is most likely not available when PostScript fonts are used.

To help with these problems the `textcomp` package nowadays knows for many subsets of the TS1 encoding what extent they implement the TS1 encoding. In addition, it offers a number of options that restrict the set of new commands for those font families it does not know about.

For any unknown font family, the option `safe` allows only commands available with the ISO-Adobe character set (except for `\textcurrency` but adding a fake `\texteuro`). The option `euro` replaces the fake euro symbol with a real glyph; hence if that glyph does not exist in the font, `\texteuro` will produce a nasty blob of ink.

The package option `full` enables all commands for fonts `textcomp` does not know about. This means in particular that the perfectly valid L<sup>A</sup>T<sub>E</sub>X commands `\textcircled` and `\t` will stop working the moment a document font is selected that does not contain the necessary glyphs in its TS1 encoding. For this reason,

<sup>1</sup>The T1 encoding has the same problem when it comes to PostScript fonts, but fortunately only five (seldom used) glyphs are missing from most fonts; see Example 7-9-2 on page 417.

<i>Accent symbols</i>		
Á \capitalacute{A}	Ä \capitalbreve{A}	Å \capitalcaron{A}
À \capitalcedilla{A}	Â \capitalcircumflex{A}	Ë \capitaldieresis{A}
À \capitaldotaccent{A}	À \capitalgrave{A}	Ã \capitalhungarumlaut{A}
À \capitalmacron{A}	Â \capitalnewtie{A}	Ü \capitalogonek{U}
À \capitalring{A}	Ö Ö \capitaltie{O}	Ã \capitaltilde{A}
ö \newtie{o}	Ⓐ Ⓛ \textcircled{A}	ő ö \t{oo}
<i>Numerals (superior, fractions, old style)</i>		
¹ \textonesuperior	² \texttwosuperior	³ \textthreesuperior
¼ \textonequarter	½ \textonehalf	¾ \textthreequarters
º \textzeroldstyle	¹ \textoneoldstyle	² \texttwooldstyle
³ \textthreeoldstyle	⁴ \textfouroldstyle	⁵ \textfiveoldstyle
⁶ \textsixoldstyle	⁷ \textsevenoldstyle	⁸ \texteightoldstyle
⁹ \textnineoldstyle		
<i>Pair symbols</i>		
{ \textlangle	} \textrangle	[ \textlbrackdbl
] \textrbrackdbl	↑ \textuparrow	↓ \textdownarrow
← \textleftarrow	→ \textrightarrow	{ \textlquill
}		
<i>Monetary and commercial symbols</i>		
฿ \textbaht	₵ \textcent	€ \textcentoldstyle
₵ \textcolonmonetary	₹ \textcurrency	\$ \textdollar
฿ \textdollaroldstyle	đ \textdong	€ \texteuro
ƒ \textflorin	₲ \textguarani	£ \textlira
₦ \textnaira	P \textpeso	£ \textsterling
₩ \textwon	¥ \textyen	
₱ \textcircled{P}	© \textcopyright	
% \textdiscount	℮ \textestimated	‰‰ \textpertenthousand
‰‰ \textperthousand	⌘ \textreferencemark	® \textregistered
℠ \textservicemark	™ \texttrademark	
<i>Footnote symbols</i>		
* \textasteriskcentered	\textbardbl	\textbrokenbar
▪ \textbullet	† \textdagger	‡ \textdaggerdbl
◦ \textopenbullet	¶ \textparagraph	· \textperiodcentered
¶ \textpilcrow	§ \textsection	
<i>Scientific symbols</i>		
°C \textcelsius	° \textdegree	÷ \textdiv
¬ \textlnot	℧ \textmho	— \textminus
μ \textmu	Ω \textohm	ª \textordfeminine
º \textordmasculine	± \textpm	√ \textsurd
× \texttimes		

*Blue* indicates symbols unavailable in most PostScript fonts.

Table 7.6: Commands made available with `textcomp`

<i>Various</i>		
" \textacutedbl	' \textasciacute	~ \textasciibreve
ˇ \textasciicaron	ˇ \textasciidieresis	ˋ \textasciigrave
– \textasciimacron	○ \textbigcircle	ˇ \textblank
★ \textborn	= \textdblhyphen	= \textdblyphenchar
† \textdied	◊ \textdivorced	/ \textfractionsolidus
“ \textgravedbl	‽ \textinterrobang	‽ \textinterrobangdown
⌚ \textleaf	♾ \textmarried	♪ \textmusicalnote
№ \textnumero	‘ \textquotesingle	, \textquotestraightbase
„ \textquotestraightdblbase	R \textrecipe	— \textthreequartersemdash
˜ \texttildelow	— \texttwelveudash	

*Blue* indicates symbols unavailable in most PostScript fonts.

Table 7.6: Commands made available with `textcomp` (cont.)

the default option `almostfull` leaves these two commands untouched, to avoid the situation shown in the next example.

CM fonts: (x) oo	CM fonts: \textcircled{x}\quad \t oo \par	Times fonts:
Times fonts: ■ □	\fontfamily{ptm}\selectfont\textcircled{x}\quad \t oo \par	7-5-8

Since Times Roman is a font that `textcomp` knows about, specifying `full` will still produce correct output; to get the ink blobs we also had to add `force` in the previous example. This option directs `textcomp` to ignore all knowledge about individual font families and use the subset denoted by the additional option in all cases.<sup>1</sup>

When `textcomp` gets loaded (with or without restricting options), a large number of new commands are made available to access the new symbols. In addition, a number of symbols that have been (historically) taken by L<sup>A</sup>T<sub>E</sub>X from math fonts (e.g., `\textbullet`, or `\textdagger`) are now taken from the companion fonts; as a consequence, they now sometimes change their shapes when the font attributes (family, series, shape) are changed.

\usepackage[safe]{textcomp}	7-5-9
\textdagger\textparagraph\textbullet{} viz.\	
\fontfamily{ptm}\selectfont\textdagger\textparagraph\textbullet	

While this is usually the right solution, it may result in changes in unexpected places. For example, the `itemize` environment by default uses `\textbullet` to indicate first-level items. If the slightly bigger bullet is preferred, then we have to

<sup>1</sup>This option is best avoided, as it can produce incorrect output without any warning.

undo the change in the default setting by returning the default to the right math encoding (usually OMS<sup>1</sup>). Compare this to Example 7-5-9.

7-5-10

<b>• now like •</b>	<pre>\usepackage[safe]{textcomp} \DeclareTextSymbolDefault{\textbullet}{OMS} \textbullet{} now like \fontfamily{ptm}\selectfont\textbullet</pre>
---------------------	--

Of course, a more sensible solution in this case may be to adjust the definition for `\labelitemi` (see Section 3.3.1). For example:

```
\renewcommand\labelitemi{\normalfont\UseTextSymbol{OMS}{\textbullet}}
```

Diacritical marks on uppercase letters are sometimes flattened in some font designs compared to their lowercase counterparts. The EC fonts follow this tradition. For example, the grave accents on ò and Ò are different (which is not the case with Lucida, the document font used in this book). This poses a problem if one needs an uncommon letter that is not available as a single glyph in the T1 encoding, but rather must be constructed by placing the diacritical mark over the base character. In that case the same diacritical mark is used, which can result in noticeable differences (see the X̄ in the next example). The `\capital... accents` shown in Table 7.6 on page 363 solve this problem by generating diacritical marks suitable for use with uppercase letters.

7-5-11

<b>ÒX ÒX ÒX</b>	<pre>\usepackage[T1]{fontenc} \usepackage[safe]{textcomp} \Huge \o'x \O'X \capitalgrave O\capitalgrave X</pre>
-----------------	--

L<sup>A</sup>T<sub>E</sub>X offers a `\textcompwordmark` command, an invisible zero-width glyph that can, for example, be used to break up unwanted ligatures (at the cost of preventing hyphenation). When the `textcomp` package is loaded, this glyph has a height of 1ex, which makes it possible to use it as the argument to an accent command, thereby placing an accent between two letters. In the next example this command is used to produce the German -burg abbreviation. With the `textcomp` package two additional compound word marks become available: `\textascendercompwordmark` and `\textcapitalcompwordmark` that have the height of the ascender or capitals in the font, respectively.

7-5-12

<b>b`g (this fails)</b> <b>bg B̄G</b>	<pre>\usepackage[T1]{fontenc} \usepackage[safe]{textcomp} b\uf{f}g (this fails) \par b\u\textcompwordmark g \quad B\u\textcapitalcompwordmark G</pre>
--	---

The above example works only with T1-encoded fonts (`textcomp` is additionally needed for the `\textcapitalcompwordmark`). The default definition for `\textcompwordmark` in L<sup>A</sup>T<sub>E</sub>X does not use a real zero-width character, but rather (lacking such a glyph) a zero-width space.

<sup>1</sup>One has to look for the default declaration in `latex.ltx` to find the right encoding.

As the \$ sign is a glyph available in both the OT1 and T1 encodings, there is no point in removing its definition and forcing L<sup>A</sup>T<sub>E</sub>X to pick up the TS1 version if you are typesetting in this encoding. However, assume you want to use the variant dollar sign \$, for your dollars automatically. In that case you have to get rid of the declarations in other encodings so that L<sup>A</sup>T<sub>E</sub>X will automatically switch to TS1.

```
\DeclareTextCommandDefault{\textdollar}
  {\UseTextSymbol{TS1}\textdollaroldstyle} % set up new default
\UndeclareTextCommand{\textdollar}{OT1}      % do not use the defs in
\UndeclareTextCommand{\textdollar}{T1}        % OT1 or T1
```

Such redeclarations will, of course, work only if the document fonts contain the desired glyph in the TS1 encoding. In this book they would have failed, because Lucida Bright (the document font for this book) has only the restricted set of ISO-Adobe symbols available. So if you wonder where the \$ and similar symbols shown in the book actually came from, the answer is simple: from the EC fonts.

What can you do if you want to use, say, \textborn, but the current font family you use does not implement it? One possible solution is to overwrite the default provided by the *textcomp* package using \DeclareTextCommandDefault. The idea is that the default switches to a font family that you know contains the desired symbol (for example, *cmr* if your main document font is a serifed font, or *cmss* if it is a sans serif one), and then you can use \UseTextSymbol to pick up the symbol from the TS1 encoding in that family.<sup>1</sup>

<pre>\usepackage[safe]{textcomp}</pre>	<pre>\DeclareTextCommandDefault{\textborn}</pre>
	<pre>{\fontfamily{cmr}\selectfont\UseTextSymbol{TS1}{\textborn}}</pre>
Burkhard and Holger	\fontfamily{ptm}\selectfont
*8.11.1997	Burkhard and Holger \textborn 8.11.1997

[7-5-13]

You can use this approach for any symbol defined by the *textcomp* package. In case of accents the definition is similar. This time we declare the default to have an argument and in the definition we use \UseTextAccent. For example:

```
\DeclareTextCommandDefault{\newtie}[1]
  {\fontfamily{cmr}\selectfont\UseTextAccent{TS1}{\newtie}{#1}}
```

In fact, for symbols (but not for accents), *textcomp* attempts to resolve the problem of missing glyphs by locally switching to a font family stored in \textcompsubstdefault (the default is Computer Modern Roman) and typesetting the symbol in this family, after having issued a suitable error message. Use the option *warn* to get only warnings instead of errors. Of course, such substitutions produce inferior results, especially for “textual symbols”, if the current font

<sup>1</sup> For more abstract symbols this approach often gives an acceptable result; in case of accents your mileage may vary.

is visually incompatible with the substitution family. In the next example we use Computer Modern Sans as a substitute. Be careful to select a family that has full TS1 coverage; otherwise, your redefinition will produce endless errors!

Helvetica with №, Ω, Ⓛ, ¶. Not perfect but better than nothing.

```
\usepackage[warn]{textcomp} \renewcommand\textcompsubstdefault{\rmss} \fontfamily{phv}\selectfont Helvetica with \textnumero, \textohm, \textcopyleft, \textpilcrow. Not perfect but better than nothing.
```

According to the specifications the TS1 encoding contains old-style digits as well as the punctuations period and comma. It allows one to typeset dates and other (positive) numbers with old-style numerals by simply switching to the TS1 font encoding. Unfortunately, old-style numerals are usually unavailable in most PostScript fonts (you must buy the “expert” font set in most cases), so that this method works correctly for only a few font families.<sup>1</sup>

The `textcomp` package solves this problem by redefining the `\oldstylenums` command to automatically use the old-style numerals in the TS1 encoding if the current font contains them. If not, it will issue a warning and produce lining numerals instead.

```
\usepackage [warn]{textcomp}
\newcommand\born[1]{\textborn\oldstylenums{#1}}
\raggedright
\fontfamily{phv}\selectfont Arno \born{29.11.1984},
\fontfamily{ccr}\selectfont Burkhard and Holger \born{8.11.1997}
```

If you own fonts that `textcomp` does not know about (or for some reason assumes that they implement a smaller subset than they actually do), you can inform the package about the font family in question by using the configuration file `textcomp.cfg`. For example, the commercial Lucida Blackletter originally contained only the basic ISO-Adobe glyphs, so `textcomp` takes a conservative approach and allows only these symbols. But nowadays it also contains the

<sup>1</sup>If the glyphs are directly accessed by manually switching to the TS1 encoding, as is done in the example, a restricting option (e.g., `safe`) will have no effect.

`\textohm` symbol, so by using `\DeclareEncodingSubset` after loading the package (or in the configuration file) you can typeset it in this font family as well.

We can now typeset  $\Omega$ , but then the ■ will fail without warning. 7-5-1

```
\usepackage[T1]{fontenc} \usepackage{textcomp} \raggedright
\DeclareEncodingSubset{TS1}{hlcf}{3}
\fontfamily{hlcf}\selectfont We can now typeset \textohm,
but then the \texteuro{} will fail without warning.
```

For details on the use of `\DeclareEncodingSubset` and the subset numbers used, see the documentation in `ltoutenc.dtx` in the standard L<sup>A</sup>T<sub>E</sub>X distribution.

### 7.5.5 exscale—Scaling large operators

Normally the font employed for large mathematical symbols is used in only one size. This set-up is usually sufficient, as the font includes most of the characters in several different sizes and (L<sup>A</sup>)T<sub>E</sub>X is specially equipped to automatically choose the symbol that fits best. However, when a document requires a lot of mathematics in large sizes—such as in headings—the selected symbols may come out too small. In this case, you can use the package `exscale`, which provides for math extension fonts in different sizes. The package only works for documents using Computer Modern math fonts. However, packages providing alternate math font set-ups often offer this functionality as a package option.

### 7.5.6 tracefnt—Tracing the font selection

The package `tracefnt` can be used to detect problems in the font selection system. This package supports several options that allow you to customize the amount of information displayed by NFSS on the screen and in the transcript file.

`errorshow` This option suppresses all warnings and information messages on the terminal; they will be written to the transcript file only. However, real errors will be shown on the terminal. Because warnings about font substitutions and so on can mean that the final result will be incorrect, you should carefully study the transcript file before printing an important publication.

`warningshow` When this option is specified, warnings and errors are shown on the terminal. This setting gives you the same amount of information as L<sup>A</sup>T<sub>E</sub>X<sup>2ε</sup> does without the `tracefnt` package loaded.

`infoshow` This option is the default when you load the `tracefnt` package. Extra information, which is normally only written to the transcript file, is now also displayed on your terminal.

`debugshow` This option additionally shows information about changes to the text font and the restoration of such fonts at the end of a brace group or the end of an environment. Be careful when you turn on this option because it can produce very large transcript files that can quickly fill up your disk space.

In addition to these “standard tracing” options,<sup>1</sup> the package `tracefnt` supports the following options:

`pausing` This option turns all warning messages into errors to help in the detection of problems in important publications.

`loading` This option shows the loading of external fonts. However, if the format or document class you use has already loaded some fonts, then these will not be shown by this option.

### 7.5.7 nfssfont.tex—Displaying font tables and samples

The L<sup>A</sup>T<sub>E</sub>X distribution comes with a file called `nfssfont.tex` that can be used to test new fonts, produce font tables showing all characters, and perform similar font-related operations. This file is an adaption of the program `testfont.tex`, which was originally written by Donald Knuth. When you run this file through L<sup>A</sup>T<sub>E</sub>X, you will be asked to enter the name of the font to test. You can answer either by giving the external font name without any extension—such as `cmr10` (Computer Modern Roman 10pt)—if you know it, or by giving an empty font name. In the latter case you will be asked to provide a NFSS font specification, that is, an encoding name (default T1), a font family name (default `cmr`), a font series (default `m`), a font shape (default `n`), and a font size (default 10pt). The package then loads the external font corresponding to that classification.

Next, you will be requested to enter a command. Probably the most important one is `\table`, which produces a font chart like the one on page 434. Also interesting is `\text`, which produces a longer text sample. To switch to a new test font, type `\init`; to finish the test, type `\bye` or `\stop`; and to learn about all the other possible tests (at the moment basically still tailored for the OT1 encoding), type `\help`.

With a bit of care you can also use the program non-interactively, provided your L<sup>A</sup>T<sub>E</sub>X implementation supports input redirection. For example, if the file `nfssfont.in` contains

```
cmr10
\table \newpage \init
T1
cmss
bx
n
10
\text \bye
```

then a call like `latex nfssfont < nfssfont.in` (on UN\*X implementations)

<sup>1</sup>It is suggested that package writers who support tracing of their packages use these four standard names if applicable.

would read all input from that particular file, first producing a glyph chart for the font `cmr10` and then creating a text sample for `T1/cmss/bx/n/10`.

Two things are important here. First, the `nfssfont.tex` program issues an implicit `\init` command, so the first input line either should contain a font name or should be completely empty (to indicate that an NFSS classification follows). Second, the input to `\init` must appear on individual lines with nothing else (not even a comment, as that would mask the line ending), because the line ending indicates the end of the answer to a question like “Font encoding [T1] : `\encoding=`” that you would get if you ran the program interactively.

## 7.6 PSNFSS—PostScript fonts with L<sup>A</sup>T<sub>E</sub>X

The PSNFSS bundle, originally developed by Sebastian Rahtz, offers a complete working set-up of the L<sup>A</sup>T<sub>E</sub>X font selection scheme for use with common PostScript fonts, covering the “Base 35” fonts (which are built into any Level 2 PostScript printing device and the ghostscript interpreter) and the free Charter and Utopia fonts.<sup>1</sup> The current implementation of PSNFSS is maintained by Walter Schmidt and is part of the required set of support files for L<sup>A</sup>T<sub>E</sub>X that should be available with every L<sup>A</sup>T<sub>E</sub>X installation.

For normal use you will probably have to include only one (or more) of the packages listed in Table 7.7 on the next page to change the default Roman, sans serif, and/or typewriter typefaces. If you study this table you will notice that only two packages attempt to set up new fonts for math and that the first eight packages only change fonts in one of the three text font categories. Thus, to get Times as the Roman text font, Helvetica as the sans serif text font, and Courier as the typewriter text font, one would need to load `mathptmx`, `helvet`, and `courier`. So why is the `times` package, which does this all in one go, considered obsolete?

*Scale Helvetica to  
blend with  
surrounding fonts*

One reason is that Helvetica, if loaded at its nominal size, is actually too large to blend well with Times or Courier. That does not matter so much in a design where Helvetica is used only for headings, say. But if these fonts are going to be mixed in running text (something that is made easy by L<sup>A</sup>T<sub>E</sub>X commands such as `\textsf`), then using a package such as `times` will produce questionable results. The `helvet` package, on the other hand, offers the ability to scale the fonts by specifying the option `scaled`, which scales the fonts down to 95% of the requested size. This option is actually a keyword/value option, so that even finer control is possible—`scaled=0.92` would load the fonts at 92% of their nominal size.

There is, however, one set of circumstances in which you might wish to use the `times` package after all: when you do not want to change the math font set-up, or you want to use some other set of fonts for math. In that case you can still load the `helvet` package afterwards to apply scaling.

<sup>1</sup>If the Utopia fonts are missing on your T<sub>E</sub>X installation they can be downloaded from the CTAN directory `fonts/utopia`. Consult the documentation of your T<sub>E</sub>X system on how to install them.

<i>Package</i>	<i>Roman Font</i>	<i>Sans Serif Font</i>	<i>Typewriter Font</i>	<i>Formulas</i>
(none)	CM Roman	CM Sans Serif	CM Typewriter	CM Math
mathptmx	Times			Times + Symbol
mathpazo	Palatino			Palatino + Pazo
charter	Charter			
utopia*	Utopia			
chancery	Zapf Chancery			
helvet		Helvetica		
avant		Avant Garde		
courier			Courier	
bookman	Bookman	Avant Garde	Courier	
newcent	New Century Schoolbook	Avant Garde	Courier	
<i>Obsolete Packages</i>				
times	Times	Helvetica	Courier	
palatino	Palatino	Helvetica	Courier	
mathptm	Times			Times + Symbol + CM
mathpple	Palatino			Palatino + Symbol + Euler

\*An alternative package that includes math support is *fourier*, which is described in Section 7.7.7.

Table 7.7: Fonts used by PSNFSS packages

The PSNFSS bundle uses the Karl Berry naming scheme [19] throughout; the classification and the external font names are shown in Table 7.8 on the following page. Using this table, it is easy to access individual fonts without loading any package, such as via a call to `\usefont` (see Example 7-6-1 below). Because these fonts can be easily scaled to any size, this method offers attractive possibilities when designing headings or title pages, as it facilitates the use of sizes different from those created with the standard LATEX font size commands.

*Direct access to  
fonts*

# Utopia- Bold

7-6-1

```
\centering
\fontsize{20mm}{22mm}\selectsize
\usefont{T1}{put}{b}{n}\selectfont
```

Utopia-Bold

Family	Series	Shape(s)	External PostScript font names and examples
<i>Times</i> (OT1, T1, TS1)			
ptm	m	n, sl, it, sc	Times-Roman (ptmr), Times-Italic (ptmri)
ptm	b, (bx)	n, sl, it, sc	Times-Bold (ptmb), Times-BoldItalic (ptmbi)
<i>Palatino</i> (OT1, T1, TS1)			
ppl	m	n, sl, it, sc	Palatino-Roman (pplr), Palatino-Italic (pplri)
ppl	b, (bx)	n, sl, it, sc	Palatino-Bold (pplb), Palatino-BoldItalic (pplbi)
<i>New Century Schoolbook</i> (OT1, T1, TS1)			
pnc	m	n, sl, it, sc	NewCenturySchlbk-Roman (pnctr), NewCenturySchlbk-Italic (pncri)
pnc	b, (bx)	n, sl, it, sc	NewCenturySchlbk-Bold (pnccb), NewCenturySchlbk-BoldItalic (pnccbi)
<i>Bookman</i> (OT1, T1, TS1)			
pbk	m	n, sl, it, sc	Bookman-Light (pbkl), Bookman-LightItalic (pbkli)
pbk	b, (bx)	n, sl, it, sc	Bookman-Demi (pbkd), Bookman-DemiboldItalic (pbkdi)
<i>Helvetica</i> (OT1, T1, TS1)			
phv	m	n, sl, sc	Helvetica (phvr), Helvetica-Oblique (phvro)
phv	b, (bx)	n, sl, sc	Helvetica-Bold (phvb), Helvetica-BoldOblique (phvbo)
phv	mc	n, sl, sc	Helvetica-Narrow (phvrrn), Helvetica-Narrow-Oblique (phvron)
phv	bc	n, sl, sc	Helvetica-Narrow-Bold (phvbrn), Helvetica-Narrow-BoldOblique (phbon)
<i>Avant Garde</i> (OT1, T1, TS1)			
pag	m	n, sl, sc	AvantGarde-Book (pagk), AvantGarde-BookOblique (pagko)
pag	b, (bx)	n, sl, sc	AvantGarde-Demi (pagd), AvantGarde-DemiOblique (pagdo)
<i>Courier</i> (OT1, T1, TS1)			
pcr	m	n, sl, sc	Courier (pcrr), CourierOblique (pcrro)
pcr	b, (bx)	n, sl, sc	Courier-Bold (pcrb), Courier-BoldOblique (pcrbo)
<i>Zapf Chancery</i> (OT1, T1, TS1)			
pzc	m	it	ZapfChancery-MediumItalic (pzcmi)
<i>Utopia</i> (OT1, T1, TS1)			
put	m	n, sl, it, sc	Utopia-Regular (putr), Utopia-Italic (putri)
put	b, (bx)	n, sl, it, sc	Utopia-Bold (putb), Utopia-BoldItalic (putbi)
<i>Charter</i> (OT1, T1, TS1)			
bch	m	n, sl, it, sc	CharterBT-Roman (bchr), CharterBT-Italic (bchri)
bch	b, (bx)	n, sl, it, sc	CharterBT-Bold (bchb), CharterBT-BoldItalic (bchbi)
<i>Symbol and Zapf Dingbats</i> (U)			
psy	m	n	Symbol (psyr): Σψμβολ
pzd	m	n	Zapf Dingbats (pzdr): *•□* ♡♦■*●○▼▲

Table 7.8: Classification of font families in the PSNFSS distribution

The PSNFSS collection contains only two packages that modify the math set-up: `mathptmx` selects math fonts that blend with Times Roman (described in Section 7.6.2 on page 376) and `mathpazo` selects math fonts designed to work with Palatino (see Section 7.6.3 on page 377). The packages `mathptm` and `mathppl` are predecessors that are retained mainly for backward compatibility. Outside the PSNFSS collection a few other packages that change the math font set-up are available (in most cases involving commercial fonts). Some free packages are described in Section 7.7 on page 381, including one that uses Utopia for typesetting text and mathematics. A collection of sample pages with different text and math fonts appears in Section 8.8.3.

Most document classes designed for use with Computer Modern set up a leading (`\baselineskip`) of 10pt/12pt. This may appear to be too tight for several of the PostScript font families shown below, due to a larger x-height of the fonts. However, as this is a matter of document design and also depends on the chosen line width and other factors, the packages in the PSNFSS collection make no attempt to adjust the leading. For a given document class you can change the leading by a *factor* by issuing the declaration `\linespread{factor}` in the preamble. For example, `\linespread{1.033}` would change the leading from, say, 12pt to approximately 12.4pt. For best results, however, one needs to use a document class designed for the selected document fonts or, lacking such a class, to redefine the commands `\normalsize`, `\footnotesize`, and so on (see page 343 for details). Also remember that changing the leading might result in a noticeable number of “Underfull `\vbox`” warnings, if the `\textheight` is no longer an integral number of text lines (see page 930 for further details).

*Adjusting the leading*

By default, L<sup>A</sup>T<sub>E</sub>X selects a Roman typeface as the document font. Packages like `helvet` or `avant` change the default sans serif typeface (by changing `\sfdefault`) but do not change the default document font family. If such a typeface should be used as the document font, issue the line

*Sans serif as document typeface*

```
\renewcommand\familydefault{\sfdefault}
```

in the preamble of your document.

Besides supporting the common PostScript text fonts, the PSNFSS collection contains the interesting `pifont` package. It sets up various commands for use with the so-called Pi fonts (i.e., special symbol fonts like Zapf Dingbats and Symbol). It is described in Section 7.6.4 on page 378.

### 7.6.1 Font samples for fonts supported by PSNFSS

This section provides textual samples of the fonts supported by the PSNFSS collection. The examples were generated by explicitly selecting the font size and leading via a call to `\fontsize` and then selecting the font with a `\usefont` command. For example, the first sample was generated with `\fontsize{9}{13}\usefont{T1}{pag}{m}{n}`.

*ITC Avant Garde Gothic*  
9pt/13pt (pag)

Avant Garde Gothic was designed by Herb Lubalin and Tom Carnase based on the distinctive logo designed for *Avant Garde* magazine. It is a geometric sans serif type with basic shapes built from circles and lines. Effective for headlines and short texts, but it needs generous leading. A (commercially available) condensed version that better retains legibility in lengthier texts was designed by Ed Benguiat.

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?

*ITC Bookman*  
10pt/12pt (pbk)

Bookman was originally designed in 1860 by Alexander Phemister for the Miller & Richard foundry in Scotland (commercially available from Bitstream). The ITC revival by Ed Benguiat has a larger x-height and a moderate stroke contrast that is well suited for body text and display applications.

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 phoenix's official rôle in fluffy soufflés?

*Bitstream Charter*  
10pt/12.4pt (bch)

Bitstream Charter is an original design by Matthew Carter intended to work well on low-resolution devices; hence, it contains squared serifs and avoids excessive use of curves and diagonals. It is useful for many applications, including books and manuals.

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*  
10pt/12pt (pcr)

Courier is a wide-running, thin-stroked monospaced font. It was designed by Howard Kettler of IBM and later redrawn by Adrian Frutiger. These days it is often used in combination with Times Roman, producing a striking contrast. One reason for the popularity of this combination is certainly its availability on any PostScript device. For alternatives see Section 7.7.4.

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 was originally designed by Max Miedinger for the Haas foundry of Switzerland, hence the name. It was later extended by the Stempel foundry, with further refinements being made by Mergenthaler Linotype in the United States. Helvetica is claimed to be the most popular typeface of all time.

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?

The New Century Schoolbook typeface was designed at the beginning of the 20th century by Morris Benton of the American Type Founders. It was created in response to a publisher's commission that sought a typeface with maximum legibility for elementary schoolbooks.

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, designed by Hermann Zapf, is one of the most widely used typefaces today. You can feel the brush that created it, which gives it a lot of elegance. Although originally designed as a display typeface, due to its legibility Palatino soon gained popularity as a text face as well.

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?

Times Roman is Linotype's version of Monotype's Times New Roman, which was originally designed under the direction of Stanley Morison for the *London Times* newspaper. The Adobe font that is built into many PostScript devices uses Linotype's 12-point design.

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?

Utopia, designed by Robert Slimbach, combines the vertical stress and pronounced stroke contrast of 18th-century Transitional types with contemporary innovations in shape and stroke details.

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?

*ITC Zapf Chancery  
10pt/12pt (pzc)* Zapf Chancery is a contemporary script based on chancery handwriting, developed during the Italian Renaissance for use by the scribes in the papal offices. Highly legible, it can be usefully applied for short texts and applications like invitations and awards.

*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?*

### 7.6.2 mathptmx—Times Roman in math and text

The mathptmx package makes Times the document text font and implements a math font set-up for use with such documents. It builds on freely available Type 1 PostScript fonts and is, therefore, somewhat inferior to some of the commercially available solutions that offer fonts especially designed for this purpose. Nevertheless, it has the advantage of being (at least potentially) available in every TeX installation.<sup>1</sup>

The mathptmx package was co-authored by Alan Jeffrey, Sebastian Rahtz, and Ulrik Vieth. It was based upon earlier work by Alan Jeffrey [72], in particular the mathptm package (the predecessor to mathptmx) and, most importantly, the fontinst system [57, pp.393–404], which provided the initial breakthrough in making PostScript fonts generally available with TeX.

Technically, the mathptmx package uses a collection of virtual fonts that implement the math fonts needed for TeX by drawing them from several font resources—Times Roman, Times Italic, Symbol, various Computer Modern fonts (mainly for delimiters, big operators, arrows, and the like), and Ralph Smith's Formal Script (RSFS). The RSFS fonts are a better solution for a script/calligraphic alphabet than Zapf Chancery, which is used in mathptm for this purpose.

An example showing a trigonometric function:

$$\sin \frac{\alpha}{2} = \pm \sqrt{\frac{1 - \cos \alpha}{2}}$$

The script looks like this: *A B C.*

```
\usepackage{mathptmx}
An example showing a trigonometric function:
\[\sin \frac{\alpha}{2} = \pm \sqrt{\frac{1 - \cos \alpha}{2}}\]
The script looks like this: \$\mathcal{ABC}\$.
```

7-6-2

It has some features in common with the mathpazo package. First, when loaded with the option slantedGreek, uppercase Greek letters are slanted instead

<sup>1</sup>The TeX installation must support virtual fonts, which is the case for nearly every distribution.

of being upright (the default). In either case the two extra commands `\upDelta` and `\upOmega` will print an upright  $\Delta$  and  $\Omega$ , respectively. Second, the functionality of the `exscale` package is automatically provided: thus big operators and delimiters scale with the current font size.

On the downside, the package disables `\boldmath` for the simple reason that no bold version of the Adobe Symbol font exists. You can get, of course, a bold math alphabet with `\mathbf`, but this gives you only upright Latin characters and digits. In particular, using the `bm` package to make individual symbols bold will produce questionable results, as the best the `\bm` command can do is to produce “poor man’s bold” by overprinting the symbols with slight offsets.

```
\usepackage{mathptmx,bm}
Bold is difficult to achieve: \alpha \neq A and at
best looks questionable: A \neq \mathbf{A} = \bm{\alpha} - \bm{\gamma}.
```

 Proper bold faces missing

Another (small) potential problem is that the commands `\jmath`, `\coprod`, and `\amalg` are unavailable. If either issue turns out to be a real problem, then alternatives to consider are the TX fonts (Section 7.7.5) and the commercial solutions MathTime (Professional) by Michael Spivak and TM-Math by MicroPress.

### 7.6.3 mathpazo—Palatino in math and text

A package named `mathpazo` supporting Adobe Palatino with matching math fonts was originally developed by Walter Schmidt based on earlier work by Aloisius Helminck. It used the same approach as `mathptm`; that is, it was built on the virtual font mechanism, combining symbols from Palatino, Symbol, Euler, and CM Math. As these fonts only partly match the style of Palatino, Diego Puga developed a set of Type 1 PostScript fonts (Pazo Math) intended to repair the defects apparent in the `mathpazo` solution. The Pazo Math fonts contain glyphs that are unavailable in Palatino and for which Computer Modern or glyphs from Symbol look odd when combined with Palatino. These include a number of math glyphs, the uppercase Greek alphabet (upright and slanted), a blackboard bold alphabet, as well as several other glyphs (such as the euro symbol) in regular and bold weights and upright and slanted shapes.

The fonts are accessible with the `mathpazo` package developed by Diego Puga and Walter Schmidt as part of the PSNFSS collection. It makes Palatino the document text font and provides a math set-up that works by using virtual fonts accessing Palatino Italic, the Math Pazo fonts, and CM fonts (for the remaining symbols).

An example showing a trigonometric function:

$$\sin \frac{\alpha}{2} = \pm \sqrt{\frac{1 - \cos \alpha}{2}}$$

7-6-4 The script looks like this: ABC.

```
\usepackage{mathpazo}
An example showing a trigonometric function:
\[\sin \frac{\alpha}{2} = \pm \sqrt{\frac{1 - \cos \alpha}{2}}\]
The script looks like this: \$\mathcal{ABC}\$.
```

This package is very similar to the `mathptmx` package. In particular, it supports the option `slantedGreek` to make uppercase Greek letters slanted instead of upright (the default). In either case the two extra commands `\upDelta` and `\upOmega` will print an upright  $\Delta$  and  $\Omega$ , respectively. Also, it provides the functionality of the `exscale` package.

However, in contrast to the `mathptmx` package, which uses the Adobe Symbol font, for which no bold-weight variant exists, the `mathpazo` package provides full access to symbols in a bold weight.

```
\usepackage{mathpazo,bm}
Bold is easy to achieve: \boldsymbol{\alpha} \neq A and
blends well: A \neq \mathbf{A} = \alpha - \gamma.          7-6-5
```

As mentioned above, the Pazo Math fonts contain a blackboard bold alphabet, which can be accessed through the math alphabet identifier `\mathbb`. The font contains the uppercase Latin letters and the digit “1”. Be careful, however: all other digits are silently ignored!

```
\usepackage{mathpazo}
\mathbb{ABCDEFGHIJK} \mathbb{0123}          7-6-6
```

If `\mathbb` should select a different alphabet, provided by some other package, it is best to suppress the Pazo Math one by using the option `noBBpl` when loading the package.

The package also offers two additional options that deal with the use of commercially available Palatino fonts<sup>1</sup> for the text font: `sc` selects Palatino with true small capitals (font family name `pplx`) and `osf` selects Palatino with small caps and old-style numerals (font family name `pplj`) instead of basic Palatino (`ppl`).

#### 7.6.4 pifont—Accessing Pi and Symbol fonts

Fonts containing collections of special symbols, which are normally not found in a text font, are called Pi fonts. One such font, the PostScript font Zapf Dingbats, is available if you use the `pifont` package originally written by Sebastian Rahtz and now incorporated as part of `PSNFSS`.

*Accessing glyphs from Zapf Dingbats*

The directly accessible characters of the PostScript Zapf Dingbats font are shown in Table 7.9 on the next page. A given character can be chosen via the `\ding` command. The parameter for the `\ding` command is an integer that specifies the character to be typeset according to the table. For example, `\ding{'46}` gives  $\textcircled{O}$ .

<sup>1</sup>These fonts are commercially available and are *not* part of the Base 35 fonts.

	'0	'1	'2	'3	'4	'5	'6	'7	
'04x		♪	♫	♪	♪	♪	♪	♪	"2x
'05x	♪	¤	¤	¤	¤	¤	¤	¤	
'06x	¤	∞	¤	✓	✓	✗	✗	✗	"3x
'07x	✗	✚	✚	+	✚	†	†	†	
'10x	✖	✡	✚	❖	❖	❖	❖	❖	"4x
'11x	★	☆	●	☆	☆	★	☆	★	
'12x	☆	*	*	*	*	*	*	*	"5x
'13x	*	*	*	*	*	*	*	✿	
'14x	✿	✿	✿	*	*	*	*	*	"6x
'15x	*	*	*	*	●	○	■	□	
'16x	□	□	□	▲	▼	◆	❖	▷	"7x
'17x	।	।	।	,	,	,	,	,	
'24x		♩	♩	♩	♩	♩	♩	♩	"Ax
'25x	♣	♦	♥	♠	①	②	③	④	
'26x	⑤	⑥	⑦	⑤	⑨	⑩	①	②	"Bx
'27x	③	④	⑤	⑥	⑦	⑧	⑨	⑩	
'30x	①	②	③	④	⑤	⑥	⑦	⑧	"Cx
'31x	⑨	⑩	①	②	③	④	⑤	⑥	
'32x	⑦	⑧	⑨	⑩	→	→	↔	↓	"Dx
'33x	↗	→	↗	→	→	→	→	→	
'34x	➡	➡	➡	➡	➡	➡	➡	➡	"Ex
'35x	➡	➡	➡	➡	➡	➡	➡	➡	
'36x	➡	➡	➡	➡	➡	➡	➡	➡	"Fx
'37x	➡	➡	➡	➡	➡	➡	➡	➡	
	"8	"9	"A	"B	"C	"D	"E	"F	

Table 7.9: Glyphs in the PostScript font Zapf Dingbats

The dinglist environment is a variation of the itemize list. The argument specifies the number of the character to be used at the beginning of each item.

- The first item.
- The second item in the list.
- A final item.

```
\usepackage{pifont}
\begin{dinglist}{"E4}
\item The first item. \item The second
item in the list. \item A final item.
\end{dinglist}
```

The environment `dingautolist` allows you to build an enumerated list from a sequence of Zapf Dingbats characters. In this case, the argument specifies the number of the first character of the sequence. Subsequent items will be numbered by incrementing this number by one. This makes some starting positions like '254, '266, '300, and '312 (i.e., in octal notation) in Table 7.9 on the preceding page very attractive, as differently designed circled number sequences (1–10) start there.

- ① The first item in the list.
- ② The second item in the list.
- ③ The third item in the list.

References to list items work as expected: ①, ②, ③

```
\usepackage{pifont}
\begin{dingautolist}{'300}
\item The first item in the list.\label{lst:a}
\item The second item in the list.\label{lst:b}
\item The third item in the list.\label{lst:c}
\end{dingautolist}
```

References to list items work as expected:  
`\ref{lst:a}, \ref{lst:b}, \ref{lst:c}`

7-6-8

You can fill a complete line (with 0.5 inch space at left and right) with a given character using the command `\dingline`, where the argument indicates the desired character. For filling parts of a line, use the command `\dingfill`. This command works similar to L<sup>A</sup>T<sub>E</sub>X's `\dotfill` command, but uses the specified glyph instead of dots.

<code>\usepackage{pifont}</code> <code>\dingline{35}</code> <span style="float: right;"><code>\par\medskip</code></span> <code>\noindent\dingfill{233} text text</code> <code>\dingfill{235} text text \dingfill{236}</code>	<code>\par\medskip</code> <code>\dingfill{233} text text</code> <code>\dingfill{235} text text \dingfill{236}</code>
---	--

7-6-9

Besides providing direct support for the Zapf Dingbats font, the `pifont` package includes a general mechanism for coping with any Pi font that conforms to the NFSS classification `U/family/m/n`—for example, the Symbol font with the family name `psy`.

To access individual glyphs from such a Pi font, use the `\Pisymbol` command, which takes the *family* name as its first argument and the glyph position in the font as its second argument. Using this command one can readily access the characters in the Symbol font, shown in Table 7.10 on page 382. For example, `\Pisymbol{psy}{210}` gives ®. In fact, `\ding` (discussed earlier) is simply an abbreviation for `\Pisymbol` with the first argument set to `pzd`.

When only Greek letters are desired, you can use the `\Pifont` command and consult the correspondence in Table 7.10. Clearly, this solution is no match for a properly designed font for the Greek language but it might serve in an emergency—for example, to typeset the text above the entrance of Plato's Academy that states “Only geometers may enter”:

```
\usepackage{pifont}
\Pifont{psy} MHDEIS\ AGEWMETPHTOS\ EISITW.
```

7-6-10

You can also make itemized lists using `Pilist` or enumerated lists using the `Piautolist` environments as follows:

- ⇒ The first item.
- ⇒ The second.
- ★ The first item.
- ★ The second.
- ★ The third.

```
\usepackage{pifont}
\begin{Pilist}{psy}{'336}
  \item The first item. \item The second.
\end{Pilist}
\begin{Piautolist}{pzd}{'115}
  \item The first item. \item The second.
  \item The third.
\end{Piautolist}
```

7-6-11

The `\dingline` and `\dingfill` commands are also merely abbreviations for the more general commands `\Piline` and `\Pifill`, as shown below. The example reveals curious gaps in the last line. They are due to `\Piline` and `\Pifill` typesetting their symbols on an invisible grid so that symbols on different lines come out vertically aligned.

```
\usepackage{pifont}
\dingline{36} \par \medskip
\dingfill{psy}{222} text
\dingfill{psy}{219} text \dingfill{psy}{220}
```

7-6-12

## 7.7 A collection of font packages

So far we have discussed font-related packages that belong to core L<sup>A</sup>T<sub>E</sub>X—that is, packages that are either part of the base distribution or, as for PSNFSS, are part of the “required” additions. There are, however, many other packages that provide font customization possibilities. Nowadays most of them are part of a L<sup>A</sup>T<sub>E</sub>X distribution. If they are not available on your local system, you can obtain them from an electronic archive or from a T<sub>E</sub>X organization; see Appendix C.

The packages described in the current section modify the document text fonts (and sometimes the math font set-up). As the section title indicates, they represent merely a selection of what is available. Further pointers can be found in the online package catalogue [169] or in one of the FAQ documents on L<sup>A</sup>T<sub>E</sub>X [46, 141].

### 7.7.1 eco—Old-style numerals with Computer Modern

The original Computer Modern fonts contain a set of old-style digits (e.g., 1982) as part of their math fonts, not because old-style numerals have anything to do with math, but because Donald Knuth tried to use the limited font space available in the most economical way, using some free slots in the math fonts to deposit the glyphs there. As the EC font implementation only concerned itself with a new

	'0	'1	'2	'3	'4	'5	'6	'7	
'04x		!	∀	#	Ξ	%	&	Ξ	
'05x	(	)	*	+	,	-	.	/	"2x
'06x	0	1	2	3	4	5	6	7	"3x
'07x	8	9	:	;	<	=	>	?	
'10x	≈	A	B	X	Δ	E	Φ	Γ	
'11x	H	I	ø	K	Λ	M	N	O	"4x
'12x	Π	Θ	P	Σ	T	Y	ς	Ω	
'13x	Ξ	Ψ	Z	[	⋮	]	⊥	–	"5x
'14x	–	α	β	χ	δ	ε	φ	γ	
'15x	η	ι	φ	κ	λ	μ	ν	ο	"6x
'16x	π	θ	ρ	σ	τ	υ	ω	ω.	
'17x	ξ	ψ	ζ	{		}	~		"7x
'24x		Υ	'	≤	/	∞	f	♣	"Ax
'25x	♦	♥	♠	↔	←	↑	→	↓	
'26x	◦	±	"	≥	×	∞	∂	•	"Bx
'27x	÷	≠	≡	≈	...		—	↔	
'30x	ℵ	ℑ	ℜ	∅	⊗	⊕	∅	∩	"Cx
'31x	∪	⊃	⊇	⊊	⊂	⊆	∈	∉	
'32x	∠	∇	®	©	™	Π	√	.	"Dx
'33x	¬	∧	∨	↔	⇐	↑↑	⇒	↓↓	
'34x	◊	⟨	®	©	™	Σ	(	)	"Ex
'35x	ℓ	Γ		ℓ	ſ	{	ℓ		
'36x		⟩	ſ	ſ		⟩	)		
'37x	)	]		]	)	{	)		"Fx
	"8	"9	"A	"B	"C	"D	"E	"F	

Table 7.10: Glyphs in the PostScript font Symbol

font encoding for text, this anomaly in the math fonts was unfortunately kept.<sup>1</sup> Actually, the designers of the text companion encoding (TS1) added old-style numerals to that encoding, but so far this is of little practical relevance because too many font families implement only a subset of the TS1 encoding. See Section 7.5.4, page 367, for more information.

<sup>1</sup> Justin Ziegler together with the L<sup>A</sup>T<sub>E</sub>X3 project team developed a rationalized font encoding design for 256-glyph math fonts [174]. Unfortunately, until now his theoretical work has not been implemented other than in a prototype using virtual fonts [40].

For easy access to old-style numerals hidden in the math fonts, L<sup>A</sup>T<sub>E</sub>X provides the command `\oldstylenums`, which can be used in text and within formulas. In its argument you should place the digits that you want to typeset as non-aligning digits. If the command is used in text, spaces in the argument are honored, but you should not try to put characters other than digits into it or the results will be unpredictable. One problem with the default definition of this command is that it will always generate old-style numerals from Computer Modern Roman, regardless of the surrounding fonts in use. For this reason the `textcomp` package contains a redefinition that produces the old-style numerals from the current font, provided they are available in the current font family; see Section 7.5.4 for details.

*Basic L<sup>A</sup>T<sub>E</sub>X support  
for old-style  
numerals*

This approach for obtaining old-style numerals might be adequate if lining numerals are the norm and old-style numerals are required only once in a while. But in a document layout in which all text numerals are supposed to be presented in old-style it is not really acceptable to require the author to explicitly mark up every occurrence in this way. What is needed in such a case are text fonts that contain old-style instead of lining numerals in the standard slot positions.

The EC fonts contain both lining and old-style numerals (albeit in a somewhat inconvenient position), so it was just a matter of time until someone developed a series of virtual fonts that reencode the fonts to make old-style numerals be the default text numbers. The `eco` fonts by Sebastian Kirsch provide this reencoding and can be accessed by loading the `eco` package. Note that the package affects only the text numbers, so it is important to mark up mathematical digits properly. Otherwise, you will obtain a result like the one shown in the example.

In 1996 Sebastian developed fonts producing old-style numerals in text but lining numerals in math. So do not write “the value can be 1 or –1”, as both numbers should be lining numerals. In text lining numerals can be obtained as well: 1996.

7-7-1

```
\usepackage{eco}
In 1996 Sebastian developed fonts producing
old-style numerals in text but lining numerals
in math. So do not write ``the value can be 1
or $-1$', as both numbers should be lining
numerals. In text lining numerals can be
obtained as well: \newstylenums{1996}.
```

## 7.7.2 ccfonts, concmath—The Concrete fonts

For the text of his book *Concrete Mathematics* [59], Donald Knuth designed a new typeface [92] to go with the Euler mathematics fonts designed by Hermann Zapf [173]. This font family, called Concrete Roman, was created from the Computer Modern METAFONT sources by supplying different parameter settings.

Starting from the work done for the EC fonts, it was relatively easy to create Concrete Roman fonts in T1 and TS1 encodings (original work by Frank Mittelbach; current version by Walter Schmidt). The fonts available in these families are shown in Table 7.11 on the following page. Ulrik Vieth used the construction method outlined by Knuth [92] to develop a companion set of Concrete Math fonts including the full range of AMS symbols (as provided by the `amssymb` or `amsfonts` package).

Family	Series	Shape(s)	Example of the Typeface
<i>Concrete Roman (T1, TS1, OT1)</i>			
ccr	m	n, it, sl, sc	Concrete Roman medium
ccr	c	sl	<i>Concrete Roman condensed slanted (only OT1 and 9pt)</i>
<i>Concrete Math (OML)</i>			
ccm	m	it	<i>Concrete Math.</i> $\alpha \Omega$
<i>Concrete Math (OMS)</i>			
ccy	m, c	n	$\mathcal{C} \nabla \sqcup \mathcal{M} \sqcup \neg \otimes \otimes$

Table 7.11: Classification of the Concrete font families

The first package that provided access to these font families for normal text was `beton` (by Frank Jensen). The following example shows the combination of Concrete text and Euler math fonts (see also Section 7.7.10 on page 396):

Concrete Roman blends well with Euler Math,  
as can be seen with

$$\sum_{0 \leq k < n} k = \frac{n(n-1)}{2}$$

```
\usepackage{beton,euler}
Concrete Roman blends well with Euler Math,
as can be seen with
\[\sum_{0 \leq k < n} k = \frac{n(n-1)}{2}\]
```

A more recent development that also provides the use of Concrete fonts for math and supports the T1 and TS1 encodings is the `ccfonts` package (by Walter Schmidt). Both packages take care of small but important typographical details, such as increasing the value of `\baselineskip` slightly (see discussion on the facing page). As the Concrete fonts have no boldface series, the `ccfonts` package offers the option `boldsans` to use the semibold series of the Computer Modern Sans fonts as a replacement. As a result, without any further adjustments, headings in standard classes will be typeset using this font series.

## 1 Testing headings

An example showing a trigonometric function:

$$\sin \frac{\alpha}{2} = \pm \sqrt{\frac{1 - \cos \alpha}{2}}$$

The script looks like this: `ABC.`  
From `textcomp`: `$ € * ω + ...`

```
\usepackage[boldsans]{ccfonts}
\usepackage[full]{textcomp}
\usepackage{ragged2e} %small measure
\section{Testing headings}
An example showing a trigonometric function:
\[\sin \frac{\alpha}{2} =
\pm \sqrt{\frac{1 - \cos \alpha}{2}}\]
The script looks like this: $€*ω+...
From textcomp: \textdollaroldstyle\texteuro\
\textborn\textmarried\textdied\ldots
```

7-7-2

7-7-3

Family	Series	Shape(s)	Example of the Typeface
<i>CM Bright (OT1, T1, TS1)</i>			
cmbr	m	n, sl	CM Bright medium
cmbr	sb	n, sl	<i>CM Bright semibold slanted</i>
cmbr	bx	n	<b>CM Bright bold extended</b>
<i>CM Typewriter Light (OT1, T1, TS1)</i>			
cmtl	m	n, sl	Typewriter Light normal
<i>CM Bright Math (OML)</i>			
cmbrm	m, b	it	<i>Bright Math.</i> $\alpha \Omega$
<i>CM Bright Math (OMS)</i>			
cmbrs	m	n	$\mathcal{B} \nabla \rangle \} \langle \sqcup M \sqcap \neg \oslash \otimes$

Table 7.12: Classification of the Computer Modern Bright font families

Because the Concrete fonts are of considerably heavier weight than, say, Computer Modern, it is advisable to use them with a larger leading than most document classes provide by default. For this reason the package automatically enlarges the leading to 10/13 and similar ratios for other document sizes. If this adjustment is undesirable for some reason, it can be canceled with the option `standard-baselineskips`.

The feature provided by the `exscale` package is available as the package option `exscale`; see Section 7.5.5 on page 368 for details. The `exscale` package itself cannot be used because it is set up to work with only Computer Modern math fonts.

If the `amssymb` or `amsfonts` package is loaded, the `ccfonts` package automatically arranges to use the Concrete variants of the AMS symbol fonts.

Finally, the package offers the option `slantedGreek` to make uppercase Greek letters slanted instead of being upright (default). The two extra commands `\upDelta` and `\upOmega` will always typeset an upright  $\Delta$  and  $\Omega$ , respectively.

### 7.7.3 `cmbright`—The Computer Modern Bright fonts

Another font family whose design is based on the METAFONT sources of the CM fonts are the Computer Modern Bright (CM Bright) fonts by Walter Schmidt, shown in Table 7.12. This family of sans serif fonts is designed to serve as a legible body font. It comes with matching typewriter and math fonts, including the AMS symbols.

Loading the `cmbright` package in the preamble ensures that these families are selected throughout the document. It is recommended that you combine this package with `fontenc`, as shown in the next example, to achieve proper hyphenation with languages other than English. All CM Bright fonts have fully implemented T1 and TS1 encoding support.

## 1 A CM Bright document

The CM Bright family contains typewriter fonts and matching fonts for math formulas, e.g.,

$$\sum_{0 \leq k < n} k = \frac{n(n - 1)}{2}$$

```
\usepackage[T1]{fontenc}
\usepackage{cmbright}
\section{A CM Bright document}
The CM Bright family contains
\textrm{typewriter} fonts and matching fonts
for math formulas, e.g.,
\[\sum_{0 \leq k < n} k = \frac{n(n - 1)}{2}\]
```

7-7-4

By default, the package selects a slightly larger leading than the default classes to account for the use of sans serif fonts; this can be canceled by specifying the package option `standard-baselineskip`. Also in other respects, this package works similarly to other works by Walter Schmidt: the option `slantedGreek` produces slanted uppercase Greek letters, with `\upDelta` and `\upOmega` typesetting an upright  $\Delta$  and  $\Omega$ , respectively. When the `amssymb` or `amsfonts` package is loaded, the `cmbright` package automatically arranges to use the CM Bright variants of the AMS symbol fonts.

The METAFONT implementation of the fonts is freely available from CTAN archives; Type 1 format versions are commercially sold by MicroPress. Recently, a freely available Type 1 (although without manual hinting) was made available by Harald Harders under the name `hfbright`. Moreover, as mentioned in Section 7.5.1, the freely available CM-Super Type 1 fonts also cover parts of the CM Bright fonts.

### 7.7.4 luximono—A general-purpose typewriter font

The choice of monospaced (typewriter) fonts for use in program listings and other applications is not very wide. Of course, with the Computer Modern fonts a suitable typewriter family (`cmtt`) is included, but if the main document fonts are being replaced, freely available choices for typewriter fonts are few. Adobe Courier runs very wide and for that reason alone it is often a poor choice. While staying with `cmtt` might be an option, the font may not blend well with the chosen document font.

Recently, with the release of version 4.2 of XFree86, the free implementation of the X Window System, a new, freely distributable, monospaced font family, called LuxiMono, has become available. This Type 1 encoded Postscript font comes with bold, oblique, and bold oblique versions (see Table 7.13 on the facing page). In that respect, it differs from other monospaced fonts, which are often offered only in medium series and more rarely in italic or oblique shapes.

Family	Series	Shape(s)	PostScript Font Names and Examples
<i>LuxiMono (T1, TS1)</i>			
ul9	m	n, sl	<b>LuxiMono</b> , <i>LuxiMono-Oblique</i>
ul9	b	n, sl	<b>LuxiMono-Bold</b> , <i>LuxiMono-BoldOblique</i>

Table 7.13: Classification of the LuxiMono font family

These fonts are original designs by Kris Holmes and Charles Bigelow (Bigelow and Holmes, Inc.), for which hinting and kerning tables have been added by URW++ Design and Development GmbH. The  $\text{\LaTeX}$  integration is provided through the `luximono` package written by Walter Schmidt.

The following example compares LuxiMono (scaled down to 85% using the option `scaled`), Computer Modern Typewriter, and Adobe Courier. LuxiMono still has the largest x-height (`\fontdimen5`) and, at the same time, the smallest width. Courier, running very wide, occupies the other end of the spectrum, with CM Typewriter being comfortably in between the two extremes.

The dazed brown fox quickly gave 12345-67890 jumps! x-height=4.50502pt (LuxiMono)	<code>\usepackage[T1]{fontenc}</code> <code>\usepackage[scaled=0.85]{luximono}</code> <code>\newcommand{\allletters}{The dazed brown</code> <code>fox quickly gave 12345--67890 jumps!</code> <code>x-height=\the\fontdimen5\font\ }</code> <code>\raggedright</code> <code>\texttt{\allletters (LuxiMono)}</code> <code>\par \renewcommand{\ttdefault}{cmtt}</code> <code>\texttt{\allletters (CM Typewriter)}</code> <code>\par \renewcommand{\ttdefault}{pcr}</code> <code>\texttt{\allletters (Adobe Courier)}</code>
---	---

If the option `scaled` is given without a value, the fonts are scaled down to 87%, which gives them a running length approximately equal to that of Computer Modern Typewriter. To get exactly the same running length, 0.87478 should be used for 10pt fonts, while for an 11pt document 0.86124 would be the correct value. This is due to the fact that LuxiMono scales linearly, while Computer Modern fonts have different designs for different sizes. Without scaling LuxiMono has the same running length as Adobe Courier.

This font contains a € symbol. 7-7-6 This font contains a € symbol.	<code>\usepackage[T1]{fontenc}\usepackage[scaled]{luximono}</code> <code>\usepackage[euro]{textcomp}</code> <code>\texttt{This font contains a \texteuro{} symbol.}</code> <code>\par \renewcommand{\ttdefault}{cmtt}</code> <code>\texttt{This font contains a \texteuro{} symbol.}</code>
--	---

Encoding	Family	Series	Shape(s)	Example of the Typeface
<i>TX Roman</i>				
OT1, T1, TS1, LY1	txr	m	n, it, sl, sc	TX Roman normal
OT1, T1, TS1, LY1	txr	bx, (b)	n, it, sl, sc	<b>TX Roman bold italic</b>
<i>TX Sans</i>				
OT1, T1, TS1, LY1	txss	m	n, (it), sl, sc	TX Sans normal
OT1, T1, TS1, LY1	txss	bx, (b)	n, (it), sl, sc	<b>TX Sans bold slanted</b>
<i>TX Typewriter</i>				
OT1, T1, TS1, LY1	txtt	m	n, (it), sl, sc	TX Typewriter normal
OT1, T1, TS1, LY1	txtt	bx, (b)	n, (it), sl, sc	<b>TX TYPEWRITER BOLD SMALL CAPS</b>
<i>TX Math</i>				
OML	txmi	m, bx	it	<i>TX Math.</i> $\alpha \Omega$
OMS	txms	m,bx	n	$\mathcal{T}\mathcal{X}$ $\lfloor\rfloor \lceil\lceil \sqcup\sqcap \sqcap\sqcap \mathcal{M}\mathcal{H}\langle\rangle \ominus \otimes$
U	txsyA, txsyB	m, bx	n	$\gtrless \checkmark \triangleleft \Delta \sim \pitchfork \lambda \exists \Leftarrow \Leftarrow \Leftarrow \Leftarrow$

Table 7.14: Classification of the TX font families

Note that the LuxiMono fonts are supported only in the T1 encoding (see the use of `fontenc` in the examples). The subset of the `textcomp` symbols typically found in PostScript fonts is available—namely, those declared when loading `textcomp` with the option `safe`. However, since the euro symbol is available, it is best to load that package with the option `euro`.<sup>1</sup>

### 7.7.5 txfonts—Alternative support for Times Roman

With the `mathptmx` package, the PSNFSS bundle supports Times Roman as a document font for both text and math, primarily using Times Italic and the Adobe Symbol font for math characters (see Section 7.6.2). In 2000, Young Ryu released his own set of virtual fonts together with accompanying Type 1 fonts to provide math support for documents using Times Roman as the document font.

The extra fonts cover glyphs typically missing in PostScript fonts—for example, a full set of `textcomp` symbols, the full range of math symbols as implemented by AMS fonts (see Chapter 8), and others. Thus, these fonts are far more complete than their counterparts in the standard L<sup>A</sup>T<sub>E</sub>X PSNFSS package.

<sup>1</sup>To see the euro symbol in the various TeX fonts, it is important to have the newest version of the file `8r.enc` installed.

The fonts are accessed by loading the package `txfonts` in the preamble. When the package is loaded, it sets up Times Roman as the main document font and Adobe Helvetica (scaled down to 95%) as the sans serif font. For the typewriter font a monospaced font developed by the package author is used.

Compare the next example with Example 7-6-2 on page 376. The extra line at the end shows a few symbols from `textcomp` that are unavailable with `mathptmx`.

An example showing a trigonometric function:

$$\sin \frac{\alpha}{2} = \pm \sqrt{\frac{1 - \cos \alpha}{2}}$$

The script looks like this: *ABC*.

7-7-7 From `textcomp`: \$ € ★ ☀ + ...

```
\usepackage{txfonts}\usepackage[full]{textcomp}
An example showing a trigonometric function:
\[\sin \frac{\alpha}{2} = \pm \sqrt{\frac{1 - \cos \alpha}{2}}\]
The script looks like this: $\mathcal{ABC}$.\ \
From textcomp: \textdollaroldstyle\ \texteuro\ \
\textborn\ \textmarried\ \textdied\ \ldots
```

The TX fonts (see Table 7.14 on the facing page) have support for the text font encodings OT1, T1, TS1, and LY1. However, the OT1 encoding is not faithfully implemented: some of the deficiencies in this encoding are (incorrectly) circumvented (for example, the fact that only either \$ or £ is available in “real” OT1 fonts). Fixing these deficiencies means that the new definitions will not work with any other OT1-encoded font. As OT1 is still the default encoding with L<sup>A</sup>T<sub>E</sub>X this change can lead to serious problems.<sup>1</sup>

The following example illustrates the use of the problematic definitions. In OT1-encoded Computer Modern, all glyphs are wrong: the \$ turns into a £ sign and all others are simply dropped. On the other hand, there is no problem with T1, so one should always combine `txfonts` with `\usepackage[T1]{fontenc}`.<sup>2</sup>

```
\usepackage{txfonts}
\fontencoding{OT1}\selectfont % LaTeX default encoding!
\l.\textdollar.\textsterling.\r{A}.\r{a}\hfill (txfont)

\fontfamily{cmtt}\itshape % italic CM Typewriter
\l.\textdollar.\textsterling.\r{A}.\r{a}\hfill
(all errors)

\l.\$.£.Å.å (txfont)
...£... (all errors) \fontencoding{T1}\selectfont % ... in T1
\l.\textdollar.\textsterling.\r{A}.\r{a}\hfill (okay)
```

In addition, a more serious problem with the current release of the fonts is that the glyph side-bearings in math are extremely tight, up to the point that

<sup>1</sup>Strictly speaking, the fonts implement a new encoding that is similar to OT1 but not identical—and incorrectly call it OT1.

<sup>2</sup>As discussed in Section 7.3.5, T1 is the preferred encoding in any case.

characters actually touch if used in subscripts or superscripts.

A problematic example:

$$t[u_1, \dots, u_n] = \sum_{k=1}^n \binom{n-1}{k-1} (1-t)^{n-k} t^{k-1} u_k$$

```
\usepackage{amsmath,txfnt}
A problematic example:
\[\sum_{k=1}^n \binom{n-1}{k-1} (1-t)^{n-k} t^{k-1} u_k]
```

It is possible that these problems will be fixed in a future release of the fonts.  
For comparison, we show the previous example using `mathptmx`:

A problematic example:

$$t[u_1, \dots, u_n] = \sum_{k=1}^n \binom{n-1}{k-1} (1-t)^{n-k} t^{k-1} u_k$$

```
\usepackage{amsmath,mathptmx}
A problematic example:
\[\sum_{k=1}^n \binom{n-1}{k-1} (1-t)^{n-k} t^{k-1} u_k]
```

To summarize, the TX font families currently show some deficiencies in math typesetting, but offer a large range of symbols for math and text, including all symbols from the AMS math fonts and a full implementation of the `textcomp` symbols. If the focus is on having many symbols available in Type 1 fonts, such as when producing PDF documents, the fonts provide an interesting alternative.

### 7.7.6 pxfonts—Alternative support for Palatino

Young Ryu also developed a set of virtual fonts together with accompanying Type 1 fonts to provide math support for documents using Adobe Palatino as the main document font. The PX fonts (see Table 7.15 on the next page) are set up by loading the `pxfonts` package. For sans serif and typewriter fonts the package uses fonts from the `txfnt` set-up (scaled-down Helvetica and TX typewriter), so both font sets need to be installed.

The next example uses the same text as Example 7-7-7 on the preceding page but this time loads the `pxfonts` package.

An example showing a trigonometric function:

$$\sin \frac{\alpha}{2} = \pm \sqrt{\frac{1 - \cos \alpha}{2}}$$

The script looks like this: `\sin \frac{\alpha}{2}`.  
From `textcomp`: `$€★⊗+...`

```
\usepackage{pxfonts}\usepackage[full]{textcomp}
An example showing a trigonometric function:
\[\sin \frac{\alpha}{2} = \pm \sqrt{\frac{1 - \cos \alpha}{2}}]
The script looks like this: $€★⊗+...
From textcomp: \textdollaroldstyle\ \texteuro\
\textborn\ \textmarried\ \textdied\ \ldots
```

Since the PX fonts have the same font layout as the TX fonts, the OT1 problems shown in Example 7-7-8 on the previous page also arise with this family.

Encoding	Family	Series	Shape(s)	Example of the Typeface
<i>PX Roman</i>				
OT1, T1, TS1, LY1	pxr	m	n, it, sl, sc	PX Roman normal
OT1, T1, TS1, LY1	pxr	bx, (b)	n, it, sl, sc	<b>PX Roman bold italic</b>
<i>PX Math</i>				
OML	pxmi	m, bx	it	<i>PX Math.</i> $\alpha \Omega$
OMS	pxms	m,bx	n	$\mathcal{P}\mathcal{X} \cup \cap \setminus \exists \forall \neg \otimes \otimes$
U	pxsy, pxsyb	m, bx	n	$\neq \Delta \sim \pitchfork \lambda \in \Leftarrow \Rightarrow$

Table 7.15: Classification of the PX font families

The typesetting in math is still very tight but not always so noticeable as in the TX fonts. Below, the Example 7-7-9 on the facing page is repeated for comparison.

A problematic example:

$$t[u_1, \dots, u_n] = \sum_{k=1}^n \binom{n-1}{k-1} (1-t)^{n-k} t^{k-1} u_k$$

7-7-12

```
\usepackage{amsmath,pxfonts}
A problematic example:
\[
t[u_1, \dots, u_n] = \sum_{k=1}^n \binom{n-1}{k-1} (1-t)^{n-k} t^{k-1} u_k
]
```

### 7.7.7 The Fourier-GUTenberg fonts

Adobe donated four fonts from the Utopia family (Utopia Regular, Utopia Italic, Utopia Bold, and Utopia BoldItalic) to the X-Consortium. Though not free software, these typefaces are available free of charge and basic support for them is available through the PSNFSS bundle (see Section 7.6).

The Fourier-GUTenberg bundle developed by Michel Bovani is a typesetting environment based on the Utopia typeface but complemented with the characters missing to provide a full T1 encoding (OT1 is *not* supported), a suitable set of math symbols, Greek sloped and upright letters, and a matching calligraphic and blackboard bold alphabet so that whole documents can be prepared without using any other typefaces. The font encoding is shown in Table 7.16 on the next page; a complete example page is given in Figure 8.4 on page 515.

An example showing a trigonometric function:

$$\sin \frac{\alpha}{2} = \pm \sqrt{\frac{1 - \cos \alpha}{2}}$$

7-7-13 The alphabets are  $\mathcal{ABC}$  and  $\mathbb{ABC}$ .

```
\usepackage{fourier}
An example showing a trigonometric function:
\[
\sin \frac{\alpha}{2} = \pm \sqrt{\frac{1 - \cos \alpha}{2}}
\]
The alphabets are $\mathcal{ABC}$ and $\mathbb{ABC}$.
```

Family	Series	Shape(s)	Examples
<i>Utopia</i> (T1, TS1)			
futs	m, b	(bx) n, sl, it, (sc)	Utopia-Regular <b><i>Utopia-BoldItalic</i></b>
<i>Fourier math letters</i> (FML)			
futm, futmi	m	it	$\Delta\Theta\Lambda \alpha\beta\gamma abcdef \Delta\Theta\Lambda \alpha\beta\gamma$
<i>Fourier math symbols</i> (FMS)			
futm	m	n	$\not\in C \  ABCDEFGHIJKLMNOP$

Table 7.16: Classification of the Fourier-GUTenberg font families

The `fourier` package supports typesetting mathematics “à la French”, with Greek letters and Roman uppercase letters in upright style, by specifying the option `upright`. Compare the next example to the output in Example 8-4-1 on page 490.

```
\usepackage{amsmath}\usepackage[upright]{fourier}
0 \leftarrow F \times \Delta(n-1) \xrightarrow[\zeta]{\partial_0 \alpha(b)} E^{\partial_0 b} \quad \begin{array}{l} \text{\small \#14} \\ \text{\small 7-7-14} \end{array}
```

If you require extended math support from the `amsmath` package as in the previous example, load this package first, so that certain aspects of the math formatting tuned for typesetting in `Utopia` will not be overwritten. For the same reason, you should load `amssymb` first, though you will find that `fourier` already contains several symbols normally available only with `amssymb`. In fact, the `fourier` package offers a small set of mathematical symbols not found elsewhere (e.g., certain integral signs, some delimiters, and other symbols). Some are shown in the next example.

```
\usepackage{fourier}
\setlength\delimitershortfall{-2pt} % make delimiters grow
\left\llbracket \right\rrbracket \left\{\right\} \left(\right) \left.\right| \left.\right. \quad \begin{array}{l} \text{\small \#15} \\ \text{\small 7-7-15} \end{array}
```

Upright and slanted variants of the Greek letters can be used together in a single document by prefixing the command names with `other`. For example:

```
\usepackage[upright]{fourier}
\Omega_\beta \neq \Omega_\beta \quad \begin{array}{l} \text{\small \#16} \\ \text{\small 7-7-16} \end{array}
```

Without the `upright` option (or with the default option `sloped`), the letters are sloped according conventional typesetting of mathematics—that is, upright

Family	Series	Shape(s)	PostScript Font Names and Examples
<i>URW Antiqua Condensed (OT1, T1, TS1)</i>			
uaq	(m), mc	n, sl, (it), sc	URWAntiquaT-RegularCondensed
<i>URW Grotesk Bold (OT1, T1, TS1)</i>			
ugq	b, (bx), (m)	n, sl, (it), sc	<b>URWGroteskT-Bold</b>

Table 7.17: Classification of the URW Antiqua and Grotesk fonts

uppercase Greek and everything else slanted. The meaning of the `\other...` commands is swapped, accordingly.

7.7-17  $\Omega_\beta \neq \Omega_\beta$  `\usepackage[sloped]{fourier}`  
 $\Omega_\beta \neq \Omega_\beta$  `\[ \Omega_\beta \neq \Omega_\beta ]`

In the current implementation `fourier` does not support `\boldmath`. Consequently, using the `bm` package will most often lead to “poor man’s bold”; see Section 8.8.2.

To complement the freely available fonts Adobe offers a commercial expert set containing old-style digits, real small capitals, a semibold series, and an extra bold series. To support these typefaces, the `fourier` package offers additional options: `expert` provides `\textsb` and `\textblack` to select the extra font series and arranges to use real small capitals with `\textsc`. The `oldstyle` option provides the same support but additionally uses old-style numerals in text (`\lining` allows you to refer to lining numerals in that case). Finally, the `fulloldstyle` works like `oldstyle` but additionally arranges for old-style numerals to be used in formulas.

*Support for the commercial expert fonts*

### 7.7.8 The URW Antiqua and Grotesk fonts

The German company URW made two PostScript fonts, URW Antiqua Condensed and URW Grotesk Bold, freely available. L<sup>A</sup>T<sub>E</sub>X support in the form of virtual fonts and .fd files is available. They are accessed using the classification given in Table 7.17. The sample below was typeset by specifying `\fontfamily{uaq}` `\selectfont`.

*URW’s Antiqua Condensed  
10pt/12pt (uaq)*

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 phoenix’s official rôle in fluffy soufflés?

As its name indicates, the URW Grotesk Bold font is available only in a bold series (although within L<sup>A</sup>T<sub>E</sub>X selecting a medium series is supported for convenience

*URW’s Grotesk Bold  
10pt/12pt (ugq)*

but refers to the same bold font). As such, it is not suitable for general running text. Potential applications include headings and other display material.

**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 souf-  
flés?**

### 7.7.9 yfonts—Typesetting with Old German fonts

There exists a set of beautiful fonts for typesetting in Gothic, Schwabacher, and Fraktur designed in METAFONT<sup>1</sup> after traditional typefaces by Yannis Haralambous [62]. These days Type 1 versions of the fonts are available as well: To use the fonts, load the `yfonts` package written by Walter Schmidt. This package internally defines some local encodings that reflect the special features found in the fonts and integrates them fully with L<sup>A</sup>T<sub>E</sub>X's font management.

The commands `\gothfamily`, `\swabfamily`, and `\frakfamily` switch to Gothic, Schwabacher, and Fraktur, respectively. If one wants to typeset a whole document in such a typeface, the corresponding command should be used directly *after* `\begin{document}`. Because of the nonstandard encodings of the fonts, redefining the document defaults (e.g., `\familydefault`) is not possible. In addition to the font switches, the usual `\text...` commands for typesetting short fragments are provided.

The package provides `\textgoth`, also called `\textswab`, and `\textfrak` typefaces, also generally known as „gebrochene Schriften“.

`\usepackage{yfonts}\usepackage[document]{ragged2e}`  
The package provides `\textgoth{Gotisch}`, also called `\textswab{Schwabacher}`, and `\textfrak{Fraktur}` typefaces, also generally known as `\textfrak{'ge\ -bro\ -che\ -ne Schriften'}`.

7-7-18

The fonts are available in the usual L<sup>A</sup>T<sub>E</sub>X sizes starting from 10pt, so that size-changing commands (e.g., `\normalsize` and larger) will work. There are, however, no further font series or shapes, so commands like `\emph`, `\textit`, and `\textbf` have no effect other than producing a warning. Following historical practice you can use Schwabacher to emphasize something inside text typeset in Fraktur.

For accents one can use the standard L<sup>A</sup>T<sub>E</sub>X representations (e.g., `\"a` for ä). To facilitate input, the fonts also contain ligatures that represent umlauts (e.g., "a). In Fraktur and Schwabacher there also exist alternate umlauts, which can be accessed with `*a` and similar ligatures. If the `yfonts` package is loaded with the option `varumlaut`, then `\"` produces the variant glyphs automatically.

<sup>1</sup>Compiling the fonts from the METAFONT sources sometimes produces error messages, but generally produces usable fonts when METAFONT is directed to ignore them. The collection also contains a font with baroque initials.

All three fonts contain a glyph for the “short s”, accessed through the ligature `s:`, and “sharp s”, accessed by `\ss`, or through the ligature `sz` or “`s`.

<span style="border: 1px solid black; padding: 2px;">7-7-19</span> <b>Fraktur:</b> ä ë ü ö å é ú ó þ ß ð viz. § <b>Swab:</b> ä ë ü ö å é ú ó þ ß ð viz. § <b>Gothic:</b> ä ë ü ö (unavail) þ ð viz. §	<pre>\usepackage{yfonts} \Large \frakfamily Fraktur: "a "e "u "o \hfil *a *e *u *o \hfil sz \hfil s viz.\ s: \par\swabfamily Swab: "a "e "u "o \hfil *a *e *u *o \hfil sz \hfil s viz.\ s: \par\gothfamily Gothic: "a "e "u "o \hfil (unavail) \hfil sz \hfil s viz.\ s:</pre>
--	--

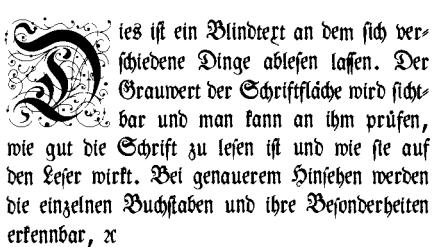
The font selected with `\gothfamily` is not a copy of Gutenberg’s font used for his Bible (which had 288 glyphs altogether), but it follows Gutenberg’s guidelines on lowercase characters and implements as many ligatures as can be fit into a 7-bit font. For this reason many standard ASCII symbols are unavailable in this font.

The two other fonts also implement only a subset of visible ASCII. Problematic are the semicolon (which is missing in Schwabacher) and the characters `+, =, ‘, ’, [, ], /, *, @, &, and %` (which are either missing or produce wrong or nonmatching shapes). Their omission is seldom a problem since typically they are not needed in documents using such fonts, but one needs to be aware that no warning or error message is issued if they are used—the only indication is missing or wrong glyphs in the printed output!

<span style="border: 1px solid black; padding: 2px;">7-7-20</span> <b>Symbols:</b> + = ‘ [ ] / * \$ % & ; @ <b>Fraktur problems:</b> + = ‘ [ ] / * \$ % & ; <b>Swab problems:</b> + = ‘ [ ] / * \$ % <b>Gothic problems:</b> §§ tt s ä ö ü ü \$ §§ ;	<pre>\usepackage{yfonts} \newcommand\test{+ = ‘ [ ] / * \\$ \% \&amp; ; @} Symbols: \ttfamily \test \par \frakfamily Fraktur problems:: \test \par \swabfamily Swab problems:: \test \par \gothfamily Gothic problems:: \test</pre>
--	---

The default line spacing of the standard classes is too large for the Old German fonts. For this reason the package implements the `\fraklines` command, which selects a suitable `\baselineskip` for Fraktur or Schwabacher. It must be repeated after every size-changing command.

The font collection also contains a font with decorative initials, as shown in the next example.

<span style="border: 1px solid black; padding: 2px;">7-7-21</span> 	<pre>\usepackage[german]{babel} \usepackage{color} \usepackage{varumlaut}{yfonts}  \frakfamily\fraklines \yinipar{\color{blue}D}ies: ist ein Blindtext an dem sich verschiedene Dinge ablesen lassen. Der Grauwert der Schriftfläche wird sichtbar und man kann an ihm pr\ "ufen, wie gut die Schrift zu lesen ist und wie sie auf den Leser wirkt. Bei genauerem Hinsehen werden die einzelnen Buchstaben und ihre Besonderheiten erkennbar, x</pre>
---	---

The command `\yinipar` used above starts a new paragraph without indentation, producing a baroque dropped initial. For this command to work, a full paragraph (up to and including the next blank line or `\par`) must be typeset using `\fraklines`. Otherwise, the space left for the initial will be either too large or too small.

As an alternative, you can access these initials with the `\textinit` command or the font switch `\initfamily`, in which case initials aligned at the baseline are produced. The example also used the command `\etc`, which produces a once-popular symbol for "etc."; it is available in Fraktur only.

The font collection contains a second Fraktur font that has slightly wider glyphs with at the same time slightly thinner stems. It can be selected by redefining `\frakdefault` as shown in the next example. When compared to Example 7-7-21, the difference in running length can be clearly observed, resulting in an overfull box on the third line.

```
\usepackage[german]{babel}    \usepackage{color}
\usepackage[varumlaut]{yfonts}
\renewcommand\frakdefault{ysmfrak}
```

 Dies ist ein Blindtext an dem sich verschiedene Dinge ablesen lassen. Der Grauwert der Schriftfläche wird sichtbar und man kann an ihm prüfen, wie gut die Schrift zu lesen ist und wie sie auf den Leser wirkt. Bei genauerem Hinsehen

```
\frakfamily\fraklines
\yinipart{\color{blue}D}ies: ist ein Blindtext an dem sich verschiedene Dinge ablesen lassen. Der Grauwert der Schriftfl\"ache wird sichtbar und man kann an ihm pr\"ufen, wie gut die Schrift zu lesen ist und wie sie auf den Leser wirkt. Bei genauerem Hinsehen
```

7-7-22

### 7.7.10 euler, eulervm—Accessing the Euler fonts

As mentioned earlier, Hermann Zapf designed a beautiful set of fonts for typesetting mathematics—upright characters with a handwritten flavor—named after the famous mathematician Leonhard Euler [99]. These fonts can be accessed as (math) alphabets of their own, or you can generally modify the math font set-up, thus making L<sup>A</sup>T<sub>E</sub>X use Euler math fonts (rather than Computer Modern) by default.

The Euler fonts contain three math alphabets: `SCRIPT`, `Euler Fraktur`, and `Euler Roman`.<sup>1</sup> The script alphabet can be used via the `eucal` package, which makes this math alphabet available under the name `\mathcal` (obsolete alternate name `\EuScript`). If the package is loaded with the `mathscr` option, the math alphabet becomes available through the command `\mathscr`, with `\mathcal` retaining its original definition.

To access Euler Fraktur in formulas, you use the package `eufrak`, which defines the math alphabet `\mathfrak` (obsolete alternate name `\EuFrak`). There is no particular package to access the Euler Roman alphabet separately. The next ex-

<sup>1</sup>None of these alphabets is suitable for typesetting text as the individual glyphs have sidebearings specially tailored for use in math formulas.

Family	Series	Shape(s)	Example of the Typeface
<i>Euler Roman (U)</i>			
eur	m	n	Euler Roman medium
eur	b	n	<b>Euler Roman bold</b>
<i>Euler Script (U)</i>			
eus	m	n	<i>EULER SCRIPT</i>
<i>Euler Fraktur (U)</i>			
euf	m	n	<i>Euler Fraktur</i>
<i>Euler Extension (U)</i>			
euex	m	n	$\sum \prod \int^{\infty}$

Table 7.18: Classification of the Euler math font families

ample shows Computer Modern Calligraphic, Euler Script, and Euler Fraktur side by side.

7-7-23 
$$\mathcal{A} \neq \sum_{k < n} \mathcal{A}_k \neq \mathfrak{A} \quad \begin{aligned} \backslash\usepackage{mathscr}\{eucal\} \backslash\usepackage{eufrak} \\ \backslash[ \backslash\mathcal{A} ] \neq \backslash\sum_{k < n} \backslash\mathscr{A}_k \neq \backslash\mathfrak{A} \end{aligned}$$

The NFSS classification for the fonts in these families is shown in Table 7.18. The fonts in the current distribution of the Euler math families are available only in encoding schemes that differ from all other encoding schemes for mathematics. For this reason, the fonts are all assigned the encoding U (unknown).

The uncommon encoding makes it difficult to simply substitute the Euler math alphabets for the default CM math fonts. Yet the *euler* package, written by Frank Jensen, went exactly this way, redeclaring most of L<sup>A</sup>T<sub>E</sub>X's math font set-up. In conjunction with the package *beton*, which sets up Concrete as the default text font family, it simulates the typography of Knuth's book *Concrete Mathematics* [59], as shown in Example 7-7-2.

One of the problems with extensive reencoding in macro packages, as done by the *euler* package, is that it is likely to break other packages that assume certain symbols in slot positions, as defined by more established font encodings. The *eulervm* package developed by Walter Schmidt attempts to avoid this problem by providing reencoded virtual fonts that follow as much as possible the standard math encodings OML, OMS, and OMX.

The *eulervm* package sets up a *\mathnormal* alphabet, which is based mainly on Euler Roman, and a *\mathcal* alphabet, which is based on Euler Script. It does not provide immediate support for the Euler Fraktur alphabet—to access this math alphabet one needs to additionally load the *eufrak* package. Also, the

*Virtual Euler fonts*

math symbols are taken from the Euler fonts, with a few exceptions coming from the Computer Modern math fonts. Compare the next example to Example 7-7-23 on the previous page and you will see that `\mathcal` has changed and that `\sum` and the indices are different, as they are now taken from the Euler fonts.

$$\mathcal{A} \neq \sum_{k < n} A_k \neq \mathfrak{A}$$

```
\usepackage{eulervm,eufrak}
[ \mathcal{A} \neq \sum_{k < n} A_k \neq \mathfrak{A} ]
```

7-7-24

The option `small` causes `eulervm` to load all Euler fonts at 95% of their normal size, thereby enabling them to blend better with some document fonts (e.g., Adobe Minion). This option also affects the Euler Fraktur fonts if they are loaded with `eufrak` and the AMS symbol fonts.

Neither the standard `\hbar` command nor `\hslash` (from the `amssymb` package) is really usable with the Euler fonts if it is used without modification (i.e., with `euler`), because `\hslash` uses a Computer Modern style "h" and `\hbar` gets the slash in a strange position.

$$\hbar \neq \mathfrak{h}$$

```
\usepackage{amssymb,euler}
[ \hslash \neq \hbar ]
```

7-7-25

This issue restricts the usage of the `euler` package somewhat for physics and related fields. The `eulervm` package resolves this problem (partially) by providing a properly slashed "h" glyph built using the possibilities offered by the virtual font mechanism ([91] explains the concepts). It does, however, provide only a slashed version (`\hslash`); if `\hbar` is used, a warning is issued and the slashed glyph is used nevertheless.

$$\hbar \equiv \mathfrak{h}$$

```
\usepackage{eulervm}
[ \hslash \equiv \hbar ]
```

7-7-26

The functionality provided by the `exscale` package is automatically available. See Section 7.5.5 on page 368 for details.

In typical font set-ups the same digits are used in text and math formulas. The Euler fonts contain a set of digits that have a distinctive look and thus make digits in text and math look noticeably different.

By default, the digits of the main document font are used in formulas as well. To switch to the digits from Euler Roman, one has to explicitly request them by specifying the option `euler-digits`. It then becomes very important to distinguish between a number in a mathematical or a textual context. For example, one must watch out for omitted \$ signs, as in the first line of the next example.

The value can be 1, 2, or -1 (wrong!)  
The value can be 1, 2, or -1 (right!)

```
\usepackage{ccffonts}
\usepackage[euler-digits]{eulervm}
The value can be 1, 2, or -$-1$ (wrong!)\par
The value can be $1$, $2$, or $-1$ (right!)
```

7-7-27

Normally, the math accent `\hat` is taken from the main document font, which might not be a good choice when text and math fonts are noticeably different. With the option `euler-hat-accent`, an alternative version from the Euler fonts is used instead. In the example we mimic that option and define the alternate accent under the name `\varhat` manually to enable comparison of the two (neither looks really perfect).

7-7-28  $\hat{x} \neq \hat{\hat{x}}$  and  $\hat{\mathfrak{K}} \neq \hat{\hat{\mathfrak{K}}}$

```
\usepackage{palatino,eulervm,eufrak}
\DeclareMathAccent\varhat{\mathalpha}{symbols}{222}
\Large $ \hat{x} \neq \varhat{x} $ and
$ \hat{\mathfrak{K}} \neq \varhat{\mathfrak{K}} $
```

It is usually best to load the `eulervm` package after all the document fonts have been defined, because `eulervm` defines the math alphabets (e.g., `\mathsf{f}`) by evaluating the document's default information that is current when the package is loaded. In the example below, the loading order is absolutely essential because the `ccfonts` package also tries to set up the math fonts and thus the one that comes last wins.

In the book *Concrete Mathematics* [59], where Euler and Concrete fonts were first used together, one can see that slanted  $\leq$  and  $\geq$  signs were once part of the Euler Math fonts. Somewhere along the way these two symbols got lost, though traces of their existence can be found in [92] and in macros that Donald Knuth developed for producing the book. With the help of the virtual font mechanism, Walter Schmidt brought them back in the `eulervm` package; compare the next example to Example 7-7-2 on page 384, which shows the straight  $\leq$  sign.

Concrete Roman blends well with Euler Math,  
as can be seen with

7-7-29 
$$\sum_{0 \leq k < n} k = \frac{n(n-1)}{2}$$

```
\usepackage{ccfonts,amssymb}
\usepackage[euler-digits]{eulervm}
Concrete Roman blends well with Euler Math,
as can be seen with
\[ \sum_{0 \leq k < n} k = \frac{n(n-1)}{2} \]
```

## 7.8 The L<sup>A</sup>T<sub>E</sub>X world of symbols

Shortly after T<sub>E</sub>X and METAFONT came into existence, people started to develop new symbol fonts for use with the system. Over time the set of available symbols grew to a considerable number. The *Comprehensive L<sup>A</sup>T<sub>E</sub>X Symbol List* by Scott Pakin [134] lists 2590 symbols<sup>1</sup> and the corresponding L<sup>A</sup>T<sub>E</sub>X commands that produce them. For some symbols the necessary fonts and support packages may have to be obtained (e.g., from a CTAN host; see Appendix C) and installed by the user. They are usually accompanied by installation instructions and general documentation.

<sup>1</sup>Counted spring 2003.

	'0	'1	'2	'3	'4	'5	'6	'7	
'00x	△	◀	≤	▷	▷	∴	∅	☎	"0x
'01x	✓	◊	▲	♪	♪	↓	◦	♪	"1x
'02x	◀	▶	⚡	∞	∞	⊗	⊗	Τ	"2x
'03x	⊓	♀	♂	⊗	⊕	∞	✗	∅	"3x
'04x	●	○	○	○	○	▷	δ	♀	"4x
'05x	⟨	⟩	^	ˇ	⌣	⦿	☼	⦿	"5x
'06x	ʊ	⊗	□	◊	⊗	□	✖	○	"6x
'07x	○	○	~	~	□	□	≤	≥	"7x
'10x	≈	*	*	◊	◊	*	▽	■	
'11x	▶	□	▷	▲	▼				
'12x	♂	‘	’			ə	σ	σ	
'13x	ȝ	ȝ	ð	ȝ	ȝ	ȝ	ȝ	ȝ	
'14x	ȝ	ȝ	ȝ	ȝ	ȝ	ȝ	ȝ	ȝ	
'15x	%	þ	Þ	ð	ð	þ	þ	þ	
'16x	☒	☒	∫	ʃʃ	ʃʃʃ	ʃ	ʃʃ	ʃ	
'17x	ʃʃ	ʃʃʃ	ʃ	ʃʃ	!	□	□	▫	
	"8	"9	"A	"B	"C	"D	"E	"F	

Table 7.19: Glyphs in the *wasy* fonts

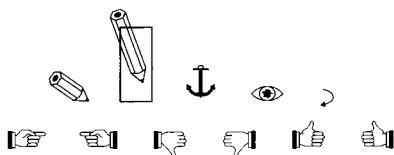
The fonts and packages described in this section form only a subset of what is available. If you cannot find a symbol here, the 70 pages of [134] are a valuable resource for locating what you need. We start by looking at a number of dingbat fonts, some of which contain quite unusual symbols. This examination is followed by an introduction to the TIPA system, which provides support for phonetic symbols. The section finishes with a discussion of ways to obtain a single (though in Europe not unimportant) symbol: the euro. Being a relatively new addition to the symbol world, it is missing in many fonts and thus needs alternative ways to produce it. All packages and fonts listed in this section and in [134] are freely available.

### 7.8.1 dingbat—A selection of hands

The *dingbat* package written by Scott Pakin provides access to two symbol fonts developed by Arthur Keller (*ark10.mf*) and Doug Henderson (*dingbat.mf*). The

package makes a set of hands and a few other symbols available; the example shows most of them. Note that the `\largepencil` glyph is bigger than the space it officially occupies (shown by the `\frame` drawn around it).

7-8-1



```
\usepackage{dingbat}
\smallpencil \quad \frame{\largepencil} \quad
\anchor \quad \eye \quad \carriagereturn \\[5pt]
\leftpointright \quad \rightpointleft \quad
\leftthumbsdown \quad \rightthumbsdown \quad
\leftthumbsup \quad \rightthumbsup
```

These fonts exist only as a METAFONT implementation, so they are not really suitable when intending to produce PDF (e.g., with pdfL<sup>A</sup>T<sub>E</sub>X).

### 7.8.2 wasysym—Waldi's symbol font

The `wasy` package developed by Axel Kielhorn provides access to the `wasy` fonts designed by Roland Waldi. These fonts first appeared in 1989 and are nowadays available both in METAFONT source and Type 1 outlines. They cover a wide range of symbols from different areas, including astronomical and astrological symbols, APL, musical notes, circles, and polygons and stars (see Table 7.19 on the facing page). The `wasy` package defines command names like `\phone` to access each glyph. Alternatively, if you want only a few glyphs from the font, you can use the `pifont` interface and access the symbols directly under the name `wasy`.

7-8-2

using `wasy`  
 using `pifont`

```
\usepackage{wasy,pifont}
\phone\ using \texttt{[wasy]}\ \par
\Pisymbol{wasy}{7}\ using \texttt{[pifont]}
```

### 7.8.3 marvosym—Interface to the MarVoSym font

The MarVoSym font designed by Martin Vogel is another Pi font containing symbols from various areas including quite uncommon ones, such as laundry signs (in case you are doing your own laundry lists ☺), astronomy and astrology symbols, and many others.

The L<sup>A</sup>T<sub>E</sub>X support package `marvosym` was written by Thomas Henlich, who also converted the font from TrueType format to PostScript Type 1. This package defines command names for all symbols, some of which are listed in the next example; the full set is given in `marvodata.pdf` accompanying the distribution.

7-8-3

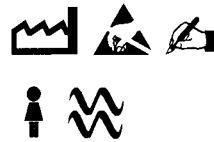


```
\usepackage{marvosym}
\Large \Mobilefone\ \Faxmachine\ \Fixedbearing\ \Lineload\
\Coffee\ \Football\ \AtForty\ \IroningII\ \Cancer\ \Virgo\
\RewindToStart\ \ForwardToIndex\ \ComputerMouse\ \Keyboard\
\Female\ \FEMALE\ \Smiley\ \Frowny\ \Yingyang\ \Bicycle
```

	'0	'1	'2	'3	'4	'5	'6	'7	
'04x		⌚	▬	Δ	Δ	Δ	⌚	⌚	"2x
'05x	(	)	×	+	,	-	.	/	
'06x	ⓧ	1	2	3	4	5	6	7	"3x
'07x	8	9	→	⇒	≤	≥	≥	↔	
'10x	@	∅	✉	€€	€	ƒ	₩	-	"4x
'11x	⌚	〽	〽	☕	₩₩₩		□	□	
'12x	-	✖		✖	✖	✖	✖	✖	"5x
'13x	✗	⌚	⌚	/	✗	≡	≡	/	
'14x	⌚⌚	✳	✳	€	€	€	✉	-	"6x
'15x	⚠	█	☢	☒	↓	⚠	⌚⌚	⌚⌚	
'16x	-	✖		✖	FAX	FAX	⌚	⌚	"7x
'17x	ⓘ	ⓘ	⚡	○	♂	♀	♀	♂	
'20x	♂	♀	♂	♂	♀	♀	†	◊	"8x
'21x	†	洗衣	洗衣	洗衣	♡	✉	₩₩₩	□	
'22x	☒	●	❖	█	█	●	-	□	"9x
'23x	□	L	I	O	T	L	I	T	
'24x	φ	β	϶	₩	€	₩	\$	⌚	"Ax
'25x	⊗	⌚	Ⓐ	Ⓐ	Ⓟ	Ⓟ	⌚⌚	⌚⌚	
'26x	洗衣	洗衣	洗衣	□	◀◀	◀	◀	▶	"Bx
'27x	▶	▶▶	▲	▼	▲	▼	Ⓕ	Ⓕ	
'30x	⌚	⌚	⌚	⌚	♂	⌚	⌚	⌚	"Cx
'31x	Ψ	♀	♂	△	△	⌚	⌚⌚	⌚⌚	
'32x	▬▬	₩₩₩	₩₩₩	₩₩₩	₩₩₩	₩₩₩	₩₩₩	₩₩₩	"Dx
'33x	洗衣	洗衣	洗衣	洗衣	洗衣	洗衣	□	□	
'34x	Ϻ	Ϙ	Ϻ	Ϙ	Ϙ	Ϻ	Ϻ	Ϻ	"Ex
'35x	✖	Ϙ	Ϻ	Ϻ	□	□	□	□	
'36x	A	p	□	□	□	□	□	▪	"Fx
'37x	□	□	□	□	□	⌚	⌚⌚	⌚⌚	
	"8	"9	"A	"B	"C	"D	"E	"F	

Table 7.20: Glyphs in the MarVoSym font

Assuming a recent distribution, one can also access the symbols directly by using the glyph chart in Table 7.20 on the preceding page and the pifont interface with the Pi font name being `mvs`. In older distributions the file `umvs.fd` that makes this method work might be missing, but it can be easily added as shown below.

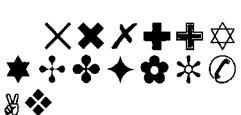


7-8-4

```
\begin{filecontents}{umvs.fd}
\DeclareFontFamily{U}{mvs}={}
\DeclareFontShape{U}{mvs}{m}{n}{<-> fmvr8x}{}
\end{filecontents}
\usepackage{pifont}
\Huge \Pisymbol{mvs}{73} \Pisymbol{mvs}{74} \Pisymbol{mvs}{98}
\Pisymbol{mvs}{120} \Pisymbol{mvs}{121} \Pisymbol{mvs}{234}
```

#### 7.8.4 bbdङ—A METAFONT alternative to Zapf Dingbats

For those who cannot use PostScript Type 1 fonts, Karel Horak designed a font with METAFONT containing most of the symbols from Hermann Zapf's dingbat font. The package `bbding` by Peter Møller Neergaard provides an interface that defines command names for each symbol (using a naming convention modeled after WordPerfect's names for accessing the Zapf Dingbats font). The complete list can be found in the package documentation, a few examples are given below.



7-8-5

```
\usepackage{bbding}
\XSolid\ \XSolidBold\ \XSolidBrush\ \Plus\ \PlusOutline\ \DavidStar\
\DavidStarSolid\ \JackStar\ \JackStarBold\ \FourStar\ \FiveFlowerPetal\
\SixFlowerOpenCenter\ \PhoneHandset\ \Peace\ \OrnamentDiamondSolid
```

Alternatively, referring to the glyph chart in Table 7.21 on the following page, you can address individual symbols via the pifont interface, by accessing the font under the name `ding` (compare this to Table 7.9 on page 379 showing the original Zapf designs).



7-8-6

```
\usepackage{pifont}
\Pisymbol{ding}{13} \Pisymbol{ding}{15} \Pisymbol{ding}{8}
\Pisymbol{ding}{17} \Pisymbol{ding}{19} \Pisymbol{ding}{9}
```

#### 7.8.5 ifsym—Clocks, clouds, mountains, and other symbols

The `ifsym` package written by Ingo Klöckl provides access to a set of symbol fonts designed in METAFONT. At present they are not available in Type 1 format. Depending on the chosen package option(s), different symbol sets are made available. We show only a small selection here. The full documentation (German only) is provided in the PostScript file `ifsym.ps`, which is part of the distribution. All available symbols are also listed in [134].

	'0	'1	'2	'3	'4	'5	'6	'7	
'00x	✈	✂	✖	✈	✖	✖	✂	✖	'0x
'01x	☎	⌚	☰	✈	✉	👉	👉	👉	
'02x	👉	👉	👉	👉	👉	🐰	👉	👉	
'03x	-pencil	-pencil	-pencil	-pencil	-pencil	↶	↶	↶	
'04x	❖	✓	✓	✗	✗	✗	✚	✚	
'05x	✚	+	+	+	✚	✚	✚	★	
'06x	✡	✚	✚	♣	✚	◆	❖	☆	
'07x	★	☆	★	☆	★	★	★	★	
'10x	★	*	*	*	*	*	*	*	'4x
'11x	*	*	*	*	*	*	*	*	
'12x	✿	✿	✿	✿	✿	*	✿	✿	
'13x	✿	*	*	*	*	*	*	*	
'14x	*	●	○	●	○	○	□	■	
'15x	□	□	□	□	□	□	▲	▼	
'16x	◆	❖	▷	◁			█	▶▶	
'17x	▶	▶	▶	▶					
	'8	'9	'A	'B	'C	'D	'E	'F	

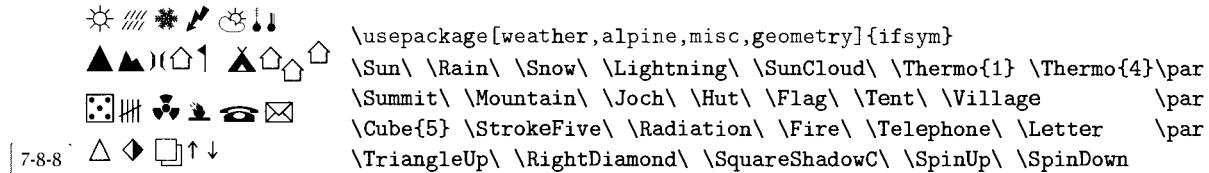
Table 7.21: Glyphs in the METAFONT font bbdng

The option `clock` makes seven clock-related symbols available. It also provides the command `\showclock` to display an analog watch, with the hands showing the correct time. Its two arguments denote the hour (0-11) and minutes (0-59). The minutes displayed are rounded to the nearest 5-minute interval; using a value greater than 11 for the hour makes the symbol disappear without warning. All symbols are available in normal and bold extended series.

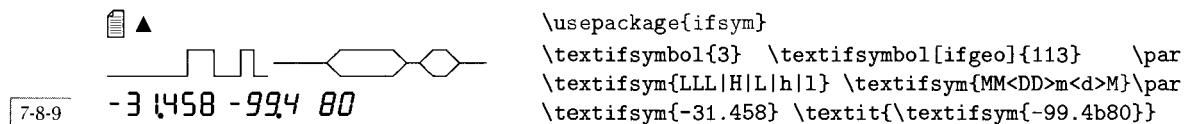
Normal: rounded:   
 Problem:  
 Fixed symbols:   
`\usepackage[clock]{ifsym}`  
`Normal: \showclock{3}{20} rounded: \textbf{\showclock{6}{17}}`  
`\\" Problem: \showclock{16}{35}\\" Fixed symbols: \Taschenuhr{} \StopWatchStart{} \StopWatchEnd{} \Interval{} | 7-8-7`

The option `weather` defines 22 weather symbols, a few of which are shown on the first line of the next example. The `\Thermo` command displays a different thermometer symbol depending on the number in its argument (0-6).

For alpinists and travelers the option `alpine` provides 17 symbols for use in route descriptions or maps. The option `misc` offers a set of unrelated symbols, some of which are also found in other fonts, and the option `geometry` provides commands for 30 geometric shapes, some of which are shown on the fourth line of the example.



The command `\text{ifsymbol}` allows you to access symbols by their slot positions. Its optional argument defines the symbol font to use (default `ifsym`). Glyph charts of all `ifsym` fonts are part of the package documentation. Somewhat more interesting is the command `\text{ifsym}`, which allows you to produce pulse diagrams. It can also be used to display digital digits (where `b` denotes an empty space of the right width).



### 7.8.6 tipa—International Phonetic Alphabet symbols

The TIPA bundle [50] developed by Rei Fukui consists of a set of fonts and a corresponding package to enable typesetting of phonetic symbols with L<sup>A</sup>T<sub>E</sub>X. TIPA contains all the symbols, including diacritics, defined in the 1979, 1989, 1993, and 1996 versions of the International Phonetic Alphabet (IPA). Besides IPA symbols, TIPA contains symbols that are useful for other areas of phonetics and linguistics including the following:

- Symbols used in American phonetics, for example,  $\alpha$ ,  $\epsilon$ ,  $\Omega$ , and  $\lambda$ ;
- Symbols used in the historical study of Indo-European languages, such as  $\beta$ ,  $\psi$ ,  $\chi$ ,  $\zeta$ ,  $\wp$ , and accents such as  $\acute{a}$  and  $\check{e}$ ;
- Symbols used in the phonetic description of languages in East Asia, such as  $\iota$ ,  $\lrcorner$ ,  $\lhd$ ,  $\eta$ ,  $\varepsilon$  (needs option `extra`);
- Diacritics used in *extIPA Symbols for Disordered Speech* and *VoQS (Voice Quality Symbols)*, for example,  $\bar{n}$ ,  $\bar{f}$ , and  $\bar{m}$  (needs option `extra`).

The IPA symbols are encoded in the standard L<sup>A</sup>T<sub>E</sub>X encoding T3, for which the package `tipa` provides additional support macros. The encoding is available for the font families Computer Modern Roman, Sans, and Typewriter (based on the

ASCII	:	;	"		0	1	2	3	4	5	6	7	8	9
TIPA	:	'	'		u	i	A	3	U	e	d	y	ø	ø
ASCII	Ø	A	B	C	D	E	F	G	H	I	J	K	L	M
TIPA	ø	a	b	c	ð	ɛ	ɸ	ɣ	f	i	j	h	k	ŋ
ASCII	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
TIPA	ŋ	ɔ	ɔ	ɸ	r	ʃ	θ	v	v	w	x	y	z	

Table 7.22: TIPA shortcut characters

METAFONT designs for Computer Modern by Donald Knuth), as well as for Times Roman and Helvetica.

Strictly speaking, T3 is not a proper L<sup>A</sup>T<sub>E</sub>X text encoding, as it does not contain the visible ASCII characters in their standard positions. However, one can take the position that phonetic symbols form a language of their own and for this language, the TIPA system provides a highly optimized input interface in which digits and uppercase letters serve as convenient shortcuts (see Table 7.22) to input common phonetic symbols within the argument of \textipa or the environment IPA. All phonetic symbols are also available in long form; for example, to produce a ø one can use \textscha. The following example shows the TIPA system in a Times and Helvetica environment.

```
\usepackage{mathptmx,tipa}
```

```
In linguistics, fœ'netik transcriptions \textprinstress n\textepsilon t\i k transcriptions
```

In linguistics, fœ'netik transcriptions are usually shown in square brackets, e.g., are usually shown in square brackets, e.g., phonetics [fœ'netiks].

```
\textsf{phonetics} \textipa{[f@U"nEtIks]}.
```

7-8-10

TIPA defines \\*, \;, \:, \!, and \| as special macros with which to easily input phonetic symbols that do not have a shortcut input as explained above. *Redefined math commands* In standard L<sup>A</sup>T<sub>E</sub>X all five are already defined for use in math mode, so loading tipa highjacks them for use by linguists. If that is not desirable, the option safe prevents these redefinitions. The long forms then have to be used—for example, the command \textroundcap instead of \|c. The following lines show a few more complicated examples with the output in Computer Modern Roman, Sans, and Typewriter.

```
\usepackage{tipa}
\begin{IPA}
\textrm{N\!o\`{\i}\!\~{o}} \textsf{*A) d\O g, B) k\ae t, C) maus}
\texttt{*k\H{m}t\H{o}m *bhr\H{a}t\H{e}r} \end{IPA} 7-8-11
```

If loaded with the option tone, TIPA provides a \tone command to produce “tone letters”. The command takes one argument consisting of a string of numbers

denoting pitch levels, 1 being the lowest and 5 the highest. Within this range, any combination is allowed and there is no limit on the length of the combination, as exemplified in the last line of the next example, which otherwise shows the usage of \tone to display the four tones of Chinese.

```
7-8-12
    \usepackage[tone]{tipa}
    ˥ma (mother) ˧ma (horse)
    ˨ma (hemp) ˩ma (scold)
    ˥˥˥˥ma (mother) ˧˧˧˧ma (horse) ˥˥˥˥ma (scold)
    ˨˨˨˨ma (hemp) ˩˩˩˩ma (scold)
```

The above examples merely scrape the surface of the possibilities offered by TIPA. To explore it in detail consult the *tipaman* manual, which is part of the TIPA distribution.

### 7.8.7 Typesetting the euro symbol (€)

On January 1, 2002, the euro (€) became the official currency in 12 countries of the European Union.<sup>1</sup> A long time before that event, the European Commission had a logo designed, to be used whenever one refers to the new European currency. The Commission now also encourages the use of symbols that are adjusted to the current font of a document. Meanwhile, most foundries have integrated specially designed euro symbols into their fonts, but there are still many fonts without euro in use. For instance, the PostScript standard fonts, which are hard-wired in most existing laser printers, cannot be assumed to have euro symbols.

The official L<sup>A</sup>T<sub>E</sub>X command to access a euro symbol is \texteuro, which is part of the *textcomp* package. However, many fonts simply do not contain a euro glyph. In such a case *textcomp* attempts to fake the symbol by putting two slashes through an uppercase C (e.g., in Times Roman €).

With popular fonts designed for use with T<sub>E</sub>X, the euro symbol is usually available but, unfortunately, the euro sign designed by Jörg Knappen for the European Computer Modern fonts (i.e., L<sup>A</sup>T<sub>E</sub>X's default font families) is somewhat futuristic and considered acceptable by many people only in the sans serif family:

```
7-8-13
A normal €, an italic €, a bold €, a bold italic €. Compare the
sans serif € and typewriter € all in EC fonts.
```

```
\usepackage{textcomp}
A normal \texteuro{}, \textit{an italic \texteuro{}},
\textbf{a bold \texteuro{}},
\textbf{\itshape a bold italic \texteuro{}}.
Compare the \textsf{sans serif \texteuro{}}
and \texttt{typewriter \texteuro{}} all in EC fonts.
```

The situation is somewhat better with the Computer Modern Bright families. Although produced using the METAFONT designs of the European Computer

<sup>1</sup>More exactly, bank notes and coins were introduced on that day.

Modern fonts, the euro symbol comes out nicely, as nearly all serifs are dropped in these families.

A normal €, a slanted €, a bold €, a bold slanted €. Compare this to the typewriter € all in CM Bright.

```
\usepackage{cmbright, textcomp}
A normal \texteuro{}, \textsl{a slanted \texteuro{}},
\textbf{a bold \texteuro{}}, \textbf{\textit{slshape a bold
slanted \texteuro{}}}. Compare this to the
\texttt{typewriter \texteuro{}} all in CM Bright.
```

7-8-14

But what should be done if the fonts used in the document do not contain the symbol? In that case the solution is to use either separate symbol fonts that provide a generic euro symbol (with a neutral design, that can be combined with many font families) or symbol fonts specially designed to be used with certain text font families. In any event the symbol should be available in several weight (and width) series and sizes so that it can be effectively used in different typesetting situations (e.g., in a heading like the one of the current section).

#### eurosym—euros for L<sup>A</sup>T<sub>E</sub>X

The first set of fonts providing generic euro symbols for use with T<sub>E</sub>X were probably the EuroSym fonts designed by Henrik Theiling. They are available as METAFONT sources as well as PostScript Type 1 outlines and contain the euro symbol designed according to the official construction method. As a nice feature, the fonts contain a picture of the construction method in slot zero. So for those who always wanted to know how the symbol should be designed, the following example is illuminating:



```
\usepackage{eurosym}
\fontsize{40}{40}\usefont{U}{eurosym}{m}{n}\symbol{0}
```

7-8-15

*Regular euros* The eurosym package, which is used to access these fonts, defines the command \euro. By default, this command generates the official symbol to vary with the series and shape attributes of the current document font. See Table 7.23 on the next page for the set of possibilities.

Regular €, a slanted €, a bold €, and a bold italic €.

```
\usepackage{eurosym}
Regular \euro{}, \textsl{a slanted \euro{}},
\textbf{a bold \euro{}}, and
\textbf{\textit{itshape a bold italic \euro{}}}
```

7-8-16

*Poor man's euros* As an alternative, the package offers commands to construct a euro symbol from the letter "C" in the current font by combining it with horizontal bars (which exist in three widths). The next example shows that the results range from unacceptable to more or less adequate, depending on the shape of the "C" and the

Family	Series	Shape(s)	Example of the Typeface
<i>EuroSym by Henrik Theiling (U)</i>			
eurosym	m	n, (it), sl, ol	regular and outline: €, €
eurosym	(b), bx	n, (it), sl, ol	bold extended upright and slanted: €, €

Table 7.23: Classification of the EuroSym font family

chosen bar width. In any case a properly defined euro symbol for a font is preferable and should be used if available.

$\text{€}, \text{€}, \text{€}$ (Times) $\text{€}, \text{€}, \text{€}$ (Helvetica) <span style="border: 1px solid black; padding: 2px;">7-8-17</span> $\text{€}, \text{€}, \text{€}$ (Courier)	$\backslash usepackage\{times,eurosym\}$ $\backslash rmfamily \backslash geneuro, \backslash geneuronarrow, \backslash geneurowide\}$ (Times) $\backslash par$ $\backslash sffamily \backslash geneuro, \backslash geneuronarrow, \backslash geneurowide\}$ (Helvetica) $\backslash par$ $\backslash ttfamily \backslash geneuro, \backslash geneuronarrow, \backslash geneurowide\}$ (Courier)
---	--

With the package options `gen`, `gennarrow`, and `genwide`, one can change the `\euro` command so that it points to `\geneuro`, `\geneuronarrow`, or `\geneurowide`, respectively. In all cases you can access the official euro symbol using the command `\officialeuro`.

Finally, the package offers the convenient command `\EUR` to typeset an amount of money together with the euro symbol separated by a small space.<sup>1</sup> As different countries have different conventions about where to place the currency sign, the package recognizes the options `left` (default) and `right`.

<span style="border: 1px solid black; padding: 2px;">7-8-18</span> Das Buch kostet 19,60€ im Handel.	$\backslash usepackage[right]\{eurosym\}$ Das Buch kostet \EUR{19,60} im Handel.
--	---

Another way to format monetary amounts is provided by the `euro` package, which is documented on page 96.

### The Adobe euro fonts

Adobe also offers a set of Type 1 fonts that contain the euro symbol. This font set contains serifed, sans serif (with a design close to the official logo), and typewriter variants. All are available in upright and italic shapes and in normal and bold weights. To exploit these fonts, one needs a PostScript printer or, more generally, a printer that can render such fonts (e.g., with the help of the `ghostscript` program).

While the fonts can be freely used for printing purposes, Adobe does not allow them to be generally distributed or included in a TeX distribution. For this reason you have to manually download them from the Adobe web site: <ftp://ftp.adobe.com/pub/adobe/type/win/all/eurofont.exe>. This is a self-extracting archive

<sup>1</sup>Some other packages use this command name to denote the euro symbol itself—an unfortunate inconsistency.

for Windows. On Unix platforms the fonts can be extracted from it using the program `unzip`.

After downloading the fonts, one has to rename them to conform to Karl Berry's font naming conventions [19] and, if necessary, get support files for  $\text{\LaTeX}$ , such as `.fd` files, a mapping file for `dvips`, and a package to make them accessible in documents. Depending on the  $\text{\TeX}$  installation (e.g., the  $\text{\TeX}live$  CD), these files might be already available. Otherwise, they can be downloaded from `CTAN:fonts/euro`.

#### eurosans—One way of getting euros from Adobe

Several  $\text{\LaTeX}$  packages are available that provide access to the Adobe euro fonts, each using a different strategy. As its name indicates, the `eurosans` package developed by Walter Schmidt provides only access to Adobe's EuroSans fonts (see Table 7.24 on the next page). The reason being that the serifed variants seldom fit the body fonts of documents, while the more neutral sans serif designs blend well with most typefaces, except for typewriter fonts. As the EuroMono typefaces from Adobe are actually condensed versions of EuroSans, they have been integrated as a condensed series (NFSS classifications `mc`, `bc` and `sbc`) by the package. Weight (medium or boldface), shape (upright or oblique), and width (regular or condensed) vary according to surrounding conditions in the document.

An important aspect of this package (and one absent from other packages), is the ability to scale the fonts by a factor, using the option `scaled`. By default, it scales the fonts down to 95% of their nominal size. If a different scale factor is needed to match the size of the document font, an explicit value can be provided, as seen in the next example.

A regular € symbol,  
an italic €, a bold €, and  
a bold italic €.

A regular € symbol,  
an italic €, a bold €,  
and a bold italic €.

```
\usepackage{lucidabr} \usepackage[scaled=0.97]{eurosans}
A regular \euro{} symbol, \textit{an italic \euro{}},
\textbf{a bold \euro{}}, and \textit{a bold italic \euro{}}.
\par\sffamily
A regular \euro{} symbol, \textit{an italic \euro{}},
\textbf{a bold \euro{}}, and \textit{a bold italic \euro{}}.  
-----
```

7-8-19

The number of produced variations can be reduced (for example, varying the *Restricting variance* font series but always using normal shape) through a redefinition of the `\euro` command.

```
\usepackage{lucidabr} \usepackage[scaled=0.97]{eurosans}
\DeclareRobustCommand{\euro}{\{\fontencoding{U}%
\fontfamily{eurosans}\fontshape{n}\selectfont E\}}
A regular \euro{} symbol, \textit{not an italic \euro{}},
\textbf{a bold \euro{}}, and \textit{not a bold italic \euro{}}.  
-----
```

7-8-20

If there is no requirement for a serifed euro symbol, the `eurosans` package is usually preferable to other solutions, as it provides the most comprehensive set

Family	Series	Shape(s)	PostScript Font Names and Examples
<i>Adobe EuroSans (U)</i>			
eurosans	m	n, it, (sl)	EuroSans-Regular ( <code>zpeurs</code> ), EuroSans-Italic ( <code>zpeuris</code> ) €, €
eurosans	b, (bx)	n, it, (sl)	EuroSans-Bold ( <code>zpeubs</code> ), EuroSans-BoldItalic ( <code>zpeubis</code> ) €, €
eurosans	mc	n, it, (sl)	EuroMono-Regular ( <code>zpeurt</code> ), EuroMono-Italic ( <code>zpeurit</code> ) €, €
eurosans	(sbc), bc	n, it, (sl)	EuroMono-Bold ( <code>zpeubt</code> ), EuroMono-BoldItalic ( <code>zpeubit</code> ) €, €

Table 7.24: Classification of the Adobe euro font families (eurosans classification)

of font series and supports scaling of the fonts. The package documentation also describes how to install the fonts and the support files if necessary.

#### europs—Another way of getting euros from Adobe

A different approach was taken in the `europs` package developed by Jörn Clausen. It provides the command `\EUR` to access the symbols from the Adobe euro fonts. This command selects a different symbol depending on the font attributes of the surrounding text, as can be seen in the next example.

```
7-8-21 \usepackage{array,times,europs}
          \begin{tabular}{lccccc}
            & rm & sf & tt & & \\
            regular & € & € & € & regular & & & & & \\
            italic & € & € & € & \itshape italic & \itshape\EUR & \itshape\EUR & \itshape\EUR & \\
            bold italic & € & € & € & \bfseries\itshape bold italic & \bfseries\itshape\EUR & \\
            (body font) & C & C & C & \bfseries\itshape\EUR & \bfseries\itshape\EUR & \\
                           & (body font) & & & & & & \\
                           & & & & & & & \\
          \end{tabular}
```

As this switch of shapes may not be desirable (e.g., the serifed euro may not blend well with the serifed document font), the package also offers the commands `\EURtm` (serifed symbol), `\EURhv` (sans serif symbol), and `\EURcr` (monospaced symbol)—the names being modeled after the three PostScript fonts Times, Helvetica, and Courier. These commands fix the font family, but react to requests for bold or oblique variants. However, as the last line in the previous example shows, none of the symbols blends particularly well with these fonts. Finally, the package offers `\EURofc`, which generates the official euro symbol (i.e., one from the sans serif regular font).

#### marvosym—Revisited for cash

Another free PostScript font that contains euro symbols as glyphs is the MarVoSym font, described in Section 7.8.3 on page 401. It is available in three shapes

to blend with Times, Helvetica, and Courier. As this font is a Pi font, it comes in only one weight series, which somewhat limits its usefulness as a source for the euro symbol. The font contains two glyphs with the official euro design, which differ in their amounts of side-bearings. To better demonstrate this difference, the following example puts a frame around them. It also shows the other currency symbols available in this package.

```
\usepackage{times,marvosym}
Currencies: \beta, \text{\textsterling}, \text{\texteuro}, \$, €
Comparisons: C €, C €, C €
Official logos: \text{\texteuro} or \text{\texteuro}

\usepackage{Shilling}
Currencies: \Shilling, \Denarius, \Pfund, \EyesDollar, \EURtm\par
Comparisons: C \EURtm, \textsf{C} \EURhv, \texttt{C} \EURcr \par
Official logos: {\Large \frame{\EUR} or \frame{\EURdig}}
```

7-8-22

## 7.9 The low-level interface

While the high-level font commands are intended for use in a document, the low-level commands are mainly for defining new commands in packages or in the preamble of a document; see also Section 7.9.4. To make the best use of such font commands, it is helpful to understand the internal organization of fonts in  $\text{\LaTeX}$ 's font selection scheme (NFSS).

One goal of  $\text{\LaTeX}$ 's font selection scheme is to allow rational font selection, with algorithms guided by the principles of generic markup. For this purpose, it would be desirable to allow independent changes for as many font attributes as possible. On the other hand, font families in real life normally contain only a subset of the myriad imaginable font attribute combinations. Therefore, allowing independent changes in too many attributes results in too many combinations for which no real (external) font is available and a default has to be substituted.

$\text{\LaTeX}$  internally keeps track of five independent font attributes: the “current encoding”, the “current family”, the “current series”, the “current shape”, and the “current size”. The encoding attribute was introduced in NFSS release 2 after it became clear that real support of multiple languages would be possible only by maintaining the character-encoding scheme independently of the other font attributes.

The values of these attributes determine the font currently in use.  $\text{\LaTeX}$  also maintains a large set of tables used to associate attribute combinations with external fonts (i.e., .tfm files that contain the information necessary for  $(\text{\La})\text{\TeX}$  to do its job). Font selection inside  $\text{\LaTeX}$  is then done in two steps:

1. A number of font attributes are changed using the low-level commands `\fontencoding`, `\fontfamily`, `\fontseries`, `\fontshape`, and `\fontsize`.
2. The font corresponding to this new attribute setting is selected by calling the `\selectfont` command.

The second step comprises several actions.  $\text{\LaTeX}$  first checks whether the font

corresponding to the desired attribute settings is known to the system (i.e., the `.tfm` file is already loaded) and, if so, this font is selected. If not, the internal tables are searched to find the external font name associated with this setting. If such a font name can be found, the corresponding `.tfm` file is read into memory and afterwards the font is selected for typesetting. If this process is not successful,  $\text{\LaTeX}$  tries to find an alternative font, as explained in Section 7.9.3.

### 7.9.1 Setting individual font attributes

Every font attribute has one command to change its current value. All of these commands will accept more or less any character string as an argument, but only a few values make sense. These values are not hard-wired into  $\text{\LaTeX}$ 's font selection scheme, but rather are conventions set up in the internal tables. The following sections introduce the naming conventions used in the standard set-up of  $\text{\LaTeX}$ , but anyone can change this set-up by adding new font declarations to the internal tables. Obviously, anybody setting up new fonts for use with  $\text{\LaTeX}$  should try to obey these conventions whenever possible, as only a consistent naming convention can guarantee that appropriate fonts are selected in a generically marked-up document.

If you want to select a specific font using this interface—say, Computer Modern Dunhill bold condensed italic 14pt—a knowledge of the interface conventions alone is not enough, as no external font exists for every combination of attribute values. You could try your luck by specifying something like the following set of commands:

```
\fontencoding{OT1}\fontfamily{cmdh}\fontseries{bc}\fontshape{it}%
\fontsize{14}{16pt}\selectfont
```

This code would be correct according to the naming conventions, as we will see in the following sections. Because this attribute combination does not correspond to a real font, however,  $\text{\LaTeX}$  would have to substitute a different font. The substitution mechanism may choose a font that is quite different from the one desired, so you should consult the font tables to see whether the desired combination is available. (Section 7.9.3 provides more details on the substitution process.) Every installation should have a local guide telling you exactly which fonts are available.

#### Choosing the font family

The font family is selected with the command `\fontfamily`. Its argument is a character string that refers to a font family declared in the internal tables. The character string was defined when these tables were set up and is usually a short letter sequence—for example, `cmtt` for the Computer Modern Typewriter family. The family names should not be longer than five letters, because they will be combined with possibly three more letters to form a file name, which on some systems can have at most eight letters.

<i>Weight Classes</i>		<i>Width Classes</i>		
Ultra Light	ul	Ultra Condensed	50%	uc
Extra Light	el	Extra Condensed	62.5%	ec
Light	l	Condensed	75%	c
Semi Light	s1	Semi Condensed	87.5%	sc
Medium (normal)	m	Medium	100%	m
Semi Bold	sb	Semi Expanded	112.5%	sx
Bold	b	Expanded	125%	x
Extra Bold	eb	Extra Expanded	150%	ex
Ultra Bold	ub	Ultra Expanded	200%	ux

Table 7.25: Weight and width classification of fonts

### Choosing the font series

The series attribute is changed with the `\fontseries` command. The series combines a weight and a width in its argument; in other words, it is not possible to change the width of the current font independently of its weight. This arrangement was chosen because it is hardly ever necessary to change weight or width individually. On the contrary, a change in weight (say, to bold) often is accompanied by a change in width (say, to extended) in the designer's specification. This is not too surprising, given that weight changes alter the horizontal appearance of the letters and thus call for adjustment in the expansion (i.e., the width) to produce a well-balanced look.

In the naming conventions for the argument for the `\fontseries` command, the names for both the weight and the width are abbreviated so that each combination is unique. The conventions are shown in Table 7.25. These classifications are combined in the argument to `\fontseries`; however, any instance of `m` (standing for medium in weight or width) is dropped, except when both weight and width are medium. The latter case is abbreviated with a single `m`. For example, bold expanded would be `bx`, whereas medium expanded would be `x` and bold medium would be `b`.

### Choosing the font shape

The `\fontshape` command is used to change the shape attribute. For the standard shapes, one- and two-letter abbreviations are used; these are shown in Table 7.26 on the facing page together with an example of the resulting shape in the Computer Modern Roman family.<sup>1</sup>

<sup>1</sup>The `o1` shape was produced using `\pcharpath` commands from the `ps-char` package, as Computer Modern does not contain such a shape. These types of graphical manipulations are discussed in [57].

<i>Abbreviation</i>	<i>Description</i>
n	upright (or normal) shape
it	<i>italic shape</i>
sl	<i>slanted or oblique shape</i>
sc	SMALL CAPS SHAPE
ui	upright italic shape
ol	OUTLINE shape

Table 7.26: Shape classification of fonts

### Choosing the font size

The font size is changed with the `\fontsize{<size>}{<skip>}` command. This is the only font attribute command that takes two arguments: the `<size>` to switch to and the baseline `<skip>` (the distance from baseline to baseline for this size). Font sizes are normally measured in points, so by convention the unit is omitted. The same is true for the second argument. However, if the baseline skip should be a rubber length—that is, if it contains plus or minus—you have to specify a unit. Thus, a valid size change could be requested by

```
\fontsize{14.4}{17}\selectfont
```

Even if such a request is valid in principle, no corresponding external font may exist in this size. In this case, L<sup>A</sup>T<sub>E</sub>X will try to find a nearby size if its internal tables allow for size correction or report an error otherwise.

If you use fonts existing in arbitrary sizes (for example, PostScript fonts), you can, of course, select any size you want. For example,

```
\fontsize{1in}{1.2in}\selectfont Happy Birthday
```

will produce a birthday poster line with letters in a one-inch size. However, there is one problem with using arbitrary sizes: if L<sup>A</sup>T<sub>E</sub>X has to typeset a formula in this size (which might happen behind the scenes without your knowledge), it needs to set up all fonts used in formulas for the new size. For an arbitrary size, it usually has to calculate the font sizes for use in subscripts and sub-subscripts (at least 12 different fonts). In turn, it probably has to load a lot of new fonts—something you can tell by looking at the transcript file. For this reason you may finally hit some internal limit if you have too many different size requests in your document. If this happens, you should tell L<sup>A</sup>T<sub>E</sub>X which sizes to load for formulas using the `\DeclareMathSizes` declaration, rather than letting it use its own algorithm. See Section 7.10.7 for more information on this issue.

### Choosing the encoding

A change of encoding is performed with the command `\fontencoding`, where the argument is the internal name for the desired encoding. This name must be

<i>Encoding</i>	<i>Description</i>	<i>Declared by</i>
T1	$\text{\LaTeX}$ text encoding (Latin) a.k.a. “Cork” encoding	$\text{\LaTeX}$
TS1	$\text{\LaTeX}$ symbol encoding (Latin)	$\text{\LaTeX}$
T2A,B,C	$\text{\LaTeX}$ text encodings (Cyrillic)	Cyrillic support packages
T3	$\text{\LaTeX}$ phonetic alphabet encoding	tipa package
TS3	$\text{\LaTeX}$ phonetic alphabet encoding (extra symbols)	tipa package
T4	$\text{\LaTeX}$ text encoding (African languages)	—
T5	$\text{\LaTeX}$ text encoding (Vietnamese)	—
T7	$\text{\LaTeX}$ text encoding (reserved for Greek)	—
OT1	$\text{\TeX}$ text as defined by Donald Knuth	$\text{\TeX}$
OT2	$\text{\TeX}$ text for Cyrillic languages (obsolete)	Cyrillic support packages
OT3	$\text{\TeX}$ phonetic alphabet encoding (obsolete)	—
OT4	$\text{\TeX}$ text with extensions for the Polish language	—
OT6	$\text{\TeX}$ text with extensions for the Armenian language	—
OML	$\text{\TeX}$ math text (italic) as defined by Donald Knuth	$\text{\TeX}$
OMS	$\text{\TeX}$ math symbol as defined by Donald Knuth	$\text{\TeX}$
OMX	$\text{\TeX}$ math extended symbol as defined by Donald Knuth	$\text{\TeX}$
X2	Extended text encoding (Cyrillic)	Cyrillic support packages
U	Unknown encoding (for arbitrary rubbish)	$\text{\TeX}$
L..	Local encoding (for private encodings)	—
LV1	Encoding used with some VTeX fonts	MicroPress
LY1	Alternative to T1 encoding	Y&Y

Table 7.27: Standard font encodings used with  $\text{\LaTeX}$ 

known to  $\text{\LaTeX}$ , either as one of the predefined encodings (loaded by the kernel) or as declared with the `\DeclareFontEncoding` command (see Section 7.10.5). A set of standard encoding names are given in Table 7.27.

$\text{\LaTeX}$ ’s font selection scheme is based on the (idealistic) assumption that most (or, even better, all) fonts for text are available in the same encoding as long as they are used to typeset in the same language. In other words, encoding changes should become necessary only if one is switching from one language to another. In that case it is normally the task of the language support packages (e.g., those from the babel system) to arrange matters behind the scenes.

In the following example we change the encoding manually by defining an environment `Cyr` for typesetting in Cyrillic. In this environment both the font encoding and the input encoding are locally changed. That might sound strange but if you work with an editor or keyboard that can switch input encodings on the fly this might be exactly the way your text is stored. Of course, for proper language support, additional work would be necessary, such as changing the hyphenation rules. The encodings are declared to  $\text{\LaTeX}$  by loading them with the `fontenc` pack-

age. T2A specifies one of the standard Cyrillic encodings; by loading T1 last, it becomes the default encoding for the document.

```
\usepackage[T2A,T1]{fontenc}\usepackage[koi8-r,latin1]{inputenc}
\newenvironment{Cyr}{\inputencoding{koi8-r}%
\fontencoding{T2A}\selectfont}{}
\raggedright \begin{Cyr}ÑÓÓÉÉÉ ÑÚÙË\end{Cyr}

```

Русский язык heißt auf  
Deutsch: die russische  
Sprache.  
7-9-1

heißt auf Deutsch: die russische Sprache.

Unfortunately, T1 is not fully implementable for most PostScript fonts. The following five characters are likely to show up as blobs of ink (indicating a missing glyph in the font). Note that the per thousand and per ten thousand symbols are actually formed by joining a percent sign and one or two additional small zeros; only the latter glyph is missing.

```
\usepackage[T1]{fontenc}
\fontfamily{cmr}\selectfont
\j{}\ \I{}\ \%{}\ \%oo
\fontfamily{ptm}\selectfont
\j{}\ \I{}\ \%{}\ \%oo
```

7-9-2

Potential T1  
encoding problems

As explained in Section 7.5.4 on page 362, the situation for TS1 is even worse, as sometimes half the glyphs from that encoding are not available in a given PostScript font.

### 7.9.2 Setting several font attributes

When designing page styles (see Section 4.4) or layout-oriented commands, you often want to select a particular font—that is, you need to specify values for all attributes. For this task L<sup>A</sup>T<sub>E</sub>X provides the command `\usefont`, which takes four arguments: the encoding, family, series, and shape. The command updates those attributes and then calls `\selectfont`. If you also want to specify the size and baseline skip, place a `\fontsize` command in front of it. For example,

```
\fontsize{14}{16pt}\usefont{OT1}{cmdh}{bc}{it}
```

would produce the same result as the hypothetical example on page 413.

Besides `\usefont`, L<sup>A</sup>T<sub>E</sub>X provides the `\DeclareFixedFont` declaration, which can be used to define new commands that switch to a completely fixed font. Such commands are extremely fast because they do not have to look up any internal tables. They are therefore very useful in command definitions that have to switch back and forth between fixed fonts. For example, for the `doc` package (see Chapter 14), one could produce code-line numbers using the following definitions:

```
\DeclareFixedFont\CodeLineFont{\encodingdefault}{\familydefault}{\seriesdefault}{\shapedefault}{7pt}
\newcommand{\theCodeLineNo}{\CodeLineFont\arabic{CodeLineNo}}
```

As you can see from the example, `\DeclareFixedFont` has six arguments: the name of the command to be defined followed by the five font attributes in the NFSS classification. Instead of supplying fixed values (except for the size), the built-in hooks that describe the main document font are used (see also Section 7.3.5). Thus, in the example above `\CodelineFont` still depends on the overall layout for the document (via the settings of `\encodingdefault` and other parameters). However, once the definition is carried out, its meaning is frozen, so later changes to the defaults will have no effect.

### 7.9.3 Automatic substitution of fonts

Whenever a font change request cannot be carried out because the combination is not known to L<sup>A</sup>T<sub>E</sub>X, it tries to recover by using a font with similar attributes. Here is what happens: if the combination of encoding scheme, family, series, and shape is not declared (see Section 7.10.3), L<sup>A</sup>T<sub>E</sub>X tries to find a known combination by first changing the shape attribute to a default. If the resulting combination is still unknown, it tries changing the series to a default. As a last resort, it changes the family to a default value. Finally, the internal table entry is looked up to find the requested size. For example, if you ask for `\ttfamily\bfseries\itshape`—a typewriter font in a bold series and italic shape (which usually does not exist)—then you will get a typewriter font in medium series and upright shape, because L<sup>A</sup>T<sub>E</sub>X first resets the shape before changing the series. If, in such a situation, you prefer a typewriter font in medium series with italic shape, you have to announce your intention to L<sup>A</sup>T<sub>E</sub>X using the `sub` function, which is explained on page 425.

The substitution process never changes the encoding scheme, because any alteration could produce wrong characters in the output. Recall that the encoding scheme defines how to interpret the input characters, while the other attributes define how the output should look. It would be catastrophic if, say, a £ sign were changed into a \$ sign on an invoice just because the software tried to be clever.

Thus, every encoding scheme must have a default family, series, and shape, and at least the combination consisting of the encoding scheme together with the corresponding defaults must have a definition inside L<sup>A</sup>T<sub>E</sub>X, as explained in Section 7.10.5.

### 7.9.4 Using low-level commands in the document

The low-level font commands described in the preceding sections are intended to be used in the definition of higher-level commands, either in class or package files or in the document preamble.

Whenever possible, you should avoid using the low-level commands directly in a document if you can use high-level font commands like `\textsf` instead. The reason is that the low-level commands are very precise instructions to switch to a particular font, whereas the high-level commands can be

customized using packages or declarations in the preamble. Suppose, for example, that you have selected Computer Modern Sans in your document using `\fontfamily{cmss}\selectfont`. If you later decide to typeset the whole document with fonts from the PSNFSS bundle—say, Times—applying a package would change only those parts of the document that do not contain explicit `\fontfamily` commands.

## 7.10 Setting up new fonts

### 7.10.1 Overview

Setting up new fonts for use with L<sup>A</sup>T<sub>E</sub>X basically means filling the internal font selection tables with information necessary for later associating a font request in a document with the external `.tfm` file containing character information used by (L)A<sub>T</sub>E<sub>X</sub>. Thus the tables are responsible for associating with

```
\fontencoding{OT1}\fontfamily{cmdh}\fontseries{m}\fontshape{n}%
\fontsize{10}{12pt}\selectfont
```

the external file `cmdunh10.tfm`. To add new fonts, you need to reverse this process. For every new external font you have to ask yourself five questions:

1. What is the font's encoding scheme—that is, which characters are in which positions?
2. What is its family name?
3. What is its series (weight and width)?
4. What is its shape?
5. What is its size?

The answers to these questions will provide the information necessary to classify your external font according to the L<sup>A</sup>T<sub>E</sub>X conventions, as described in Section 7.9. The next few sections discuss how to enter new fonts into the NFSS tables so that they can be used in the main text. You normally need this information if you want to make use of new fonts—for example, if you want to write a short package file for accessing a new font family. Later sections discuss more complicated concepts that come into play if you want to use, for example, special fonts for math instead of the standard ones.

If new fonts from the non-T<sub>E</sub>X world are to be integrated into L<sup>A</sup>T<sub>E</sub>X, it might be necessary to start even one step earlier: you may have to generate `.tfm` and probably virtual font files first. The tool normally used for this step is the `fontinst` program, written by Alan Jeffrey and further developed and now maintained by Lars Hellström. It is described in [57] and [64] and in the source documentation [74,75].

F	TT	W	[V.]	[N.]	[E]	[DD]
Foundry	Typeface Name	Weight	Variant	Encoding	Expansion	Design Size
e.g., p=Adobe	t=Times	b=Bold	i=Italic	8t=T1	n=narrow	10=10 point

Table 7.28: Karl Berry’s font file name classification scheme

### 7.10.2 Naming those thousands of fonts

A font naming scheme that can be used with TeX was proposed by Karl Berry [18], provoking some discussion [118]. The current version is described in [19] and has become the de facto standard in the TeX world. Berry tries to classify all font file names using eight alphanumeric characters, where case is not significant. This eight-character limit guarantees that the same file names can be used across all computer platforms and, more importantly, conforms to the ISO 9660 norm for CD-ROM. The principle of the scheme is described in Table 7.28, where the parts in brackets are omitted if they correspond to a default. For example, a design size is given only if the font is not linearly scaled. Table 7.8 on page 372 shows the classification of the 35 “basic” PostScript fonts according to L<sup>A</sup>T<sub>E</sub>X’s font interface. For each font the full Adobe name and, in parentheses, the corresponding short (Karl Berry) file name is given (without the encoding part). For OT1, T1, or TS1 one would need to append 7t, 8t, or 8c, respectively, to obtain the full file name—for example, putr8t for Utopia Regular in T1 encoding.

The naming convention covers internal TeX names for fonts (i.e., those used in \DeclareFontShape declarations as described in the next section), names for virtual fonts and their components (e.g., particular reencodings of physical fonts) [91], and the names of physical fonts. In case of PostScript fonts, the physical font names are often different from those used internally by TeX.

*A glimpse of the underworld*

In the latter case the mapping between internal font names and the external world has to happen when the result of a L<sup>A</sup>T<sub>E</sub>X run is viewed or printed. For example, the PostScript driver dvips uses mapping files (default extension .map) that contain lines such as

```
putr8r Utopia-Regular "TeXBase1Encoding ReEncodeFont" <8r.enc <putr8a.pfb
```

telling it that the font putr8r can be obtained from the external font putr8a.pfb by reencoding it via a special encoding vector (8r.enc in this case). However, when you look into t1put.f1 (the file that contains the \DeclareFontShape declarations for the Utopia family in the T1 encoding), you will find that putr8r is not referenced. Instead, you will find names such as putr8t. The reason is that putr8t is a virtual font (built with the help of fontinst [74, 75]) that references putr8r. The latter link is difficult to find (other than through the naming convention itself) if you do not have access to the sources that were used to build the virtual fonts actually used by TeX. Fortunately, you seldom have to dig into that part of a TeX system; if you do, you will find more information in [57, Chapter 10] or in the references listed above.

### 7.10.3 Declaring new font families and font shape groups

Each family/encoding combination must be made known to L<sup>A</sup>T<sub>E</sub>X through the command `\DeclareFontFamily`. This command has three arguments. The first two arguments are the encoding scheme and the family name. The third is usually empty, but it may contain special options for font loading and is explained on page 426. Thus, if you want to introduce a new family—say, Computer Modern Dunhill with the old T<sub>E</sub>X encoding scheme—you would write

```
\DeclareFontFamily{OT1}{cmdh}{}{}
```

A font family normally consists of many individual fonts. Instead of announcing each family member individually to L<sup>A</sup>T<sub>E</sub>X, you have to combine fonts that differ only in size and declare them as a group.

Such a group is entered into the internal tables of L<sup>A</sup>T<sub>E</sub>X with the command `\DeclareFontShape`, which takes six arguments. The first four are the encoding scheme, the family name, the series name, and the shape name under which you want to access these fonts later on. The fifth argument is a list of sizes and external font names, given in a special format that we discuss below. The sixth argument is usually empty; its use is explained on page 426.

We will first show a few examples and introduce terminology; then we will discuss all the features in detail.

As an example, an NFSS table entry for Computer Modern Dunhill medium (series) upright (shape) in the encoding scheme “T<sub>E</sub>X text” could be entered as

```
\DeclareFontShape{OT1}{cmdh}{m}{n}{<10> cmdunh10 }{}
```

assuming that only one external font for the size 10pt is available. If you also have this font available at 12pt (scaled from 10pt), the declaration would be

```
\DeclareFontShape{OT1}{cmdh}{m}{n}{<10> <12>cmdunh10 }{}
```

If the external font is available in all possible sizes, the declaration becomes very simple. This is the case for Type 1 PostScript (outline) fonts, or when the driver program is able to generate fonts on demand by calling METAFONT.

For example, Times Roman bold (series) upright (shape) in the L<sup>A</sup>T<sub>E</sub>X T1 encoding scheme could be entered as

```
\DeclareFontShape{T1}{ptm}{b}{n}{<-> ptmb8t }{}
```

This example declares a size range with two open ends (no sizes specified to the left and the right of the `-`). As a result, the same external .tfm file (ptmb8t) is used for all sizes and is scaled to the desired size. If you have more than one

.tfm file for a font—say, emtt10 for text sizes and emtt12 for display sizes (this is European Modern Typewriter)—the declaration could be

```
\DeclareFontShape{T1}{emtt}{m}{n}{<-12> emtt10 <12-> emtt12}{}
```

In this case, the .tfm file emtt10 would be used for sizes smaller than 12pt, and emtt12 for all sizes larger than or equal to 12pt.

The preceding examples show that the fifth argument of the command \DeclareFontShape consists of size specifications surrounded by angle brackets (i.e., < . . . >) intermixed with loading information for the individual sizes (e.g., font names). The part inside the angle brackets is called the “size info” and the part following the closing angle bracket is called the “font info”. The font info is further structured into a “size function” (often empty) and its arguments; we discuss this case below. Within the arguments of \DeclareFontShape, blanks are ignored to help make the entries more readable.<sup>1</sup> In the unusual event that a real space has to be entered, you can use the command \space.

### Simple sizes and size ranges

The size infos—the parts between the angle brackets in the fifth argument to \DeclareFontShape—can be divided into “simple sizes” and “size ranges”. A simple size is given by a single (decimal) number, like <10> or <14.4>, and in principle can have any positive value. However, because the number represents a font size measured in points, you probably will not find values less than 4 or greater than 120. A size range is given by two simple sizes separated by a hyphen, to indicate a range of font sizes that share the same font info. The lower boundary (i.e., the size to the left of the hyphen) is included in the range, while the upper boundary is excluded. For example, <5–10> denotes sizes greater than or equal to 5pt and less than 10pt. You can omit the number on either side of the hyphen in a size range, with the obvious interpretation: <–> stands for all possible sizes, <–10> stands for all sizes less than 10pt, and <12–> stands for all sizes greater than or equal to 12pt.

Often several simple sizes have the same font info. In that case a convenient shorthand is to omit all but the last font infos:

```
\DeclareFontShape{OT1}{panr}{m}{n}{<5> <6> <7> <8> <9> <10>
<10.95> <12> <14.4> <17.28> <20.74> <24.88> pan10 }{}
```

This example declares the font Pandora medium Roman as being available in several sizes, all of them produced by scaling from the same design size.

<sup>1</sup>This is true only if the command is used at the top level. If such a declaration is used inside other constructs (e.g., the argument of \AtBeginDocument), blanks might survive and in that case entries will not be recognized.

### Size functions

As noted earlier, the font info (the string after the closing angle bracket) is further structured into a size function and its argument. If an \* appears in the font info string, everything to the left of it forms the function name and everything to the right is the argument. If there is no asterisk, as in all of the examples so far, the whole string is regarded as the argument and the function name is “empty”.

Based on the size requested by the user and the information in the \DeclareFontShape command, size functions produce the specification necessary for L<sup>A</sup>T<sub>E</sub>X to find the external font and load it at the desired size. They are also responsible for informing the user about anything special that happens. For example, some functions differ only in terms of whether they issue a warning. This capability allows the system maintainer to set up L<sup>A</sup>T<sub>E</sub>X in the way best suited for the particular site.

The name of a size function consists of zero or more letters. Some of the size functions can take two arguments, one optional and one mandatory. Such an optional argument has to be enclosed in square brackets. For example, the specification

```
<-> s * [0.9] cmfib8
```

would select, for all possible sizes (we have the range 0 to  $\infty$ ), the size function **s** with the optional argument 0.9 and the mandatory argument **cmfib8**.

The size specifications in \DeclareFontShape are inspected in the order in which they are given. When a size info matches the requested user size, the corresponding size function is executed. If this process yields a valid font, no further entries are inspected. Otherwise, the search continues with the next entry. The standard size functions are listed below. The document **fntguide.tex** [109], which is part of the L<sup>A</sup>T<sub>E</sub>X distribution, describes how to define additional functions should it ever become necessary.

**The “empty” function** Because the empty function is used most often, it has the shortest possible name. (Every table entry takes up a small bit of internal memory, so the syntax chosen tries to find a balance between a perfect user interface and compactness of storage.) The empty function loads the font info exactly at the requested size if it is a simple size. If there is a size range and the size requested by the user falls within that range, it loads the font exactly at the user size.

For example, if the user requested 14.4, then the specification

```
<-> panr10
```

would load the .tfm file called **panr10.tfm** at 14.4pt. Because this font was designed for 10pt (it is the Pandora Roman font at 10pt), all the values in the .tfm file are scaled by a factor of 1.44.

Sometimes one wants to load a font at a slightly larger or smaller size than the one requested by the user. This adjustment may be necessary when fonts from one family appear to be too large compared to fonts from other families used in the same document. For this purpose the empty size function allows an optional argument to represent a scale factor that, if present, is multiplied by the requested size to yield the actual size to be loaded. Thus

```
<-> [0.95] phvr8t
```

would always load the `.tfm` file called `phvr8t.tfm` (Helvetica in T1 encoding) at 95% of the requested size. If the optional argument is used, the empty size function will issue a warning to alert the user that the font is not being loaded at its intended size.

**The “s” function** The `s` function has the same functionality as the empty function, but does not produce warnings (the `s` means “silence”). Writing

```
\DeclareFontShape{T1}{phv}{m}{n}{ <-> s * [0.95] phvr8t }{}
```

avoids all the messages that would be generated on the terminal if the empty function were used. Messages are still written to the transcript file, so you can find out which fonts were used if something goes wrong. The `helvet` package is implemented in this way, except that the scaling factor is not hard-wired but rather passed via a package option to the `\DeclareFontShape` declaration.

**The “gen” function** Often the external font names are built by appending the font size to a string that represents the typeface. For example, `cmtt8`, `cmtt9`, and `cmtt10` are the external names for the fonts Computer Modern Typewriter at 8, 9, and 10pt, respectively. With font names organized according to such a scheme, you can make use of the `gen` function to shorten the entry. This function combines the font info and the requested size to generate (hence `gen`) the external font names. Thus, you can write

```
<8> <9> <10> gen * cmtt
```

as shorthand for

```
<8> cmtt8 <9> cmtt9 <10> cmtt10
```

thereby saving eight characters in the internal tables of NFSS. This function combines both parts literally, so you should not use it with decimal sizes like `14.4`. Also, you must ensure that the digits in the external font name really represent the design size (for example, `cmr17` is actually Computer Modern Roman at 17.28pt).

In all other respects, the `gen` function behaves like the empty function. That is, the optional argument, if given, represents a scale factor and, if used, generates an information message.

**The “sgen” function** The `sgen` function is the silent variant of the `gen` function. It writes any message only to the transcript file.

**The “genb” function** This size function is similar to `gen`, but is intended for fonts in which the size is encoded in the font name in centipoins, such as the EC fonts. As a consequence, a line such as

```
<9> <10> <10.95> <12> genb * ecrm
```

acts as shorthand for

```
<9> ecrm0900 <10> ecrm1000 <10.95> ecrm1095 <12> ecrm1200
```

An optional argument, if present, will have the same effect as it would with the empty function—it provides a scale factor and, if used, generates an information message.

**The “sgenb” function** The `sgenb` function is the silent variant of the `genb` function. It writes any message only to the transcript file.

**The “sub” function** The `sub` function is used to substitute a different font shape group if no external font exists for the current font shape group. In this case the argument is not an external font name but rather a different family, series, and shape combination separated by slashes (the encoding will not change for the reasons explained earlier). For example, the Computer Modern Sans family has no italic shape, only a slanted shape. Thus, it makes sense to declare the slanted shape as a substitute for the italic one:

```
\DeclareFontShape{OT1}{cmss}{m}{it}{ <-> sub * cmss/m/sl }{}
```

Without this declaration, L<sup>A</sup>T<sub>E</sub>X’s automatic substitution mechanism (see Section 7.9.3) would substitute the default shape, Computer Modern Sans upright.

Besides the substitution of complete font shape groups, there are other good uses for the `sub` function. Consider the following code:

```
\DeclareFontShape{OT1}{cmss}{m}{sl}{ <-8> sub * cmss/m/n
<8> cmssi8 <9> cmssi9 <10><10.95> cmssi10 <12><14.4> cmssi12
<17.28><20.74><24.88> cmssi17 }{}
```

This declaration states that for sizes smaller than 8pt L<sup>A</sup>T<sub>E</sub>X should look in the font shape declaration for OT1/cmss/m/n. Such substitutions can be chained. People familiar with the standard font distribution know that there is no Computer Modern Sans font smaller than 8pt, so the substituted font shape group will probably contain another substitution entry. Nevertheless, using this device has the advantage that when you get an additional font you have to change only one font shape group declaration—other declarations that use this the font will benefit automatically.

**The “ssub” function** The `ssub` function has the same functionality as the `sub` function, but does not produce on-screen warnings (the first `s` means “silence”).

**The “subf” function** The `subf` function is a cross between the empty function and `sub`, in that it loads fonts in the same way as the empty function but produces a warning that this operation was done as a substitution because the requested font shape is not available. You can use this function to substitute some external fonts without having to declare a separate font shape group for them, as in the case of the `sub` function. For example,

```
\DeclareFontShape{OT1}{ptm}{bx}{n}{} <-> subf * ptmb7t {}{}
```

would warn the user that the requested combination is not available and, therefore, that the font `ptmb7t` was loaded instead. As this is less informative than using the `sub` function, the latter should be preferred.

**The “ssubf” function** The silent variant of `subf`, this function writes its messages only to the transcript file.

**The “fixed” function** This function disregards the requested size and instead loads the external font given as an argument. If present, the optional argument denotes the size (in points) at which the font will be loaded. Thus, this function allows you to specify size ranges for which one font in some fixed size will be loaded.

**The “sfixed” function** The silent variant of `fixed`, this function is used, for example, to load the font containing the large math symbols, which is often available only in one size.

#### Font-loading options

As already mentioned, you need to declare each family using the command `\DeclareFontFamily`. The third argument to this command, as well as the sixth argument to `\DeclareFontShape`, can be used to specify special operations that

are carried out when a font is loaded. In this way, you can change parameters that are associated with a font as a whole.

For every external font, (I<sup>A</sup>) $\text{\TeX}$  maintains, besides the information about each character, a set of global dimensions and other values associated with the font. For example, every font has its own “hyphen character”, the character that is inserted automatically when (I<sup>A</sup>) $\text{\TeX}$  hyphenates a word. Another example is the normal width and the stretchability of a blank space between words (the “interword space”); again a value is maintained for every font and changed whenever (I<sup>A</sup>) $\text{\TeX}$  switches to a new font. By changing these values when a font is loaded, special effects can be achieved.

Normally, changes apply to a whole family; for example, you may want to prohibit hyphenation for all words typeset in the typewriter family. In this case, the third argument of `\DeclareFontFamily` should be used. If the changes should apply only to a specific font shape group, you must use the sixth argument of `\DeclareFontShape`. In other words, when a font is loaded, NFSS first applies the argument of `\DeclareFontFamily` and then the sixth argument of `\DeclareFontShape`, so that it can override the load options specified for the whole family if necessary.

Below we study the information that can be set in this way (unfortunately, not everything is changeable) and discuss some useful examples. This part of the interface addresses very low-level commands of  $\text{\TeX}$ . Because it is so specialized, no effort was made to make the interface more I<sup>A</sup> $\text{\TeX}$ -like. As a consequence, the methods for assigning integers and dimensions to variables are somewhat unusual.

With `\hyphenchar\font=\langle number\rangle`, (I<sup>A</sup>) $\text{\TeX}$  specifies the character that is inserted as the hyphen when a word is hyphenated. The `\langle number\rangle` represents the position of this character within the encoding scheme. The default is the value of `\defaulthyphenchar`, which is 45, representing the position of the “-” character in most encoding schemes. If this number is set to -1, hyphenation is suppressed. Thus, by declaring

```
\DeclareFontFamily{OT1}{cmtt}{\hyphenchar\font=-1}
```

you can suppress hyphenation for all fonts in the `cmtt` family with the encoding scheme OT1. Fonts with the T1 encoding have an alternate hyphen character in position 127, so that you can set, for example,

```
\DeclareFontFamily{T1}{cmr}{\hyphenchar\font=127}
```

This makes the hyphen character inserted by (I<sup>A</sup>) $\text{\TeX}$  different from the compound-word dash entered in words like “so-called”. (I<sup>A</sup>) $\text{\TeX}$  does not hyphenate words that already contain explicit hyphen characters (except just after the hyphen), which can create a real problem in languages in which the average word length is much larger than in English. With the above setting this problem can be solved.

*Changing the  
hyphenation  
character*

Every (I)TeX font has an associated set of dimensions, which are changed by assignments of the form `\fontdimen<number>\font=<dimen>`, where `<number>` is the reference number for the dimension and `<dimen>` is the value to be assigned. The default values are taken from the `.tfm` file when the font is loaded. Each font has at least seven such dimensions:

- `\fontdimen1` Specifies the slant per point of the characters. If the value is zero, the font is upright.
- `\fontdimen2` Specifies the normal width of a space used between words (interword space).
- `\fontdimen3` Specifies the additional stretchability of the interword space—that is, the extra amount of white space that (I)TeX is allowed to add to the space between words to produce justified lines in a paragraph. In an emergency (I)TeX may add more space than this allowed value; in that case an “underfull box” will be reported.
- `\fontdimen4` Specifies the allowed shrinkability of the interword space—that is, the amount of space that (I)TeX is allowed to subtract from the normal interword space (`\fontdimen2`) to produce justified lines in a paragraph. (I)TeX will never shrink the interword space to less than this minimum.
- `\fontdimen5` Specifies the x-height. It defines the font-oriented dimension `1ex`.
- `\fontdimen6` Specified the quad width. It defines the font-oriented dimension `1em`.
- `\fontdimen7` Specifies the amount intended as extra space to be added after certain end-of-sentence punctuation characters when `\nonfrenchspacing` is in force. The exact rules for when TeX uses this dimension (all or some of the extra space) are somewhat complex; see *The TeXbook* [82] for details. It is always ignored or rather replaced by the value `\xspaceskip`, when that value is nonzero.

When changing the interword spacing associated with a font, you cannot use an absolute value because such a value must be usable for all sizes within one font shape group. You must, therefore, define the value by using some other parameter that depends on the font. You could say, for example,

```
\DeclareFontShape{OT1}{cmr}{m}{n}{\dots}
  {\fontdimen2\font=.7\fontdimen2\font}
```

This declaration reduces the normal interword space to 70% of its original value. In a similar manner, the stretchability and shrinkability could be changed.

Some fonts used in formulas need more than seven font dimensions—namely, the symbol fonts called “symbols” and “largesymbols” (see Section 7.10.7). TeX will not typeset a formula if these symbol fonts have fewer than 22 and 13

\fontdimen parameters, respectively. The values of these parameters are used to position the characters in a math formula. An explanation of the meaning of every such \fontdimen parameter is beyond the scope of this book; details can be found in Appendix G of *The TeXbook* [82].

One unfortunate optimization is built into the TeX system: TeX loads every .tfm file only once for a given size. It is, therefore, impossible to define one font shape group (with the \DeclareFontShape command) to load some external font—say, cmtt10—and to use another \DeclareFontShape command to load the same external font, this time changing some of the \fontdimen parameters or some other parameter associated with the font. Trying to do so changes the values for both font shape groups.

Suppose, for example, that you try to define a font shape with tight spacing by making the interword space smaller:

```
\DeclareFontShape{T1}{ptm}{m}{n}{ <-> ptmr8t }{}
\DeclareFontShape{T1}{ptm}{c}{n}{ <-> ptmr8t }
    {\fontdimen2\font=.7\fontdimen2\font}
```

This declaration will not work. The interword spacing for the medium shape will change when the tight shape is loaded to the values specified there, and this result is not what is wanted. The best way to solve this problem is to define a virtual font that contains the same characters as the original font, but differs in the settings of the font dimensions (see [73, 74, 91]). Another possible solution is to load the font at a slightly different size, as in the following declaration:

```
\DeclareFontShape{T1}{ptm}{c}{n}{ <-> [0.9999] ptmr8t }
    {\fontdimen2\font=.7\fontdimen2\font}
```

That strategy makes them different fonts for TeX with separate \fontdimen parameters. Alternatively, in this particular case you can control the interword space by setting \spaceskip, thereby overwriting the font values. See Section 3.1.12 for some discussion of that parameter.

#### 7.10.4 Modifying font families and font shape groups

If you need a nonstandard font shape group declaration for a particular document, just place your private declaration in a package or the preamble of your document. It will then overwrite any existing declaration for the font shape combination. Note, however, that the use of \DeclareFontFamily prevents a later loading of the corresponding .fd file (see Section 7.10.6). Also, your new declaration has no effect on fonts that are already loaded.

Today's L<sup>A</sup>T<sub>E</sub>X format preloads by default only a small number of fonts. However, by using the configuration file preload.cfg, more or fewer fonts can be loaded when the format is built. None of these preloaded fonts can be manipulated using font family or font shape declarations. Thus, if you want some special settings for the core fonts, you must ensure that none of these fonts is preloaded.

For additional information on ways to customize a  $\text{\LaTeX}$  installation, refer to the document `cfgguide.tex` [110], which is part of the  $\text{\LaTeX}$  distribution.

### 7.10.5 Declaring new font encoding schemes

Font changes that involve alterations in the encoding scheme require taking certain precautions. For example, in the T1 encoding, most accented letters have their own glyphs, whereas in the traditional  $\text{\TeX}$  text encoding (OT1), accented letters must be generated from accents and letters using the `\accent` primitive. (It is desirable to use glyphs for accented letters rather than employing the `\accent` primitive because, among other things, the former approach allows for correct hyphenation.) If the two approaches have to be mixed, perhaps because a font is available only in one of the encodings, the definition of a command such as `\"` must behave differently depending on the current font encoding.

For this reason, each encoding scheme has to be formally introduced to  $\text{\LaTeX}$  with a `\DeclareFontEncoding` command, which takes three arguments. The first argument is the name of the encoding under which you access it using the `\fontencoding` command. Table 7.27 on page 416 provides a list of standard encoding schemes and their internal NFSS names.

The second argument contains any code (such as definitions) to be executed every time  $\text{\TeX}$  switches from one encoding to another using the `\fontencoding` command. The final argument contains code to be used whenever the font is accessed as a mathematical alphabet. Thus, these three arguments can be used to redefine commands that depend on the positions of characters in the encoding. To avoid spurious spaces in the output (coming from extra spaces in the arguments), the space character is ignored within them. In the unlikely event that you need spaces in a definition in one of the arguments, use the `\space` command.

The  $\text{\TeX}3$  project reserves the use of encodings starting with the following letters: T (standard text encodings with 256 characters), TS (symbols that are designed to extend the corresponding T encoding), X (text encodings that do not conform to the strict requirements for T encodings), M (standard math encodings with 256 characters), S (other symbol encodings), A (other special applications), OT (standard text encodings with 128 characters), and OM (standard math encodings with 128 characters). The letter O was chosen to emphasize that the 128-character encodings are old and *obsolete*. Ideally, these encodings will be superseded by standards defined by the  $\text{\TeX}$  user groups so that in the future a change of encoding will be necessary only if one is switching from one language to another.

For your own private encodings, you should choose names starting with L for “local” or E for “experimental”. Encodings starting with U are for “Unknown” or “Unclassified” encodings—that is, for fonts that do not fit a common encoding pattern. This naming convention ensures that files using official encodings are portable. New standard encodings will be added to the  $\text{\TeX}$  documentation as they emerge. For example, the T2\* and T5 encodings have appeared since the first edition of this book was published.

The `\DeclareFontEncoding` command stores the name of the newly declared encoding in the command `\LastDeclaredEncoding`. This feature is sometimes useful when you are declaring other related encoding information and is, for example, used in the encoding declaration files for the Cyrillic languages.

Also, as we saw in Section 7.9.3 on font substitution, the default values for the family, series, and shape may need to be different for different encodings. For this purpose, NFSS provides the command `\DeclareFontSubstitution`, which again takes the encoding as the first argument. The next three arguments are the default values (associated with this encoding) for family, series, and shape for use in the automatic substitution process, as explained in Section 7.9.3. It is important that these arguments form a valid font shape—in other words, that a `\DeclareFontShape` declaration exists for them. Otherwise, an error message will be issued when NFSS checks its internal tables at `\begin{document}`.

### 7.10.6 Internal file organization

Font families can be declared when a format file is generated, declared in the document preamble, or loaded on demand when a font change command in the document requests a combination that has not been used so far. The first option consumes internal memory in every *L<sup>A</sup>T<sub>E</sub>X* run, even if the font is not used. The second and third possibilities take a little more time during document formatting, because the font definitions have to be read during processing time. Nevertheless, it is preferable to use the latter solutions for most font shape groups, because it allows you to typeset a wide variety of documents with a single *L<sup>A</sup>T<sub>E</sub>X* format.

When the format is generated, *L<sup>A</sup>T<sub>E</sub>X* will read a file named `fonttext.ltx`, which contains the standard set of font family definitions and some other declarations related to text fonts. With some restrictions<sup>1</sup> this set can be altered by providing a configuration file `fontdef.cfg`; see the documentation `cfgguide.tex`.

All other font family definitions should be declared in external files loaded on request: either package files or font definition (.fd) files. If you place font family definitions in a package file, you must explicitly load this package after the `\documentclass` command. But there is a third possibility: whenever NFSS gets a request for a font family `foo` in an encoding scheme `BAR`, and it has no knowledge about this combination, it will try to load a file called `barfoo.fd` (all letters lowercase). If this file exists, it is supposed to contain font shape group definitions for the family `foo` in the encoding scheme `BAR`—that is, declarations of the form

```
\DeclareFontFamily{BAR}{foo}{...}
\DeclareFontShape{BAR}{foo}{...}{...}{...}{...}
...
\endinput
```

<sup>1</sup>Any such customization should not be undertaken lightly as it is unfortunately very easy to produce a *L<sup>A</sup>T<sub>E</sub>X* format that shows subtle or even glaring incompatibilities with other installations.

In this way it becomes possible to declare a huge number of font families for L<sup>A</sup>T<sub>E</sub>X without filling valuable internal memory with information that is almost never used.<sup>1</sup>

Each .fd file should contain all font definitions for one font family in one encoding scheme. It should consist of one or more \DeclareFontShape declarations and exactly one \DeclareFontFamily declaration. Other definitions should not appear in the file, except perhaps for a \ProvidesFile declaration or some \typeout statement informing the user about the font loading. As an alternative to the \typeout command, you can use the plain T<sub>E</sub>X command \wlog, which writes its argument only into the transcript file. Detailed information in the transcript file should be generated by all .fd files that are used in production, because looking at this transcript will help to locate errors by providing information about the files and their versions used in a particular job. If \typeout or \wlog commands are used, it is important to know that spaces and empty lines in a .fd file are ignored. Thus, you have to use the command \space in the argument to \typeout or \wlog to obtain a blank space on the screen and the transcript file.

New encoding schemes cannot be introduced via the .fd mechanism. NFSS will reject any request to switch to an encoding scheme that was not explicitly declared in the L<sup>A</sup>T<sub>E</sub>X format (i.e., fonttext.1tx), in a package file, or in the preamble of the document.

### 7.10.7 Declaring new fonts for use in math

#### Specifying font sizes

For every text size NFSS maintains three sizes that are used to typeset formulas (see also Section 8.7.1): the size in which to typeset most of the symbols (selected by \textstyle or \displaystyle); the size for first-order subscripts and superscripts (\scriptstyle); and the size for higher-order subscripts and superscripts (\scriptstyle). If you switch to a new text size, for which the corresponding math sizes are not yet known, NFSS tries to calculate them as fractions of the text size. Instead of letting NFSS do the calculation, you might want to specify the correct values yourself via \DeclareMathSizes. This declaration takes four arguments: the outer text size and the three math sizes for this text size. For example, the class file for *The L<sup>A</sup>T<sub>E</sub>X Companion* contains settings like the following:

```
\DeclareMathSizes{14}{14}{10}{7} \DeclareMathSizes{36}{ }{ }{ }
```

The first declaration defines the math sizes for the 14pt heading size to be 14pt, 10pt, and 7pt, respectively. The second declaration (the size for the chapter head-

<sup>1</sup>Unfortunately, this feature is not fully available on (I)A<sub>T</sub>E<sub>X</sub> installations that use different search paths for the commands \input and \openin. On such systems the .fd feature can be activated at installation time by supplying NFSS with a full path denoting the directories containing all the .fd files. As a result, local .fd files—those stored in the current directory—may not be usable on such systems.

ings) informs NFSS that no math sizes are necessary for 36pt text size. This avoids the unnecessary loading of more than 30 additional fonts. For the first edition of *The L<sup>A</sup>T<sub>E</sub>X Companion* such declarations were very important to be able to process the book with all its examples as a single document (the book loaded 228 fonts out of a maximum of 255). Today, TeX installations are usually compiled with larger internal tables (e.g., the laptop implementation used to write this chapter allows 1000 fonts), so conserving space is no longer a major concern. In any event you should be careful about disabling math sizes, because if some formula is typeset in such a size after all, it will be typeset in whatever math sizes are still in effect from an earlier text size.

### Adding new symbol fonts

We have already seen how to use math alphabet commands to produce letters with special shapes in a formula. We now discuss how to add fonts containing special symbols, called “symbol fonts”, and how to make such symbols accessible in formulas.

The process of adding new symbol fonts is similar to the declaration of a new math alphabet identifier: `\DeclareSymbolFont` defines the defaults for all math versions, and `\SetSymbolFont` overrides the defaults for a particular version.

The math symbol fonts are accessed via a symbolic name, which consists of a string of letters. If, for example, you want to install the AMS fonts `msbm10`, shown in Table 7.29 on the following page, you first have to make the typeface known to NFSS using the declarations described in the previous sections. These instructions would look like

```
\DeclareFontFamily{U}{msb}{}  
\DeclareFontShape{U}{msb}{m}{n}{<5> <6> <7> <8> <9> gen * msbm  
<10> <10.95> <12> <14.4> <17.28> <20.74> <24.88> msbm10}{}
```

and are usually placed in an `.fd` file. You then have to declare that symbol font for all math versions by issuing the command

```
\DeclareSymbolFont{AMSB}{U}{msb}{m}{n}
```

It makes the font shape group `U/msb/m/n` available as a symbol font under the symbolic name `AMSB`. If there were a bold series in this font family (unfortunately there is not), you could subsequently change the set-up for the bold math version by saying

```
\SetSymbolFont{AMSB}{bold}{U}{msb}{b}{n}
```

After taking care of the font declarations, you can make use of this symbol font in math mode. But how do you tell NFSS that `$a\lessdot b$` should produce

	'0	'1	'2	'3	'4	'5	'6	'7	
'00x	$\leq$	$\geq$	$\not\leq$	$\not\geq$	$\not\approx$	$\not\equiv$	$\not\asymp$	$\not\sim$	"0x
'01x	$\triangleleft$	$\triangleright$	$\not\triangleleft$	$\not\triangleright$	$\trianglelefteq$	$\trianglerighteq$	$\not\trianglelefteq$	$\not\trianglerighteq$	"1x
'02x	$\asymp$	$\sim$	$\not\asymp$	$\not\sim$	$\asymp$	$\not\asymp$	$\asymp$	$\not\asymp$	"2x
'03x	$\asymp$	$\sim$	$\not\asymp$	$\not\sim$	$\approx$	$\not\approx$	$/$	$\backslash$	"3x
'04x	$\not\subseteq$	$\not\supseteq$	$\not\subset$	$\not\supset$	$\subseteq$	$\supseteq$	$\not\subseteq$	$\not\supseteq$	"4x
'05x	$\subset$	$\supset$	$\not\subset$	$\not\supset$	$\not\parallel$	$\parallel$	$\not\parallel$	$\not\parallel$	"5x
'06x	$\nabla$	$\nabla$	$\not\nabla$	$\not\nabla$	$\not\nabla$	$\not\nabla$	$\not\nabla$	$\not\nabla$	"6x
'07x	$\leftrightarrow$	$\not\leftrightarrow$	$\not\leftrightarrow$	$\not\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$*$	$\emptyset$	"7x
'10x	$\pm$	A	B	C	D	E	F	G	
'11x	H	I	J	K	L	M	N	O	
'12x	P	Q	R	S	T	U	V	W	
'13x	X	Y	Z	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	
'14x	$\exists$	$\exists$					$\wp$	$\wp$	
'15x	$\approx$	$\bowtie$	$\sqcap$	$\sqcup$	$\triangleleft$	$\triangleleft$	$\times$	$\times$	
'16x	$\vdash$	$\vdash$	$\backslash$	$\sim$	$\approx$	$\approx$	$\approx$	$\approx$	
'17x	$\curvearrowleft$	$\curvearrowright$	F	$\not\approx$	k	$\hbar$	$\hbar$	$\circ$	
	"8	"9	"A	"B	"C	"D	"E	"F	

Table 7.29: Glyph chart for msbm10 produced by the nfssfont.tex program

$a \lessdot b$ , for example? To do so, you have to introduce your own symbol names to NFSS, using `\DeclareMathSymbol`.

```
\DeclareMathSymbol{cmd}{type}{symbol-font}{slot}
```

The first argument to `\DeclareMathSymbol` is your chosen command name. The second argument is one of the commands shown in Table 7.30 on the next page and describes the nature of the symbol—whether it is a binary operator, a relation, and so forth. (I) $\mathrm{\ddot{E}}$ X uses this information to leave the correct amount of space around the symbol when it is encountered in a formula. Incidentally, except for `\mathalpha`, these commands can be used directly in math formulas as functions with one argument, in which case they space their (possibly complex) argument as if it were of the corresponding type; see Section 8.9 on page 524.

The third argument identifies the symbol font from which the symbol should be fetched—that is, the symbolic name introduced with the `\DeclareSymbolFont` command. The fourth argument gives the symbol's position in the font encoding, either as a decimal, octal, or hexadecimal value. Octal (base 8) and hexadecimal

Type	Meaning	Example	Type	Meaning	Example
\mathord	Ordinary	/	\mathopen	Opening	(
\mathop	Large operator	\sum	\mathclose	Closing	)
\mathbin	Binary operation	+	\mathpunct	Punctuation	,
\mathrel	Relation	=	\mathalpha	Alphabet character	A

Table 7.30: Math symbol type classification

(base 16) numbers are preceded by ' and ", respectively. If you look at Table 7.29 on the preceding page, you can easily determine the positions of all glyphs in this font. Such tables can be printed using the  $\text{\LaTeX}$  program `nfssfont.tex`, which is part of the  $\text{\LaTeX}$  distribution; see Section 7.5.7 on page 369. For example, `\lessdot` would be declared using

```
\DeclareMathSymbol{\lessdot}{\mathbin}{AMSb}{"6C}
```

Instead of a command name, you can use a single character in the first argument. For example, the `eulervm` package has several declarations of the form

```
\DeclareMathSymbol{0}{\mathalpha}{letters}{"30}
```

that specify where to fetch the digits from.

Because `\DeclareMathSymbol` is used to specify a position in some symbol font, it is important that all external fonts associated with this symbol font via the `\DeclareSymbolFont` and `\SetSymbolFont` commands have the same character in that position. The simplest way to ensure this uniformity is to use only fonts with the same encoding (unless it is the U, a.k.a. unknown, encoding, as two fonts with this encoding are not required to implement the same characters).

Besides `\DeclareMathSymbol`,  $\text{\LaTeX}$  knows about `\DeclareMathAccent`, `\DeclareMathDelimiter`, and `\DeclareMathRadical` for setting up math font support. Details about these slightly special declarations can be found in [109], which is part of every  $\text{\LaTeX}$  distribution.

If you look again at the glyph chart for `msbm10` (Table 7.29 on the preceding page), you will notice that this font contains “blackboard bold” letters, such as ABC. If you want to use these letters as a math alphabet, you can define them using `\DeclareMathAlphabet`, but given that this symbol font is already loaded to access individual symbols, it is better to use a shortcut:

```
\DeclareSymbolFontAlphabet{\mathbb}{AMSb}
```

That is, you give the name of your math alphabet identifier and the symbolic name of the previously declared symbol font.

An important reason for not unnecessarily loading symbol fonts twice is that there is an upper limit of 16 math fonts that can be active at any given time in (I)TeX. In calculating this limit, each symbol font counts; math alphabets count only if they are actually used in the document, and they count locally in each math version. Thus, if eight symbol fonts are declared, you can use a maximum of eight (possibly different) math alphabet identifiers within every version.

To summarize: to introduce new symbol fonts, you need to issue a small number of `\DeclareSymbolFont` and `\SetSymbolFont` declarations and a potentially large number of `\DeclareMathSymbol` declarations; hence, adding such fonts is best done in a package file.

### Introducing new math versions

We have already mentioned that the standard set-up automatically declares two math versions, normal and bold. To introduce additional versions, you use the declaration `\DeclareMathVersion`, which takes one argument, the name of the new math version. All symbol fonts and all math alphabets previously declared are automatically available in this math version; the default fonts are assigned to them—that is, the fonts you have specified with `\DeclareMathAlphabet` or `\DeclareSymbolFont`.

You can then change the set-up for your new version by issuing appropriate `\SetMathAlphabet` and `\SetSymbolFont` commands, as shown in previous sections (pages 352 and 433) for the bold math version. Again, the introduction of a new math version is normally done in a package file.

### Changing the symbol font set-up

Besides adding new symbol fonts to access more symbols, the commands we have just seen can be used to change an existing set-up. This capability is of interest if you choose to use special fonts in some or all math versions.

The default settings in L<sup>A</sup>T<sub>E</sub>X are given here:

```
\DeclareMathVersion{normal} \DeclareMathVersion{bold}

\DeclareSymbolFont{operators}    {OT1}{cmr}{m}{n}
\DeclareSymbolFont{letters}      {OML}{cmm}{m}{it}
\DeclareSymbolFont{symbols}      {OMS}{cmsy}{m}{n}
\DeclareSymbolFont{largesymbols}  {OMX}{cmex}{m}{n}

% Special bold fonts only for these:
\SetSymbolFont  {operators}{bold}{OT1}{cmr}{bx}{n}
\SetSymbolFont  {letters}  {bold}{OML}{cmm}{b}{it}
```

In the standard set-up, digits and text produced by “log-like operators” such as `\log` and `\max` are taken from the symbol font called `operators`. To change this situation so that these elements agree with the main text font—say, Computer

Modern Sans rather than Computer Modern Roman—you can issue the following commands:

```
\SetSymbolFont{operators}{normal}{OT1}{cmss}{m}{n}
\SetSymbolFont{operators}{bold} {OT1}{cmss}{bx}{n}
```

Symbol fonts with the names `symbols` and `largesymbols` play a unique rôle in TeX, and for this reason they need a special number of `\fontdimen` parameters associated with them. Thus, only specially prepared fonts can be used for these two symbol fonts. In principle one can add such parameters to any font at load time by using the third parameter of `\DeclareFontFamily` or the sixth parameter of `\DeclareFontShape`. Information on the special parameters for these symbol fonts can be found in Appendix G of [82].

### 7.10.8 Example: Defining your own .fd files

If you want to set up new (PostScript) fonts and create the necessary .fd files, you should follow the procedure explained earlier in this section. If `fontinst` [74] is used to generate the necessary font metric files, then the corresponding .fd files are automatically generated as well. However, an .fd file for a single font family is also easy to write by hand, once you know which font encoding is used. As an example, let's study the declaration file `t1bch.fd` for Bitstream Charter in the T1 encoding:

```
\ProvidesFile{t1bch.fd}[2001/06/04 font definitions for T1/bch.]
% Primary declarations
\DeclareFontFamily{T1}{bch}{}
\DeclareFontShape{T1}{bch}{m}{n}{<->bchr8t}{}
\DeclareFontShape{T1}{bch}{m}{sc}{<->bchrc8t}{}
\DeclareFontShape{T1}{bch}{m}{sl}{<->bchro8t}{}
\DeclareFontShape{T1}{bch}{m}{it}{<->bchri8t}{}
\DeclareFontShape{T1}{bch}{b}{n}{<->bchb8t}{}
\DeclareFontShape{T1}{bch}{b}{sc}{<->bchbc8t}{}
\DeclareFontShape{T1}{bch}{b}{sl}{<->bchbo8t}{}
\DeclareFontShape{T1}{bch}{b}{it}{<->bchbi8t}{}
% Substitutions
\DeclareFontShape{T1}{bch}{bx}{n}{<->ssub * bch/b/n}{}
\DeclareFontShape{T1}{bch}{bx}{sc}{<->ssub * bch/b/sc}{}
\DeclareFontShape{T1}{bch}{bx}{sl}{<->ssub * bch/b/sl}{}
\DeclareFontShape{T1}{bch}{bx}{it}{<->ssub * bch/b/it}{}
\endinput
```

The file starts with an identification line and then declares the font family and encoding (i.e., `bch` in `T1`) using `\DeclareFontFamily`—the arguments of this command should correspond to the name of the .fd file, except that by convention the encoding is in lowercase there. Then each combination of series and shape

is mapped to the name of a `.tfm` file. These fonts can and will be scaled to any desired size—hence the `<->` declarations on the `\DeclareFontShape` commands. The second part of the file sets up some substitutions for combinations for which no font is available (i.e., replacing the bold extended series with the bold series).

Assuming you have bought the additional Charter fonts (Black and BlackItalic), which are *not* available for free, then you may want to add the related declarations to the `.fd` file. Of course, one would first need to provide the appropriate virtual fonts (using, for example, `fontinst`) to emulate the T1 character set; fortunately, for many fonts these can be downloaded from the Internet.<sup>1</sup>

*Special license for .fd files*

In contrast to most other files in the L<sup>A</sup>T<sub>E</sub>X world, the usual license for `.fd` files allows their modification without renaming the files. However, you are normally not allowed to distribute such a modified file!

Another possible reason for producing your own `.fd` files might be the need to combine fonts from different font families and present them to L<sup>A</sup>T<sub>E</sub>X as a single new font family. For example, in 1954 Hermann Zapf designed the Aldus font family as a companion to his Palatino typeface (which was originally designed as a display typeface). As Aldus has no bold series, Palatino is a natural choice to use as a bold substitute. In the example below we combine Aldus (with old-style numerals) in its medium series with Palatino bold, calling the resulting “font family” `zasj`. We present only a fragment of a complete `.fd` file that enables us to typeset Example 7-10-1 on the facing page.

```
\ProvidesFile{t1zasj.fd}
[2003/10/12 font definitions for T1 Aldus/Palatino mix.]
\DeclareFontFamily{T1}{zasj}{}
% Medium series
\DeclareFontShape{T1}{zasj}{m}{n}{<->pasr9d}{}
\DeclareFontShape{T1}{zasj}{m}{sc}{<->pasrc9d}{}
\DeclareFontShape{T1}{zasj}{m}{it}{<->pasri9d}{}
\DeclareFontShape{T1}{zasj}{m}{sl}{<->ssub * pasj/m/it}{}
% Bold series
\DeclareFontShape{T1}{zasj}{b}{n}{<->pplb8t}{}
\DeclareFontShape{T1}{zasj}{b}{sc}{<->pplbc8t}{}
\DeclareFontShape{T1}{zasj}{b}{sl}{<->pplbo8t}{}
\DeclareFontShape{T1}{zasj}{b}{it}{<->pplbi8t}{}
```

To access this “pseudo-family” we have to select `zasj` in the T1 encoding. We also have to ensure that `\textbf` switches to bold and not to bold extended, as our `.fd` file does not provide any substitutions. All that can be automatically provided by writing a tiny package (named `fontmix.sty`) like this:

```
\ProvidesPackage{fontmix}[2003/10/12 T1 Aldus/Palatino mix.]
\RequirePackage[T1]{fontenc}
\renewcommand\rmdefault{zasj} \renewcommand\bfdefault{b}
```

<sup>1</sup>A good resource is Walter Schmidt's home page: <http://home.vr-web.de/~was/fonts.html>.

Thus, by loading `fontmix`, we get Aldus with Palatino Bold for headlines. In many cases such a mixture does not enhance your text, so do not mistake this example as a suggestion to produce arbitrary combinations.

## Zapf's Palatino and Aldus

This text is set in the typeface Aldus with matching *old-style* numerals '123456789'.

As a companion **bold face** Zapf's Palatino is selected.

```
\usepackage{fontmix}
% t1zasj.fd and fontmix.sty as defined above
\section*[Zapf's Palatino and Aldus]{This text is set in the typeface Aldus with
matching \emph{old-style} numerals '123456789'.
```

As a companion `\textbf{bold face}` Zapf's Palatino is selected.

### 7.10.9 The order of declaration

NFSS forces you to give all declarations in a specific order so that it can check whether you have specified all necessary information. If you declare objects in the wrong order, it will complain. Here are the dependencies that you have to obey:

- `\DeclareFontFamily` checks that the encoding scheme was previously declared with `\DeclareFontEncoding`.
- `\DeclareFontShape` checks that the font family was declared to be available in the requested encoding (`\DeclareFontFamily`).
- `\DeclareSymbolFont` checks that the encoding scheme is valid.
- `\SetSymbolFont` additionally ensures that the requested math version was declared (`\DeclareMathVersion`) and that the requested symbol font was declared (`\DeclareSymbolFont`).
- `\DeclareSymbolFontAlphabet` checks that the command name for the alphabet identifier can be used and that the symbol font was declared.
- `\DeclareMathAlphabet` checks that the chosen command name can be used and that the encoding scheme was declared.
- `\SetMathAlphabet` checks that the alphabet identifier was previously declared with `\DeclareMathAlphabet` or `\DeclareSymbolFontAlphabet` and that the math version and the encoding scheme are known.
- `\DeclareMathSymbol` makes sure that the command name can be used (i.e., is undefined or was previously declared to be a math symbol) and that the symbol font was previously declared.
- When the `\begin{document}` command is reached, NFSS makes some additional checks—for example, verifying that substitution defaults for every encoding scheme point to known font shape group declarations.

## 7.11 L<sup>A</sup>T<sub>E</sub>X's encoding models

For most users it will probably be sufficient to know that there exist certain input and output encodings and to have some basic knowledge about how to use them, as described in the previous sections. However, sometimes it is helpful to know the whole story in some detail, so as either to set up a new encoding or to better understand packages or classes that implement special features. So here is everything you always wanted to know about encodings in L<sup>A</sup>T<sub>E</sub>X.

We start by describing the general character data flow within the L<sup>A</sup>T<sub>E</sub>X system, deriving from that the base requirements for various encodings and the mapping between them. We then have a closer look at the internal representation model for character data within L<sup>A</sup>T<sub>E</sub>X, followed by a discussion of the mechanisms used to map incoming data via input encodings into that internal representation.

Finally, we explain how the internal representation is translated, via the output encodings, into the form required for the actual task of typesetting.

### 7.11.1 Character data within the L<sup>A</sup>T<sub>E</sub>X system

Document processing with the L<sup>A</sup>T<sub>E</sub>X system starts by interpreting data present in one or more source files. This data, which represents the document content, is stored in these files in the form of octets representing characters. To correctly interpret these octets, L<sup>A</sup>T<sub>E</sub>X (or any other program used to process the file, such as an editor) must know the encoding that was used when the file was written. In other words, it must know the mapping between abstract characters and the octets representing them.

With an incorrect mapping, all further processing will be flawed to some extent unless the file contains only characters of a subset common in both encodings.<sup>1</sup>

L<sup>A</sup>T<sub>E</sub>X makes one fundamental assumption at this stage: that (nearly) all characters of visible ASCII (decimal 32–126) are represented by the number that they have in the ASCII code table; see Table 7.31 on the next page.

There is both a practical and a T<sub>E</sub>Xnical reason for this assumption. The practical reason is that most 8-bit encodings in use today share a common 7-bit plane. The T<sub>E</sub>Xnical reason is to effectively<sup>2</sup> use T<sub>E</sub>X, the majority of the visible portion of ASCII needs to be processed as characters of category “letter”—since only char-

<sup>1</sup>As most encodings in the Western world share as a common subset a large fraction of the ASCII code (i.e., most of the 7-bit plane), documents consisting mainly of unaccented Latin characters are still understandable if viewed or processed in an encoding different from the one in which they were originally written. However, the more characters outside visible ASCII are used, the less comprehensible the text will become. A text can become completely unintelligible when, for instance, Greek or Russian documents are reprocessed under the assumption that the text is encoded in, say, the encoding for U.S.-Windows.

<sup>2</sup>At least this was true when this interface was being designed. These days, with computers being much faster than before, it would be possible to radically change the input method of T<sub>E</sub>X by basically disabling it altogether and parsing the input data manually—that is, character by character.

<i>Represented as Characters</i>	
Digits:	0 1 2 3 4 5 6 7 8 9
Lowercase letters:	a b c d e f g h i j k l m n o p q r s t u v w x y z
Uppercase letters:	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Punctuation:	. , ; : ? ! ‘ ’
Miscellaneous symbols:	* + - = ( ) [ ] / @
<i>Not Represented as Characters</i>	
<i>\TeX</i> syntax characters:	\$ ^ _ { } # & % \ ~
Missing in (some) OT1 fonts	< >   "

Table 7.31: LICR objects represented with single characters

acters with this category can be used in multiple-character command names in *\TeX*—or category “other”—since *\TeX* will not, for example, recognize the decimal digits as being part of a number if they do not have this category code.

When a character—or more exactly an 8-bit number—is declared to be of category “letter” or “other” in *\TeX*, then this 8-bit number will be transparently passed through *\TeX*. This means that in the output *\TeX* will typeset whatever symbol is in the font at the position addressed by that number.

A consequence of the assumption mentioned earlier is that fonts intended to be used for general text require that (most of) the visible ASCII characters are present in the font and are encoded according to the ASCII encoding. The exact list is given in Table 7.31.

All other 8-bit numbers (i.e., those outside visible ASCII) potentially being present in the input file are assigned a category code of “active”, which makes them act like commands inside *\TeX*. This allows *\TeX* to transform them via the input encodings to a form that we call the *\TeX* internal character representation (LICR).

Unicode’s UTF-8 encoding is handled similarly: the ASCII characters represent themselves, and the starting octets for multiple-byte representations act as active characters that scan the input for the remaining octets. The result will be turned into an object in the LICR, if it is mapped, or it will generate an error, if the given Unicode character is not mapped.

The most important characteristic of objects in the LICR is that the representation is 7-bit ASCII so that it is invariant to any input encoding change, because all input encodings are supposed to be transparent with respect to visible ASCII. This enables *\TeX*, for example, to write auxiliary files (e.g., *.toc* files) using the LICR representation and to read them back in a different context (and possibly different encoding) without any misinterpretations.

The purpose of the output (or font) encoding is then to map the internal character representations to glyph positions in the current font used for typesetting or, in some cases, to initiate more complex actions. For example, it might place an

*\TeX* internal  
character  
representation  
(LICR)

accent (present in one position in the current font) over some glyph (in a different position in the current font) to achieve a printed image of the abstract character represented by the command(s) in the internal character encoding.

Because the LICR encodes all possible characters addressable within  $\text{\LaTeX}$ , it is far larger than the number of characters that can be represented by a single  $\text{\TeX}$  font (which can contain a maximum of 256 glyphs). In some cases a character in the internal encoding can be rendered with a font by combining glyphs, such as accented characters mentioned above. However, when the internal character requires a special shape (e.g., the currency symbol “ $\text{\textcent}$ ”), there is no way to fake it if that glyph is not present in the font.

Nevertheless, the  $\text{\LaTeX}$  model for character encoding supports automatic mechanisms for fetching glyphs from different fonts so that characters missing in the current font will get typeset—provided a suitable additional font containing them is available, of course.

### 7.11.2 $\text{\LaTeX}$ 's internal character representation (LICR)

Technically speaking, text characters are represented internally by  $\text{\LaTeX}$  in one of three ways, each of which will be discussed in the following sections.

#### Representation as characters

A small number of characters are represented by “themselves”; for example, the Latin A is represented as the character “A”. Characters represented in this way are shown in Table 7.31 on the previous page. They form a subset of visible ASCII, and inside  $\text{\TeX}$  all of them are given the category code of “letter” or “other”. Some characters from the visible ASCII range are not represented in this way, either because they are part of the  $\text{\TeX}$  syntax<sup>1</sup> or because they are not present in all fonts. If one uses, for example, “<” in text, the current font encoding determines whether one gets < (T1) or perhaps a ; (OT1) in the printout.<sup>2</sup>

#### Representation with character sequences

$\text{\TeX}$ 's internal ligature mechanism supports the generation of new characters from a sequence of input characters. While this is actually a property of the font, some such sequences have been explicitly designed to serve as input shortcuts for characters that are otherwise difficult to address with most keyboards. Only a very few characters generated in this way are considered to belong to  $\text{\LaTeX}$ 's internal representation. These include the en dash and em dash, which are generated by

<sup>1</sup>The  $\text{\TeX}$  syntax knows a few more characters, such as \*[ ]. They play a dual rôle, also being used to represent the characters in straight text. sometimes problems arise trying to keep the two meanings apart. For example, a ] within an optional argument is only possible when it is hidden by a set of braces; otherwise,  $\text{\TeX}$  will think the optional argument has ended.

<sup>2</sup>This describes the situation in text. In math “<” has a well-defined meaning: “generate a less than relation symbol”.

the ligatures -- and ---, and the opening and closing double quotes, which are generated by ‘‘ and ’’ (the latter can usually also be represented by the single character "). While most fonts also implement !‘ and ?‘ to generate ; and , this feature is not universally available in all fonts. For this reason *all* such characters have an alternative internal representation as a command (e.g., \textendash or \textexclamdown).

#### Representation as “font-encoding-specific” commands

The other way to represent characters internally in L<sup>A</sup>T<sub>E</sub>X (and this covers the majority of characters) is with special L<sup>A</sup>T<sub>E</sub>X commands (or command sequences) that remain unexpanded when written to a file or when placed into a moving argument. These special commands are sometimes referred to as “font-encoding-specific commands” because their meaning depends on the font encoding current when L<sup>A</sup>T<sub>E</sub>X is ready to typeset them. Such commands are declared using special declarations, as discussed below. They usually require individual definitions for each font encoding. If no definition exists for the current encoding, either a default is used (if available) or an error message is presented to the user.

Technically, when the font encoding is changed at some point in the document, the definitions of the encoding-specific commands do not change immediately, as that would mean changing a large number of commands on the spot. Instead, these commands have been implemented in a such way that they notice, once they are used, if their current definition is no longer suitable for the font encoding in force. In such a case they call upon their counterparts in the current font encoding to do the actual work.

The set of “font-encoding-specific commands” is not fixed, but rather implicitly defined to be the union of all commands defined for individual font encodings. Thus, by adding new font encodings to L<sup>A</sup>T<sub>E</sub>X, new “font-encoding-specific commands” might emerge.

#### 7.11.3 Input encodings

Once the package `inputenc` is loaded (with or without options), the two declarations `\DeclareInputText` and `\DeclareInputMath` for mapping 8-bit input characters to L<sup>I</sup>C<sup>R</sup> objects become available. Their usage should be confined to input encoding files (described below), packages, or, if necessary, to the preamble of documents.

These commands take an 8-bit number as their first argument, which can be given as a decimal number (e.g., 239), octal number (e.g., '357), or hexadecimal notation (e.g., "EF). It is advisable to use decimal notation given that the characters ' or " might get special meanings in a language support package, such as shortcuts for accents, thereby preventing octal or hexadecimal notation from working correctly if packages are loaded in the wrong order.

```
\DeclareInputText{number}{LICR-object}
```

The `\DeclareInputText` command declares character mappings for use in text. Its second argument contains the encoding-specific command (or command sequence), that is the LICR object, to which the character number should be mapped. For instance,

```
\DeclareInputText{239}{\"i}
```

maps the number 239 to the encoding-specific representation of ï, which is `\\"i`. Input characters declared in this way cannot be used inside mathematical formulas.

```
\DeclareInputMath{number}{math-object}
```

If the number represents a character for use in mathematical formulas, then the declaration `\DeclareInputMath` must be used. For example, in the input encoding `cp437de` (German MS-DOS keyboard),

```
\DeclareInputMath{224}{\alpha}
```

associates the number 224 the command `\alpha`. Note that this declaration would make the key producing this number usable only in math-mode, as `\alpha` is not allowed elsewhere.

```
\DeclareUnicodeCharacter{hex-number}{LICR-object}
```

This declaration is available only if the option `utf8` is used. It maps Unicode numbers to LICR objects (i.e., characters usable in text). For example,

```
\DeclareUnicodeCharacter{00A3}{\textsterling}
\DeclareUnicodeCharacter{011A}{\v E}
\DeclareUnicodeCharacter{2031}{\textpertenthousand}
```

In theory, there should be only a single unique bidirectional mapping between the two name spaces, so that all such declarations could be already automatically made when the `utf8` option is selected. In practice, the situation is a little more complicated. For one, it is not sensible to automatically provide the whole table, because that would require a huge amount of `TEX`'s memory. Additionally, there are many Unicode characters for which no LICR object exists (so far), and conversely many LICR objects have no equivalents in Unicode.<sup>1</sup> The `inputenc` package solves that problem by loading only those Unicode mappings that correspond

<sup>1</sup>This is perhaps a surprising statement, but simply consider that, for example, accent commands like `\"` combined with some other character form a new LICR object, such as `\"d` (whether sensible or not). Many such combinations are not available in Unicode.

to the encodings used in a particular document (as far as they are known) and responds to any other request for a Unicode character with a suitable error message. It then becomes your task to either provide the right mapping information or, if necessary, load an additional font encoding.

As mentioned previously, the input encoding declarations can also be used in packages or in the preamble of a document. For this approach to work, it is important to load the `inputenc` package first, thereby selecting a suitable encoding. Subsequent input encoding declarations will act as a replacement for (or addition to) those being defined by the present input encoding.

There are two internal commands that you might see when using the `inputenc` package. The `\IeC` command is used internally by the `\DeclareInputText` declaration in certain circumstances. It ensures that when the encoding-specific command is written to a file, a space following it is not gobbled up when the file is read back in. This processing is handled automatically, so that a user never has to write this command. We mention it here because it might show up in `.toc` files or other auxiliary files.

The other command, `\@tabacckludge`, stands for “tabbing accent kludge”. It is (unfortunately) needed because the current version of L<sup>A</sup>T<sub>E</sub>X inherited an overloading of the commands `\=`, `\'`, and `\``, which normally denote certain accents (i.e., are encoding-specific commands), but have special meanings inside the tabbing environment. For this reason, mappings that involve any of these accents need to be encoded in a special way. If, for example, you want to map 232 to the character è which has the internal representation `\`e`, you should not write

```
\DeclareInputText{232}{\`e}
```

but rather

```
\DeclareInputText{232}{\@tabacckludge`e}
```

The latter form works everywhere, including inside a tabbing environment.

### Mapping to text and/or math

For technical as well as conceptual reasons, T<sub>E</sub>X makes a very strong distinction between characters usable in text and those usable in math. Except for the visible ASCII characters, commands that produce characters can normally be used in either text or math mode but not in both modes.

Unfortunately, for some keyboard keys it is not clear whether they should be regarded as generating characters for use in math or text. For example, should the key generating the character ± be mapped to `\textpm`, which is an encoding-specific command and thus can be used only in text, or should it be mapped to `\pm` and therefore be available only in math?

The early releases of the `inputenc` package used the following strategy: all keyboard keys available in standard T<sub>E</sub>X fonts for text (i.e., those encoded in either

OT1 or T1) were mapped to encoding-specific text commands, while the remaining keys got mapped to available math commands. But using a strategy solely driven by the availability of glyphs has the disadvantage that only users with a good knowledge of TeX internals could tell immediately whether using a key labeled, say “¾” or “³” would be allowed only in text or only in math.<sup>1</sup>

What can be done to resolve this situation gracefully? The approach of checking for the current mode, as used in babel’s `\textormath` command,

```
\ifmmode \ddots a\else \"a\fi
```

fails if such a construction is used in a math alignment structure (it selects the wrong part of the conditional and usually ends in an incomprehensible TeX error message). Fixing this problem by starting the above construction with `\relax` will prevent kerning and ligatures that may otherwise be present in a word. This is, in fact, a problem that is unsolvable in TeX. However, it can be solved if eTeX is used as the base formatter for L<sup>A</sup>T<sub>E</sub>X and as nowadays eTeX is available with nearly every TeX system, there are plans to make this program the basis for future maintenance releases of L<sup>A</sup>T<sub>E</sub>X.

At the time of this book’s writing, work on an extension of `inputenc` (based on eTeX) was under way. This proposed extension will automatically support all accessible keyboard characters in text and formulas. Once it becomes officially available, you will be able to comfortably typeset your formulas by simply adding the option `math` when loading the `inputenc` package.

### **Input encoding files for 8-bit encodings**

Input encodings are stored in files with the extension `.def`, where the base name is the name of the input encoding (e.g., `latin1.def`). Such files should contain only the commands described in the current section.

The file should start with a `\ProvidesFile` declaration describing the nature of the file. For example:

```
\ProvidesFile{latin1.def}[2000/07/01 v0.996 Input encoding file]
```

If there are mappings to encoding-specific commands that might not be available unless additional packages are loaded, one could declare defaults for them using `\ProvideTextCommandDefault`. For example:

```
\ProvideTextCommandDefault{\textonehalf}{\ensuremath{\frac{1}{2}}}
\ProvideTextCommandDefault{\textcent}{\TextSymbolUnavailable\textcent}
```

The command `\TextSymbolUnavailable`, used above, issues a warning indicating that a certain character is not available with the currently used fonts. This can

<sup>1</sup>In the first releases of the `inputenc` package, “¾” was a text glyph but “³” a was math glyph—comprehensible?

be useful as a default—that is, when such characters are available only if special fonts are loaded and no suitable way exists to fake the characters with existing characters (as was possible for a default for `\textonehalf` above).

The remaining part of the file should consist only of input encoding declarations using `\DeclareInputText` or `\DeclareInputMath`. As mentioned earlier, the use of the latter command, though allowed, is discouraged. No other commands should be used inside an input encoding file; in particular, no commands that prevent reading the file several times (e.g., `\newcommand`), as the encoding files are often loaded several times in a single document!

### Input mapping files for UTF-8

As mentioned earlier, the mapping from Unicode to L<sup>I</sup>C<sup>R</sup> objects is not done in a single large mapping file, but rather organized in a way that enables L<sup>A</sup>T<sub>E</sub>X to load only those mappings that are relevant for the font encodings used in the current document. This is done by attempting to load for each encoding *(name)* a file *(name)enc.dfu* that, if it exists, contains the mapping information for those Unicode characters provided by that particular encoding. Other than a number of `\DeclareUnicodeCharacter` declarations, such files should contain only a `\ProvidesFile` line.

As different font encodings often provide to a certain extent the same characters, it is quite common for declarations for the same Unicode character to be found in different *.dfu* files. It is, therefore, very important that these declarations in different files be identical (which in theory they should be anyway, but...). Otherwise, the declaration loaded last will survive, which may be a different one from document to document.

So anyone who wants to provide a new *.dfu* file for some encoding that was previously not covered should carefully check the existing definitions in *.dfu* files for related encodings. Standard files provided with `inputenc` are guaranteed to have uniform definition—they are, in fact, all generated from a single list that is suitably split up. A full list of currently existing mappings can be found in the file `utf8enc.dfu`.

#### 7.11.4 Output encodings

As we learned earlier, output encodings define the mapping from the L<sup>I</sup>C<sup>R</sup> to the glyphs (or constructs built from glyphs) available in the fonts used for typesetting. These mappings are referenced inside L<sup>A</sup>T<sub>E</sub>X by two- or three-letter names (e.g., OT1 and T3). We say that a certain font is in a certain encoding if the mapping corresponds to the positions of the glyphs in the font in question. So what are the exact components of such a mapping?

Characters internally represented by ASCII characters are simply passed on to the font. In other words, T<sub>E</sub>X uses the ASCII code to select a glyph from the current font. For example, the character “A” with ASCII code 65 will result in typesetting

the glyph in position 65 in the current font. This is why  $\text{\LaTeX}$  requires that fonts for text contain all such ASCII letters in their ASCII code positions, as there is no way to interact with this basic  $\text{T}_{\text{E}}\text{X}$  mechanism (other than to disable it and do everything “manually”). Thus, for visible ASCII, a one-to-one mapping is implicitly present in all output encodings.

Characters internally represented as sequences of ASCII characters (e.g., “–”), are handled as follows: when the current font is first loaded,  $\text{T}_{\text{E}}\text{X}$  is informed that the font contains a number of so-called ligature programs. These define certain character sequences that are not to be typeset directly but rather to be replaced<sup>1</sup> by some other glyphs from the font (the exact position of each replacement glyph is font dependent and not important otherwise). For example, when  $\text{T}_{\text{E}}\text{X}$  sees “–” in the input (i.e., ASCII code 45 twice), a ligature program might direct it to use the glyph in position 123 instead (which then would hold the glyph “‐”). Again, no interaction with this mechanism is possible. Some such ligatures are present for purely aesthetic reasons and may or may not be available in certain fonts (e.g., ff generating “ff” rather than “ff”). Others are supposed to be implemented for a certain encoding (e.g., “–” producing  $\text{\textemdash}$ ).

Nevertheless, the bulk of the internal character representation consists of “font-encoding-specific” commands. They are mapped using the declarations described below. All declarations have the same structure in their first two arguments: the font-encoding-specific command (or the first component of it, if it is a command sequence), followed by the name of the encoding. Any remaining arguments will depend on the type of declaration.

Thus, an encoding XYZ is defined by a bunch of declarations all having the name XYZ as their second argument. Of course, to be of any use, some fonts must be encoded in that encoding. In fact, the development of font encodings is normally done the other way around—namely, someone starts with an existing font and then provides appropriate declarations for using it. This collection of declarations is then given a suitable name, such as OT1. In the next section, we will take the font `ecrm1000`, shown in Table 7.32 on the facing page, whose font encoding is called T1 in  $\text{\LaTeX}$ , and build appropriate declarations to access the glyphs from a font encoded in this way. The blue characters in this table are those that have to be present in the same positions in every text encoding, as they are transparently passed through  $\text{T}_{\text{E}}\text{X}$ .

### Output encoding files

Like input encoding files, output encoding files are identified by the extension `.def`. However, the base name of the file is slightly more structured: the name of the encoding in lowercase letters, followed by the letters `enc` (e.g., `t1enc.def` for the T1 encoding).

<sup>1</sup>The actions carried out by a font ligature program can, in fact, be far more complex, but for the purpose of our discussion here this simplified view is appropriate. For an in-depth discussion, see Knuth’s paper on virtual fonts [91].

	'0	'1	'2	'3	'4	'5	'6	'7	
'00x	'	'	'^	'~	'"	'``	'`	'`'	'0x
'01x	'`	'-	'`.	'`>	'`‘	'`‘,	'`‘<	'`‘>	
'02x	'`"	'`"	'`,,	'`⟨`	'`⟩`	'`—	'`—	'`—	'1x
'03x	'`o	'`1	'`J	'`ff	'`fi	'`fl	'`ffi	'`fF	
'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	'`Ā`	'`Ā`	'`Ć`	'`Ć`	'`Đ`	'`Ě`	'`Ē`	'`Ğ`	'8x
'21x	'`Ł`	'`Ł`	'`Ń`	'`Ń`	'`Ń`	'`DJ`	'`Ó`	'`Ŕ`	
'22x	'`Ř`	'`Ś`	'`Ś`	'`Ś`	'`Ť`	'`Ť`	'`Ů`	'`Ů`	'9x
'23x	'`Ŷ`	'`Ž`	'`Ž`	'`Ž`	'`IJ`	'`Í`	'`đ`	'`§`	
'24x	'`ă`	'`ä`	'`ć`	'`č`	'`đ`	'`ě`	'`ę`	'`ğ`	'Ax
'25x	'`í`	'`ł`	'`ł`	'`ń`	'`ň`	'`յ`	'`ő`	'`ŕ`	
'26x	'`ř`	'`ś`	'`ś`	'`ś`	'`ť`	'`ť`	'`ú`	'`ú`	'Bx
'27x	'`ÿ`	'`ž`	'`ž`	'`ž`	'`ij`	'`í`	'`ł`	'`ł`	
'30x	'`À`	'`Á`	'`Â`	'`Ã`	'`Ä`	'`Å`	'`Æ`	'`Ç`	'Cx
'31x	'`È`	'`É`	'`Ê`	'`Ë`	'`Ì`	'`Í`	'`Î`	'`Ï`	
'32x	'`Đ`	'`Ñ`	'`Ò`	'`Ó`	'`Ô`	'`Ö`	'`Œ`	'`œ`	'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`	'`ö`	'`ö`	'`œ`	'Fx
'37x	'`ø`	'`ù`	'`ú`	'`û`	'`ü`	'`ý`	'`þ`	'`ß`	
	'`8`	'`9`	'`A`	'`B`	'`C`	'`D`	'`E`	'`F`	

Characters marked in blue need to be present (in the same positions) in every text encoding, as they are transparently passed through T<sub>E</sub>X.

Table 7.32: Glyph chart for a T1-encoded font (ecrm1000)

Such files should contain only the declarations described in the current section. As output encoding files might be read several times by L<sup>A</sup>T<sub>E</sub>X, it is particularly important to adhere to this rule strictly and to refrain from using, for example, `\newcommand`, which prevents reading such a file multiple times!

For identification purposes an output encoding file should start with a `\ProvidesFile` declaration describing the nature of the file. For example:

```
\ProvidesFile{t1enc.def}[2001/06/05 v1.94 Standard LaTeX file]
```

To be able to declare any encoding-specific commands for a particular encoding, we first have to make this encoding known to L<sup>A</sup>T<sub>E</sub>X. This is achieved via the `\DeclareFontEncoding` declaration. At this point it is also useful to declare the default substitution rules for the encoding with the help of the command `\DeclareFontSubstitution`; both declarations are described in detail in Section 7.10.5 starting on page 430.

```
\DeclareFontEncoding{T1}{}{}
\DeclareFontSubstitution{T1}{cmr}{m}{n}
```

Having introduced the T1 encoding in this way to L<sup>A</sup>T<sub>E</sub>X, we can now proceed with declaring how font-encoding-specific commands should behave in that encoding.

```
\DeclareTextSymbol{LICR-object}{encoding}{slot}
```

Perhaps the simplest form of declaration is the one for text symbols, where the internal representation can be directly mapped to a single glyph in the target font. This is handled by the `\DeclareTextSymbol` declaration, whose third argument—the font position—can be given as a decimal, hexadecimal, or octal number. For example,

```
\DeclareTextSymbol{\ss}{T1}{255}
\DeclareTextSymbol{\AE}{T1}{'306} % font position as octal number
\DeclareTextSymbol{\ae}{T1}{`E6} % ... as hexadecimal number
```

declare that the font-encoding-specific commands `\ss`, `\AE`, and `\ae` should be mapped to the font (decimal) positions 255, 198, and 230, respectively, in a T1-encoded font. As mentioned earlier, it is safest to use decimal notation in such declarations, even though octal or hexadecimal values are often easier to identify in glyph charts like the one on the previous page. Mixing them like we did in the example above is certainly bad style. All in all, there are 49 such declarations for the T1 encoding.

```
\DeclareTextAccent{LICR-accent}{encoding}{slot}
```

Often fonts contain diacritical marks as individual glyphs to allow the production of accented characters by combining such a diacritical mark with some other

glyph. Such accents (as long as they are to be placed on top of other glyphs) are declared using the `\DeclareTextAccent` command; the third argument *slot* is the position of the diacritical mark in the font. For example,

```
\DeclareTextAccent{\"}{T1}{4}
```

defines the “umlaut” accent. From that point onward, an internal representation such as `\"a` has the following meaning in the T1 output encoding: typeset “ä” by placing the accent in position 4 over the glyph in position 97 (the ASCII code of the character a). In fact, such a declaration implicitly defines a huge range of internal character presentations—that is, anything of the type `\"(base-glyph)`, where `(base-glyph)` is something defined via `\DeclareTextSymbol` or any ASCII character belonging to the LICR, such as “a”.

Even those combinations that do not make much sense, such as `\\"P` (i.e., pilcrow sign with umlaut ¶) conceptually become members of the set of font-encoding-specific commands in this way. There are a total of 11 such declarations in the T1 encoding.

```
\DeclareTextComposite
  {LICR-accent}{encoding}{simple-LICR-object}{slot}
```

The glyph chart on page 449 contains a large number of accented characters as individual glyphs—for example, “ä” in position ’344 octal. Thus, in T1 the encoding-specific command `\"a` should not result in placing an accent over the character “a” but instead should directly access the glyph in that position of the font. This is achieved by the declaration

```
\DeclareTextComposite{\"}{T1}{a}{228}
```

which states that the encoding-specific command `\"a` results in typesetting the glyph 228, thereby disabling the accent declaration above. For all other encoding-specific commands starting with `\\"`, the accent declaration remains in place. For example, `\\"b` will produce a “b” by placing an accent over the base character b.

The third argument, *simple-LICR-object*, should be a single letter, such as “a”, or a single command, such as `\j` or `\oe`. There are 110 such composites declared for the T1 encoding.

```
\DeclareTextCompositeCommand
  {LICR-object}{encoding}{simple-LICR-object}{code}
```

Although not used for the T1 encoding, there also exists a more general variant of `\DeclareTextComposite` that allows arbitrary code in place of a slot position. This is, for example, used in the OT1 encoding to lower the ring accent over the

“A” compared to the way it would be typeset with TeX’s `\accent` primitive. The accents over the “i” are also implemented using this form of declaration:

```
\DeclareTextCompositeCommand{\'{0T1}{i}{\@tabacckludge'`i}
\DeclareTextCompositeCommand{\^{\0T1}{i}{\^`i}}
```

What have we not covered for the T1 encoding? A number of diacritical marks are not placed on top of other characters but are placed somewhere below them. There is no special declaration form for such marks, as the actual placement usually involves low-level TeX code. Instead, the generic `\DeclareTextCommand` declaration can be used for this purpose.

```
\DeclareTextCommand{LICR-object}{encoding}[num][default]{code}
```

For example, the “underbar” accent `\b` in the T1 encoding is defined with the following wonderful piece of prose:

```
\DeclareTextCommand{\b}{T1}[1]
{\hmode\bgroup\o@align{\relax#1\crcr\hidewidth\sh@ft{29}%
\vbox to.2ex{\hbox{\char9}\vss}\hidewidth}\egroup}
```

Without going into detail about what the code precisely means, we can see that the `\DeclareTextCommand` is similar in structure to `\newcommand`. That is, it has an optional *num* argument denoting the number of arguments (one here), a second optional *default* argument (not present here), and a final mandatory argument containing the code in which it is possible to refer to the argument(s) using #1, #2, and so on. T1 has four such declarations, for `\b`, `\c`, `\d`, and `\k`.

`\DeclareTextCommand` can also be used to build font-encoding-specific commands consisting of a single control sequence. In this case it is used without optional argument, thus defining a command with zero arguments. For example, in T1 there is no glyph for a % sign, but there exists a strange little “`\%`” in position ’30, which, if placed directly behind a %, will give the appropriate glyph. Thus, we can write

```
\DeclareTextCommand{\textperthousand}{T1}{\%\char 24 }
\DeclareTextCommand{\textpertenthousand}{T1}{\%\char 24\char 24 }
```

This discussion has now covered all commands needed to declare the font-encoding-specific commands for a new encoding. As mentioned earlier, only these commands should appear in encoding definition files.

### Output encoding defaults

What happens if an encoding-specific command is used for which there is no declaration in the current font encoding? In that case one of two things might happen: either TeX has a default definition for the LICR object, in which case this

default is used, or the user gets an error message stating that the requested LICR object is unavailable in the current encoding. There are a number of ways to set up defaults for LICR objects.

```
\DeclareTextCommandDefault{\LICR-object}[num][default]{code}
```

The `\DeclareTextCommandDefault` command provides the default definition for an *LICR-object* that is to be used whenever there is no specific setting for the object in the current encoding. Such default definitions can, for example, fake a certain character. For instance, `\textregistered` has a default definition in which the character is built from two others, like this:

```
\DeclareTextCommandDefault{\textregistered}{\textcircled{\texttt{scshape r}}}
```

Technically, the default definitions are stored as an encoding with the name “?”. While you should not rely on this fact, as the implementation might change in the future, it means that you cannot declare an encoding with this name.

```
\DeclareTextSymbolDefault{\LICR-object}{encoding}
```

In most cases, a default definition does not require coding but simply directs L<sup>A</sup>T<sub>E</sub>X to pick up the character from some encoding in which it is known to exist. The `textcomp` package, for example, consists of a large number of default declarations that all point to the TS1 encoding. Consider the following declaration:

```
\DeclareTextSymbolDefault{\texteuro}{TS1}
```

The `\DeclareTextSymbolDefault` command can, in fact, be used to define the default for any LICR object without arguments, not just those that have been declared with the `\DeclareTextSymbol` command in other encodings.

```
\DeclareTextAccentDefault{\LICR-accent}{encoding}
```

A similar declaration exists for LICR objects that take one argument, such as accents (which gave this declaration its name). This form is again usable for any LICR object with one argument. The L<sup>A</sup>T<sub>E</sub>X kernel, for example, contains quite a number of declarations of the type:

```
\DeclareTextAccentDefault{\"}{OT1}
\DeclareTextAccentDefault{\t}{OML}
```

This means that if the `\"` is not defined in the current encoding, then use the one from an OT1-encoded font. Likewise, if you need a tie accent, pick up one from OML<sup>1</sup> if nothing better is available.

<sup>1</sup>OML is a math font encoding, but it contains this text accent mark.

```
\ProvideTextCommandDefault{LICR-object} [num] [default] {code}
```

With the `\ProvideTextCommandDefault` declaration a different kind of default can be “provided”. As the name suggests, it does the same job as the declaration `\DeclareTextCommandDefault`, except that the default is provided only if no default has been defined before. This is mainly used in input encoding files to provide some sort of trivial defaults for unusual LICR objects. For example:

```
\ProvideTextCommandDefault{\textonequarter}{\ensuremath{\frac{1}{4}}}
\ProvideTextCommandDefault{\textcent}{\TextSymbolUnavailable\textcent}
```

Packages like `textcomp` can then replace such definitions with declarations pointing to real glyphs. Using `\Provide..` instead of `\Declare..` ensures that a better default is not accidentally overwritten if the input encoding file is read.

```
\UndeclareTextCommand{LICR-object}{encoding}
```

In some cases an existing declaration needs to be removed to ensure that a default declaration is used instead. This task can be carried out by the `\UndeclareTextCommand` command. For example, the `textcomp` package removes the definitions of `\textdollar` and `\textsterling` from the OT1 encoding because not every OT1-encoded font actually has these symbols.<sup>1</sup>

```
\UndeclareTextCommand{\textsterling}{OT1}
\UndeclareTextCommand{\textdollar}{OT1}
```

Without this removal, the new default declarations to pick up the symbols from TS1 would not be used for fonts encoded with OT1.

```
\UseTextSymbol{encoding}{LICR-object}
\UseTextAccent{encoding}{LICR-object}{simple-LICR-object}
```

The action hidden behind the declarations `\DeclareTextSymbolDefault` and `\DeclareTextAccentDefault` is also available for direct use. Assume, for example, that the current encoding is U. In that case,

```
\UseTextSymbol{OT1}{\ss}
\UseTextAccent{OT1}{'}{a}
```

has the same effect as entering the code below. Note in particular that the “a” is typeset in encoding U—only the accent is taken from the other encoding.

```
{\fontencoding{OT1}\selectfont\ss}
{\fontencoding{OT1}\selectfont'\{\fontencoding{U}\selectfont a\}}
```

<sup>1</sup>This is one of the deficiencies of the old T<sub>E</sub>X encodings; besides missing accented glyphs, they are not even identical from one font to another.

### A listing of standard LICR objects

Table 7.33 provides a comprehensive overview of the L<sup>A</sup>T<sub>E</sub>X internal representations available with the three major encodings for Latin-based languages: OT1 (the original T<sub>E</sub>X text font encoding), T1 (the L<sup>A</sup>T<sub>E</sub>X standard encoding, also known as Cork encoding), and LY1 (an alternate 8-bit encoding proposed by Y&Y). In addition, it shows all LICR objects declared by TS1 (the L<sup>A</sup>T<sub>E</sub>X standard text symbol encoding) provided by loading the `textcomp` package.

The first column of the table shows the LICR object names alphabetically sorted, indicating which LICR objects act like accents. The second column shows a glyph representation of the object.

The third column describes whether the object has a default declaration. If an encoding is listed, it means that by default the glyph is being fetched from a suitable font in that encoding; `constr.` means that the default is produced from low-level T<sub>E</sub>X code; if the column is empty it means that no default is defined for this LICR object. In the last case a “Symbol unavailable” error is returned when you use it in an encoding for which it has no explicit definition. If the object is an alias for some other LICR object, we list the alternative name in this column.

Columns four through seven show whether an object is available in the given encoding. Here **X** means that the object is natively available (as a glyph) in fonts with that encoding, **O** means that it is available through the default for all encodings, and `constr.` means that it is generated from several glyphs, accent marks, or other elements. If the default is fetched from TS1, the LICR object is available only if the `textcomp` package is loaded.

Table 7.33: Standard LICR objects

LICR Object	Glyph	Default from	OT1	T1	LY1	TS1
ABC..XYZ (Uppercase letters)	ABC..XYZ		X	X	X	
abc..xyz (Lowercase letters)	abc..xyz		X	X	X	
0123..9 (Digits)	0123..9		X	X	X	X
. , / (Punctuation)	. , /		X	X	X	X
; : ? ! " ' (Punctuation cont.)	; : ? ! " '		X	X	X	
*+-=()[] (Misc)	*+-=()[]		X	X	X	
# & %	#&%		X	X	X	
\ " (accent)	"	OT1	X	X	X	
\ "A	Ä		constr.	X	X	
\ "E	Ë		constr.	X	X	
\ "I	Ï		constr.	X	X	
\ "O	Ö		constr.	X	X	
\ "U	Ü		constr.	X	X	
\ "Y	Ÿ		constr.	X	X	
\ "a	ä		constr.	X	X	
<b>X</b> defined in encoding		<b>O</b> defined via default				

LICR Object	Glyph	Default from	OT1	T1	LY1	TS1
\"e	ë			constr.	x	x
\\"i	ï			constr.	x	x
\\"i	(alias) ï	\\"i		constr.	x	x
\\"o	ö			constr.	x	x
\\"u	ü			constr.	x	x
\\"y	ÿ			constr.	x	x
\\$	(alias) \$	\textdollar	○	x	x	x
\'	(accent) ‘	OT1	x	x	x	
\'A	Á			constr.	x	x
\'C	Ć			constr.	x	constr.
\'E	É			constr.	x	x
\'I	Í			constr.	x	x
\'L	Ĺ			constr.	x	constr.
\'N	Ń			constr.	x	constr.
\'O	Ó			constr.	x	x
\'R	Ŕ			constr.	x	constr.
\'S	Ś			constr.	x	constr.
\'U	Ú			constr.	x	x
\'Y	Ý			constr.	x	x
\'Z	Ž			constr.	x	constr.
\'a	á			constr.	x	x
\'c	ć			constr.	x	constr.
\'e	é			constr.	x	x
\'i	(alias) í	\'i		constr.	x	x
\'l	í			constr.	x	constr.
\'n	ń			constr.	x	constr.
\'o	ó			constr.	x	x
\'r	ŕ			constr.	x	constr.
\'s	ś			constr.	x	constr.
\'u	ú			constr.	x	x
\'y	ý			constr.	x	x
\'z	ž			constr.	x	constr.
\`.	(accent) ‘	OT1	x	x	x	
\`I	í			constr.	x	constr.
\`Z	ž			constr.	x	constr.
\`i	i			x	x	constr.
\`i	(alias) i	\`i		x	x	constr.
\`z	ž			constr.	x	constr.
\`=	(accent) ‘	OT1	x	x	x	

x defined in encoding      ○ defined via default

LICR Object	Glyph	Default from	OT1	T1	LY1	TS1
\AE	Æ	OT1	✗	✗	✗	
\DH	Đ			✗	✗	
\DJ	Đ			✗		
\H	(accent)	~	OT1	✗	✗	✗
\H o	Ö			constr.	✗	constr.
\H U	Ü			constr.	✗	constr.
\H o	ő			constr.	✗	constr.
\H u	ű			constr.	✗	constr.
\L	Ł	OT1	✗	✗	✗	
\NG	Ŋ				✗	
\Ø	Ø	OT1	✗	✗	✗	
\OE	Œ	OT1	✗	✗	✗	
\P	(alias)	¶	\textparagraph	○	○	✗
\S	(alias)	§	\textsection	○	✗	✗
\SS	SS	constr.	○	✗	○	
\TH	Þ			✗	✗	
\^	(accent)	^	OT1	✗	✗	✗
\^A	Â			constr.	✗	✗
\^E	Ê			constr.	✗	✗
\^I	Î			constr.	✗	✗
\^O	Ô			constr.	✗	✗
\^U	Û			constr.	✗	✗
\^a	â			constr.	✗	✗
\^e	ê			constr.	✗	✗
\^i	î		\^i	constr.	✗	✗
\^o	ô			constr.	✗	✗
\^u	û			constr.	✗	✗
\_	(alias)	_	\textunderscore	○	✗	✗
\`	(accent)	`	OT1	✗	✗	✗
\`A	À			constr.	✗	✗
\`E	È			constr.	✗	✗
\`I	Ì			constr.	✗	✗
\`O	Ò			constr.	✗	✗
\`U	Ù			constr.	✗	✗
\`a	à			constr.	✗	✗
\`e	è			constr.	✗	✗
\`i	ì		\`i	constr.	✗	✗
\`o	ò			constr.	✗	✗

✗ defined in encoding

○ defined via default

LICR Object	Glyph	Default from	OT1	T1	LY1	TS1
\`u	�			x	x	
\ae	æ	OT1	x	x	x	
\b (accent)	-	OT1	x	x	x	
\c (accent)	,,	OT1	x	x	x	
\c C	�			x	x	
\c S	�			x	constr.	
\c T	�			x	constr.	
\c c	�			x	x	
\c s	�			x	constr.	
\c t	�			x	constr.	
\capitalacute (accent)	'	TS1	o	o	o	x
\capitalcaron (accent)	ˇ	TS1	o	o	o	x
\capitaldieresis (accent)	�	TS1	o	o	o	x
\capitalgrave (accent)	ˋ	TS1	o	o	o	x
\capitalmacron (accent)	ˉ	TS1	o	o	o	x
\capitalogonek (accent)	�	TS1	o	o	o	x
\capitalring (accent)	�	TS1	o	o	o	x
\capitaltilde (accent)	˜	TS1	o	o	o	x
\copyright (alias)	�	\textcopyright	o	o	o	x
\d (accent)	.	OT1	x	x	x	
\dag (alias)	†	\textdagger	o	o	x	x
\ddag (alias)	‡	\textdaggerdbl	o	o	x	x
\dh	�			x	x	
\dj	�			x		
\dots (alias)	...	\textellipsis	o	o	x	
\guillemotleft	«	OT1	x	x	x	
\guillemotright	»	OT1	x	x	x	
\guilsinglleft	�	OT1	x	x	x	
\guilsinglright	�	OT1	x	x	x	
\i	�	OT1	x	x	x	
\j	�	OT1	x	x	x	
\k (accent)	�			x	x	
\k A	�			x	constr.	
\k E	�			x	constr.	
\k O	�			x		
\k a	�			x	constr.	
\k e	�			x	constr.	
\k o	�			x		
\l	�	OT1	x	x	x	
\ng	�			x		

x defined in encoding

o defined via default

LICR Object	Glyph	Default from	OT1	T1	LY1	TS1
\o	ø	OT1	x	x	x	
\oe	œ	OT1	x	x	x	
\pounds (alias)	£	\textsterling	○	x	x	x
\quotedblbase	"			x	x	
\quotesinglbase	,			x	x	
\r (accent)	°	OT1	x	x	x	
\r A	Å		x	x	x	
\r U	Ü		constr.	x	constr.	
\r a	å		constr.	x	x	
\r u	ü		constr.	x	constr.	
\ss	ß	OT1	x	x	x	
\t (accent)	~	OML	x	x	○	
\textacutedbl	"	TS1	○	○	○	x
\textascendercompwordmark	invisible	TS1	○	○	○	x
\textasciacute	'	TS1	○	○	○	x
\textascibreve	ˇ	TS1	○	○	○	x
\textasciicaron	ˇ	TS1	○	○	○	x
\textasciicircum	^	constr.	○	x	x	
\textasciidieresis	˝	TS1	○	○	○	x
\textasciigrave	ˋ	TS1	○	○	○	x
\textasciimacron	—	TS1	○	○	○	x
\textasciitilde	˜	constr.	○	x	x	
\textasteriskcentered	*	OMS/TS1	○	○	○	x
\textbackslash	\	OMS	○	x	x	x
\textbaht	฿	TS1	○	○	○	x
\textbar		OMS	○	x	x	
\textbardbl		TS1	○	○	○	x
\textbigcircle	○	TS1	○	○	○	x
\textblank	⠄	TS1	○	○	○	x
\textborn	⠄⠄	TS1	○	○	○	x
\textbraceleft	{	OMS	○	x	x	
\textbraceright	}	OMS	○	x	x	
\textbrokenbar	⠄⠄⠄	TS1	○	○	x	x
\textbullet	•	OMS/TS1	○	○	x	x
\textcapitalcompwordmark	invisible	TS1	○	○	○	x
\textcelsius	°C	TS1	○	○	○	x
\textcent	¢	TS1	○	○	x	x
\textcentoldstyle	¢	TS1	○	○	○	x
\textcircled (accent)	○	OMS/TS1	○	○	○	x
\textcircledP	®	TS1	○	○	○	x

x defined in encoding

○ defined via default

LICR Object	Glyph	Default from	OT1	T1	LY1	TS1
\textcolonmonetary	₡	TS1	○	○	○	✗
\textcompwordmark	invisible	constr.	○	✗	○	
\textcopyleft	⌚	TS1	○	○	○	✗
\textcopyright	⌚	constr./TS1	○	○	○	✗
\textcurrency	¤	TS1	○	○	✗	✗
\textdagger	†	OMS/TS1	○	○	✗	✗
\textdaggerdbl	‡	OMS/TS1	○	○	✗	✗
\textdblyhyphen	=	TS1	○	○	○	✗
\textdblyhyphenchar	=	TS1	○	○	○	✗
\textdegree	°	TS1	○	○	✗	✗
\textdied	†	TS1	○	○	○	✗
\textdiscount	%	TS1	○	○	○	✗
\textdiv	÷	TS1	○	○	○	✗
\textdivorced	օօ	TS1	○	○	○	✗
\textdollar	\$	OT1/TS1	○	✗	✗	✗
\textdollaroldstyle	\$	TS1	○	○	○	✗
\textdong	đ	TS1	○	○	○	✗
\textdownarrow	↓	TS1	○	○	○	✗
\texteightoldstyle	۸	TS1	○	○	○	✗
\textellipsis	...	constr.	○	○	✗	
\textemdash	—	OT1	✗	✗	✗	
\textendash	—	OT1	✗	✗	✗	
\textestimated	℮	TS1	○	○	○	✗
\texteuro	€	TS1	○	○	○	✗
\textexclamdown	¡	OT1	✗	✗	✗	
\textfiveoldstyle	᳚	TS1	○	○	○	✗
\textflorin	ƒ	TS1	○	○	✗	✗
\textfouroldstyle	᳚	TS1	○	○	○	✗
\textfractionsolidus	/	TS1	○	○	○	✗
\textgravedbl	“	TS1	○	○	○	✗
\textgreater	>	OML	○	✗	✗	
\textguarani	₲	TS1	○	○	○	✗
\textinterrobang	‽	TS1	○	○	○	✗
\textinterrobangdown	‽	TS1	○	○	○	✗
\textlangle	(	TS1	○	○	○	✗
\textlbrackdbl	॥	TS1	○	○	○	✗
\textleaf	Ѡ	TS1	○	○	○	✗
\textleftarrow	←	TS1	○	○	○	✗
\textless	<	OML	○	✗	✗	
\textlira	£	TS1	○	○	○	✗

✗ defined in encoding

○ defined via default

LICR Object	Glyph	Default from	OT1	T1	LY1	TS1
\textlnot	¬	TS1	○	○	○	✗
\textlquill	{	TS1	○	○	○	✗
\textmarried	ø	TS1	○	○	○	✗
\textmho	Ը	TS1	○	○	○	✗
\textminus	—	TS1	○	○	○	✗
\textmu	μ	TS1	○	○	✗	✗
\textmusicalnote	♪	TS1	○	○	○	✗
\textnaira	₦	TS1	○	○	○	✗
\textnineoldstyle	⁹	TS1	○	○	○	✗
\textnumero	№	TS1	○	○	○	✗
\textogonekcentered (accent)	‘			✗		
\textohm	Ω	TS1	○	○	○	✗
\textonehalf	½	TS1	○	○	✗	✗
\textoneoldstyle	۱	TS1	○	○	○	✗
\textonequarter	¼	TS1	○	○	✗	✗
\textonesuperior	۱	TS1	○	○	○	✗
\textopenbullet	◦	TS1	○	○	○	✗
\textordfeminine	؂	constr./TS1	○	○	✗	✗
\textordmasculine	؂	constr./TS1	○	○	✗	✗
\textparagraph	¶	OMS/TS1	○	○	✗	✗
\textperiodcentered	·	OMS/TS1	○	○	✗	✗
\textpertenthousand	%oo	TS1	○	✗	○	✗
\textperthousand	%o	TS1	○	○	✗	✗
\textpeso	₱	TS1	○	○	○	✗
\textpilcrow	¶	TS1	○	○	○	✗
\textpm	±	TS1	○	○	○	✗
\textquestiondown	՞	OT1	✗	✗	✗	
\textquotedbl	“			✗	✗	
\textquotedblleft	“	OT1	✗	✗	✗	
\textquotedblright	”	OT1	✗	✗	✗	
\textquotelleft	‘	OT1	✗	✗	✗	
\textquoteright	’	OT1	✗	✗	✗	
\textquotesingle	‘	TS1	○	○	○	✗
\textquotestraightbase	‘	TS1	○	○	○	✗
\textquotestraightdblbase	”	TS1	○	○	○	✗
\textrangle	》	TS1	○	○	○	✗
\textrbrackdbl	]]	TS1	○	○	○	✗
\textrecipe	R	TS1	○	○	○	✗
\textreferencemark	*	TS1	○	○	○	✗
\textregistered	®	constr./TS1	○	○	✗	✗

✗ defined in encoding      ○ defined via default

LICR Object	Glyph	Default from	OT1	T1	LY1	TS1
\textrightarrow	→	TS1	○	○	○	✗
\textrquill	}	TS1	○	○	○	✗
\textsection	§	OMS/TS1	○	✗	✗	✗
\textserviceemark	℠	TS1	○	○	○	✗
\textsevenoldstyle	7	TS1	○	○	○	✗
\textsixoldstyle	6	TS1	○	○	○	✗
\textsterling	£	OT1/TS1	○	✗	✗	✗
\textsurd	√	TS1	○	○	○	✗
\textthreeoldstyle	3	TS1	○	○	○	✗
\textthreequarters	¾	TS1	○	○	✗	✗
\textthreequartersemdash	—	TS1	○	○	○	✗
\textthreesuperior	³	TS1	○	○	○	✗
\texttildelow	˜	TS1	○	○	˜	✗
\texttimes	×	TS1	○	○	○	✗
\texttrademark	™	constr./TS1	○	○	✗	✗
\texttwelveudash	—	TS1	○	○	○	✗
\texttwooldstyle	₂	TS1	○	○	○	✗
\texttwosuperior	₂	TS1	○	○	○	✗
\textunderscore	—	constr.	○	✗	✗	
\textuparrow	↑	TS1	○	○	○	✗
\textvisiblespace	~	constr.	○	✗	○	
\textwon	₩	TS1	○	○	○	✗
\textyen	¥	TS1	○	○	✗	✗
\textzerooldstyle	₀	TS1	○	○	○	✗
\th	þ			✗	✗	
\u	(accent)	‐	OT1	✗	✗	✗
\u A	Ā			constr.	✗	constr.
\u G	Ğ			constr.	✗	constr.
\u a	ă			constr.	✗	constr.
\u g	ğ			constr.	✗	constr.
\v	(accent)	ˇ	OT1	✗	✗	✗
\v C	Č			constr.	✗	constr.
\v D	Ď			constr.	✗	constr.
\v E	Ě			constr.	✗	constr.
\v L	Ľ			constr.	✗	constr.
\v N	Ň			constr.	✗	constr.
\v R	Ř			constr.	✗	constr.
\v S	Š			constr.	✗	✗
\v T	Ť			constr.	✗	constr.
\v Z	Ž			constr.	✗	✗

✗ defined in encoding

○ defined via default

LICR Object	Glyph	Default from	OT1	T1	LY1	TS1
\v c	č		constr.	✗	constr.	
\v d	đ		constr.	✗	constr.	
\v e	ě		constr.	✗	constr.	
\v l	ł		constr.	✗	constr.	
\v n	ń		constr.	✗	constr.	
\v r	ř		constr.	✗	constr.	
\v s	š		constr.	✗	✗	
\v t	ť		constr.	✗	constr.	
\v z	ž		constr.	✗	✗	
\{ (alias)	{	\textbraceleft	○	✗	✗	
\} (alias)	}	\textbraceright	○	✗	✗	
\~ (accent)	~	OT1	✗	✗	✗	
\~A	Ā		constr.	✗	✗	
\~N	Ñ		constr.	✗	✗	
\~O	Ӯ		constr.	✗	✗	
\~a	ā		constr.	✗	✗	
\~n	ñ		constr.	✗	✗	
\~o	õ		constr.	✗	✗	

✗ defined in encoding      ○ defined via default

## 7.12 Compatibility packages for very old documents

The font interface in  $\text{\LaTeX}$  changed from a fixed font structure ( $\text{\LaTeX}$  2.09 prior to 1990) to a flexible system ( $\text{\LaTeX}_2\epsilon$  with NFSS version 2 integrated in 1994). During the years 1990–1993 NFSS version 1 was widely used in Europe. Although the differences between versions 1 and 2 have not been that enormous, they nevertheless make it impossible to run documents from that time successfully through today's  $\text{\LaTeX}$ . For this reason a number of compatibility packages have been developed to help in processing documents written for  $\text{\LaTeX}$  2.09 with or without NFSS 1.

### 7.12.1 `oldlfont`, `rawfonts`, `newlfont`—Processing old documents

As we have seen, NFSS—and thus  $\text{\LaTeX}_2\epsilon$ —differs from  $\text{\LaTeX}$  2.09 in several ways in its treatment of font commands. This difference is most noticeable in math formulas, where commands like `\bfseries` are not supported. Nevertheless, it is a very simple matter to typeset older documents with NFSS.

*Backward  
compatibility to  
1993 and earlier*

If you merely want to reprint a document,  $\text{\LaTeX}$  will see the `\documentstyle` command and automatically switch to compatibility mode, thereby emulating the old font selection mechanism of  $\text{\LaTeX}$  2.09 as described in the first edition

of the *L<sup>A</sup>T<sub>E</sub>X Manual*. Alternatively, you can load the `oldlfont` package after the `\documentclass` command. If you do so, all old font-selecting commands will be defined, font-changing commands cancel each other, and all of these commands can be used in mathematical formulas.

Some old documents refer to L<sup>A</sup>T<sub>E</sub>X 2.09 internal font commands such as `\twlrm` or `\nintt`. These commands now generate error messages, because they are no longer defined (not even in compatibility mode). One reason they are not supported is that they were never available on all installations. To process a document containing such explicit font-changing commands, you have to define them in the preamble using the commands described in Section 7.9. For example, for the above commands, it would be sufficient to add the following definitions to the preamble:

```
\newcommand\twlrm{\fontsize{12pt}{14pt}\normalfont\rmfamily}
\newcommand\nintt{\fontsize{9pt}{11pt}\normalfont\ttfamily}
```

A package exists to assist you in this task: if you load the `rawfonts` package with the options `only`, `twlrm`, and `nintt`, it will make the above declarations for you. If you load it without any option, it will define all L<sup>A</sup>T<sub>E</sub>X 2.09 hard-wired font commands for you.

Reusing parts of documents also is very simple: just paste them into the new document and watch what happens. There is a good chance that L<sup>A</sup>T<sub>E</sub>X will happily process the old document fragment and, if not, it will explicitly inform you about the places where you have to change your source—for example, where you have to change occurrences of `\it`, `\sf`, and similar commands in formulas to the corresponding math alphabet identifier commands `\mathit`, `\mathsf`, and so on.

*Backward compatibility with the first release of NFSS*

In the first release of NFSS, the two-letter font-changing commands were redefined to modify individual attributes only. For example, `\sf` and `\it` behaved just like the NFSS2 commands `\sffamily` and `\itshape`, respectively. If you reprocess an old document that was written for this convention, load the package `newlfont` in your document preamble to reinitiate it.

### 7.12.2 `latexsym`—Providing symbols from L<sup>A</sup>T<sub>E</sub>X 2.09 lasy fonts

Eleven math symbols provided by L<sup>A</sup>T<sub>E</sub>X 2.09 are no longer defined in the base set-up of NFSS:

$\square$ $\diamond$ $\bowtie$ $\triangleright$ $\triangleleft$ $\square$ $\square$	$\rightsquigarrow$ $\triangleleft$ $\square$ $\square$ $\triangleleft$ $\square$ $\square$	$\leadsto$ $\Join$ $\leq$ $\leq$ $\leq$ $\leq$ $\leq$	$\quad$ $\quad$ $\quad$ $\quad$ $\quad$ $\quad$ $\quad$
---	--	---	---

```
\usepackage{latexsym} \newcommand\Q[1]{\$#1\$ \quad}
\Q{\Box} \Q{\Diamond} \Q{\Join} \Q{\leadsto} \Q{\lhd} \Q{\rhd}
\Q{\sqsubset} \Q{\sqsupset} \Q{\unlhd} \Q{\unrhd}
```

7-12-1

If you want to use any of these symbols, load the `latexsym` package in your document. These symbols are also made available if you load the `amsfonts` or the `amssymb` package; see Section 8.9.

## CHAPTER 8

# Higher Mathematics

Basic L<sup>A</sup>T<sub>E</sub>X offers excellent mathematical typesetting capabilities for straightforward documents. However, when complex displayed equations or more advanced mathematical constructs are heavily used, something more is needed. Although it is possible to define new commands or environments to ease the burden of typing in formulas, this is not the best solution. The American Mathematical Society (AMS) provides a major package, *amsmath*, which makes the preparation of mathematical documents much less time-consuming and more consistent.<sup>1</sup> It forms the core of a collection of packages known as *AMS-L<sup>A</sup>T<sub>E</sub>X* [8] and is the major subject of this chapter. A useful book by George Grätzer [60] also covers these packages in detail.

This chapter describes briefly, and provides examples of, a substantial number of the many features of these packages as well as a few closely related packages; it also gives a few pointers to other relevant packages. In addition, it provides some essential background on mathematical typesetting with T<sub>E</sub>X. Thus, it covers some of standard L<sup>A</sup>T<sub>E</sub>X's features for mathematical typesetting and layout and contains some general hints on how to typeset mathematical formulas, though these are not the main aims of this chapter.

It is also definitely not a comprehensive manual of good practice for typesetting mathematics with L<sup>A</sup>T<sub>E</sub>X. Indeed, many of the examples are offered up purely for illustration purposes and, therefore, present neither good design, nor good mathematics, nor necessarily good L<sup>A</sup>T<sub>E</sub>X coding.

Advice on how to typeset mathematics according to late 20th-century U.S. practice can be found in Ellen Swanson's *Math into Type* [156]. Many details concerning how to implement this advice using T<sub>E</sub>X or, equally, standard L<sup>A</sup>T<sub>E</sub>X appear in Chapters 16–18 of Donald Knuth's *The T<sub>E</sub>Xbook* [82].

<sup>1</sup>This package has its foundations in the macro-level extensions to T<sub>E</sub>X known as *AMS-T<sub>E</sub>X*.

To use the majority of the material described in this chapter, you need to load at least the `amsmath` package in the preamble of your document. If other packages are needed, they are clearly marked in the examples. Detailed installation and usage documentation is included with the individual packages.

## 8.1 Introduction to $\mathcal{AM}\mathcal{S}$ - $\text{\LaTeX}$

The  $\mathcal{AM}\mathcal{S}$ - $\text{\LaTeX}$  project commenced in 1987 and three years later  $\mathcal{AM}\mathcal{S}$ - $\text{\LaTeX}$  version 1.0 was released. This was the original conversion to  $\text{\LaTeX}$  of the mathematical capabilities in Michael Spivak's  $\mathcal{AM}\mathcal{T}\mathcal{E}\mathcal{X}$  by Frank Mittelbach and Rainer Schöpf, working as consultants to the American Mathematical Society, with assistance from Michael Downes of the AMS technical staff. In 1994, further work was done with David Jones. This work was coordinated by Michael Downes and the packages have throughout been supported and much enhanced under his direction and the patronage of the AMS.<sup>1</sup>

*Thanks to a great  
guy* Michael would have been the author of this chapter had he not died in spring 2003. Much of the chapter is based on the documentation he prepared for  $\mathcal{AM}\mathcal{S}$ - $\text{\LaTeX}$ ; thus, what you are reading is a particular and heartfelt tribute by its current authors to the life and work of our dearest friend and colleague with whom we shared many coding adventures in the uncharted backwaters of  $\text{\TeX}$ .

*Available package  
options* A few options are recognized by the `amsmath` package. Most of these affect only detailed positioning of the “limits” on various types of mathematical operators (Section 8.4.4) or that of equation tags (Section 8.2.4).

The following three options are often supplied as global document options, set on the `\documentclass` command. They are, however, also recognized when the `amsmath` package is loaded with the `\usepackage` command.

- `reqno` (**default**) Place equation numbers (tags) on the right.
- `leqno` Place equation numbers (tags) on the left.<sup>2</sup>
- `fleqn` Position equations at a fixed indent from the left margin rather than centered in the text column.

*Available  
sub-packages* The  $\mathcal{AM}\mathcal{S}$ - $\text{\LaTeX}$  distribution also contains components that can be loaded independently by the `\usepackage` command. In particular, some features of the `amsmath` package are also available in these smaller packages:

- `amsopn` Provides `\DeclareMathOperator` for defining new operator names such as `\Ker` and `\esssup`.

<sup>1</sup>Some material in this chapter is reprinted from the documentation distributed with  $\mathcal{AM}\mathcal{S}$ - $\text{\LaTeX}$  (with permission from the American Mathematical Society).

<sup>2</sup>When using the  $\mathcal{AM}\mathcal{S}$ - $\text{\LaTeX}$  document classes, the default is `leqno`.

- amstext** Provides the `\text` command for typesetting a fragment of text in the correct type size.

The following packages, providing functionality additional to that in **amsmath**, *Extension packages* must be loaded explicitly; they are listed here for completeness.

- amscd** Defines some commands for easing the generation of commutative diagrams by introducing the `CD` environment (see Section 8.3.4 on page 488). There is no support for diagonal arrows.
- amsthm** Provides a method to declare theorem-like structures and offers a `proof` environment. It is discussed in Section 3.3.3 on page 138.
- amsxtra** Provides certain odds and ends that are needed for historical compatibility, such as `\fracwithdelims`, `\accentedsymbol`, and commands for placing accents as superscripts.
- upref** Makes `\ref` print cross-reference numbers in an upright/Roman font regardless of context.

The principal documentation for these packages is the *User's Guide for the amsmath Package (Version 2.0)* [8].

The current  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{\LaTeX}$  collection includes three document classes: **amsart**, **amsproc**, and **amsbook**, corresponding to  $\text{\LaTeX}$ 's `article`, `proc`, and `book`, respectively. They are designed to be used in the preparation of manuscripts for submission to the AMS [6], but nothing prohibits their use for other purposes. With these class files the **amsmath** package is automatically loaded, so that you can start your document simply with `\documentclass{amsart}`. These classes are not covered in this book as they provide an interface similar to that provided by the  $\text{\LaTeX}$  standard classes; refer to [6] for details of their use.

*The  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{\LaTeX}$  document classes*

Some of the material in this chapter refers to another collection of packages from the American Mathematical Society, namely the **AMSfonts** distribution. These packages, listed below, set up various fonts and commands for use in mathematical formulas.

*The AMSfonts collection*

- amsfonts** Defines the `\mathfrak` and `\mathbb` commands and sets up the fonts `msam` (extra math symbols A), `msbm` (extra math symbols B and blackboard bold), `eufm` (Euler Fraktur), extra sizes of `cmmib` (bold math italic and bold lowercase Greek), and `cmbsy` (bold math symbols and bold script).
- amssymb** Defines the names of the mathematical symbols available with the **AMSfonts** collection. These commands are discussed in Section 8.9. The package automatically loads the **amsfonts** package.
- eufrak** Sets up the fonts for the Euler Fraktur letters (`\mathfrak`), as discussed in Section 7.7.10. This alphabet is also available from the **amsfonts** package.
- eucl** Makes `\mathcal` use the Euler script instead of the usual Computer Modern script letters, see Section 7.7.10 for details.

All of these packages recognize the `psamsfonts` option, which will set up L<sup>A</sup>T<sub>E</sub>X to use the Y&Y/Blue Sky Research version of these fonts in the AMSfonts collection. This will be useful only if you have this version of the fonts installed on your system; they are available on CTAN and are often available as the default in modern distributions of L<sup>A</sup>T<sub>E</sub>X. The principal piece of documentation for these packages is the *User's Guide to AMSFonts Version 2.2d* [9].

### A few important warnings

*Watch out for fragile commands*

Many of the commands described in this chapter are fragile and need to be `\protected` in moving arguments (see Appendix B.1 on page 892). Thus, when strange error messages appear, a missing `\protect` is a likely cause.

*Do not abbreviate environments*

It is never a good idea to use shortcut codes for L<sup>A</sup>T<sub>E</sub>X environments. With the `amsmath` display environments described in this chapter, such shortcuts are always disastrous—don't do it! For closely related reasons, you will also find that verbatim material cannot be used within these environments. Here are some examples of declarations for disaster:

```
\newenvironment{mlt}{\begin{multiline}}{\end{multiline}}
\newcommand\bga{\begin{gather}} \newcommand\ega{\end{gather}}
```

Both will produce errors of the form “`\begin{...}` ended by ...”. However, you can define synonyms and variant forms of these environments as follows:

```
\newenvironment{mlt}{\multline}{\endmultline}
\newenvironment{longgather}{\allowdisplaybreaks\gather}{\endgather}
```

Note that these must have the command form of an existing environment as the last command in the “begin-code”, and the corresponding `\end...` command as the first thing in the “end-code”. See also Section A.1.3, for more details.

## 8.2 Display and alignment structures for equations

The `amsmath` package defines several environments for creating displayed mathematics. These cover single- and multiple-line displays with single or multiple alignment points and various options for numbering equations within displays.

Throughout this section the term “equation” will be used in a very particular way: to refer to a *logical* distinct part of a mathematical display that is frequently numbered for reference purposes and is also labeled (commonly by its number in parentheses). Such labels are often called *tags*.

The complete list of all the display environments you will need for mathematical typesetting is given in Table 8.1 on the next page; the majority of these environments are covered in this section, along with examples of their use. Where

<code>equation</code>	<code>equation*</code>	One line, one equation
<code>multline</code>	<code>multline*</code>	One unaligned multiple-line equation, one equation number
<code>gather</code>	<code>gather*</code>	Several equations without alignment
<code>align</code>	<code>align*</code>	Several equations with multiple alignments
<code>flalign</code>	<code>flalign*</code>	Several equations: horizontally spread form of align
<code>split</code>		A simple alignment within a multiple-line equation
<code>gathered</code>		A “mini-page” with unaligned equations
<code>aligned</code>		A “mini-page” with multiple alignments

Table 8.1: Display environments in the `amsmath` package

appropriate they have starred forms in which there is no numbering or tagging of the equations.

In these examples of alignment environments, other commands from the `amsmath` package are also used. A detailed understanding of how these work is not necessary at this stage; an interested reader can turn to later sections for more information. The display width is the measure that defines the right and left margins (or extents) of a display; in the examples these extents are indicated by thin blue vertical rules at the right and left margins of the display.

Except where noted, all examples in this chapter are typeset with the mathematical material centered and the equation numbers, or tags, on the right (the default settings for the `amsmath` package). When the option `leqno` is specified for the `amsmath` package or the document class, the equation number tags will be printed at the left side of the equation.

<span style="border: 1px solid black; padding: 2px;">8-2-1</span> 1) $(a + b)^2 = a^2 + 2ab + b^2$ $\sin^2 \eta + \cos^2 \eta = 1$	<code>\usepackage[leqno]{amsmath}</code> <code>\begin{equation} (a+b)^2 = a^2+2ab+b^2 \end{equation}</code> <code>\[ \sin^2\eta+\cos^2\eta = 1 \]</code>
---	--

To position the mathematics at a fixed indent from the left margin, rather than centered in the text column, you use the option `fleqn`. You will then normally need to set the size of the indent in the preamble. It is the value of the rubber length `\mathindent`, which gets its default value from the indentation of a first-level list—which is probably not the value you want! Observe the differences between the next example and the previous example. In this particular case, use of the `reqno` option is redundant (as it is the default), but it forces the equation number to the right side regardless of what the document class specifies.

<span style="border: 1px solid black; padding: 2px;">8-2-2</span> (1) $(a + b)^2 = a^2 + 2ab + b^2$ $\sin^2 \eta + \cos^2 \eta = 1$	<code>\usepackage[fleqn,reqno]{amsmath}</code> <code>\setlength\mathindent{1pc}</code> <code>\begin{equation} (a+b)^2 = a^2+2ab+b^2 \end{equation}</code> <code>\[ \sin^2\eta+\cos^2\eta = 1 \]</code>
--	---

As later examples will show, as in standard L<sup>A</sup>T<sub>E</sub>X, & and \\ are used for column and line separation within displayed alignments. The details of their usage change in the amsmath environments, however (see the next section).

### 8.2.1 Comparison with standard L<sup>A</sup>T<sub>E</sub>X

Some of the multiple-line display environments allow you to align parts of the formula. In contrast to the original L<sup>A</sup>T<sub>E</sub>X environments eqnarray and eqnarray\*, the structures implemented by the amsmath package use a slightly different and more straightforward method for marking the alignment points. Standard L<sup>A</sup>T<sub>E</sub>X's eqnarray\* is similar to an array environment with {rc1} as the preamble and, therefore, requires two ampersand characters indicating the two alignment points. In the equivalent amsmath structures there is only a single alignment point (similar to a {r1} preamble), so only a single ampersand character should be used, placed to the left of the symbol (usually a relation) that should be aligned.

The amsmath structures give fixed spacing at the alignment points, whereas the eqnarray environment produces extra space depending on the parameter settings for array. The difference can be seen clearly in the next example, where the same equation is typeset using the equation, align, and eqnarray environments; the spaces in the eqnarray environment come out too wide for conventional standards of mathematical typesetting.

$x^2 + y^2 = z^2$ $x^2 + y^2 = z^2$ $x^3 + y^3 < z^3$ $x^2 + y^2 = z^2$ $x^3 + y^3 < z^3$	<pre>\usepackage{amsmath} (1) \begin{equation}       x^2 + y^2 = z^2     \end{equation} (2) \begin{align}       x^2 + y^2 &amp;= z^2 \\       x^3 + y^3 &amp;&lt; z^3     \end{align} (3) \begin{eqnarray}       x^2 + y^2 &amp;=&amp; z^2 \\       x^3 + y^3 &amp;&amp; &lt; z^3     \end{eqnarray}</pre>
---	--

8-2-3

As in standard L<sup>A</sup>T<sub>E</sub>X, lines in an amsmath display are marked with \\ (or the end of the environment). Because line breaking in a mathematical display usually requires a thorough understanding of the structure of the formula, it is commonly considered to be beyond today's software capabilities. However, one of the last bigger projects undertaken by Michael Downes precisely tackled this problem; it resulted in the breqn package (see [42] for details).

Unlike eqnarray, the amsmath environments do not, by default, allow page breaks between lines (see Section 8.2.10).

*Space after \\ not ignored* Another difference concerns the use of \\[dimension] or \\\* within mathematical display environments. With amsmath, there must be no space between

the `\\"` and the `[` or the `*`; otherwise, the optional argument or star will not be recognized. The reason is that brackets and stars are very common in mathematical formulas, so this restriction avoids the annoyance of having a genuine bracket belonging to the formula be mistaken for the start of the optional argument.

Finally, there is one less obvious change that is very unlikely to cause any problems for users: in standard L<sup>A</sup>T<sub>E</sub>X the parameter `\mathindent` is a non-rubber length, whereas in `amsmath` it becomes a rubber length. The reasons for, and consequences of, this change are discussed in `amsmath.dtx`, the documented source of the `amsmath` package.

### 8.2.2 A single equation on one line

The `equation` environment produces a single equation with an automatically generated number or tag placed on the extreme left or right according to the option in use (see Section 8.2.11); `equation*` does the same but omits a tag.<sup>1</sup>

Note that the presence of the tag does not affect the positioning of the contents. If there is not enough room for it on the one line, the tag will be shifted up or down: to the previous line when equation numbers are on the left, and to the next line when numbers are on the right.

```
\usepackage[leqno]{amsmath}
\begin{equation*}
n^2 + m^2 = k^2
\end{equation*}
\begin{equation}
n^p + m^p \neq k^p \quad p > 2
\end{equation}
```

8-2-4

### 8.2.3 A single equation on several lines: no alignment

The `multline` environment is a variation of the `equation` environment used only for equations that do not fit on a single line. In this environment `\\"` must be used to mark the line breaks, as they are not found automatically.

The first line of a `multline` will be aligned on an indentation from the left margin and the last line on the same indentation from the right margin.<sup>2</sup> The size of this indentation is the value of the length `\multlinegap`; thus, it can be changed using L<sup>A</sup>T<sub>E</sub>X's `\setlength` and `\addtolength` commands.

If a `multline` contains more than two lines, each line other than the first and last is centered individually within the display width (unless the option `fleqn` is used). It is, however, possible to force a single line to the left or the right by adding either `\shoveleft` or `\shoveright` within that line.

<sup>1</sup>Standard L<sup>A</sup>T<sub>E</sub>X also has `equation`, but not `equation*`, as the latter is similar to the standard displayed math environment.

<sup>2</sup>Never use `multline` for a single-line equation because the effect is unpredictable.

A `multiline` environment is a single (logical) equation and thus has only a single tag, the `multiline*` having none; thus, none of the individual lines can be changed by the use of `\tag` or `\notag`. The tag, if present, is placed flush right on the last line with the default `reqno` option or flush left on the first line when the `leqno` option is used.

First line of a multiline

Centered Middle line

A right Middle

Another centered Middle

Yet another centered Middle

A left Middle

Last line of the multiline (1)

```
\usepackage{amsmath}
\begin{multiline}
\text{First line of a multiline} \\
\text{Centered Middle line} \\
\shoveright{\text{A right Middle}} \\
\text{Another centered Middle} \\
\text{Yet another centered Middle} \\
\shoveleft{\text{A left Middle}} \\
\text{Last line of the multiline}
\end{multiline}
```

8-2-5

The next example shows the effect of `\multlinegap`. In the first setting, the “`dy`”s line up and make it appear that a tag is missing from the first line of the equation. When the parameter is set to zero, the space on the left of the second line does not change because of the tag, while the first line is pushed over to the left margin, thus making it clear that this is only one equation.

$$\sum_{t \in T} \int_a^t \left\{ \int_a^t f(t-x)^2 g(y)^2 dx \right\} dy = \sum_{t \notin T} \int_t^a \left\{ g(y)^2 \int_t^a f(x)^2 dx \right\} dy \quad (2)$$

```
\usepackage{amsmath}
\begin{multiline} \tag{2}
\sum_{t \in T} \int_a^t \left\{ \int_a^t f(t-x)^2 g(y)^2 dx \right\} dy \\
= \sum_{t \notin T} \int_t^a \left\{ g(y)^2 \int_t^a f(x)^2 dx \right\} dy
\end{multiline}
```

$$\sum_{t \in T} \int_a^t \left\{ \int_a^t f(t-x)^2 g(y)^2 dx \right\} dy = \sum_{t \notin T} \int_t^a \left\{ g(y)^2 \int_t^a f(x)^2 dx \right\} dy \quad (2)$$

```
\setlength{\multlinegap}{0pt}
\begin{multiline} \tag{2}
\sum_{t \in T} \int_a^t \left\{ \int_a^t f(t-x)^2 g(y)^2 dx \right\} dy \\
= \sum_{t \notin T} \int_t^a \left\{ g(y)^2 \int_t^a f(x)^2 dx \right\} dy
\end{multiline}
```

8-2-6

### 8.2.4 A single equation on several lines: with alignment

When a simple alignment is needed within a single multiple-line equation, the `split` environment is almost always the best choice. It uses a single ampersand (`&`) on each line to mark the alignment point.

$$\begin{aligned}
 (a+b)^4 &= (a+b)^2(a+b)^2 \\
 [8-2-7] \quad &= (a^2 + 2ab + b^2)(a^2 + 2ab + b^2) \quad (1) \\
 &= a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4
 \end{aligned}$$

```
\usepackage{amsmath}
\begin{equation}
\begin{split}
(a+b)^4 &\\
&\&= (a+b)^2 \quad (a+b)^2 \\ 
&\&= (a^2 + 2ab + b^2) \\
&\& \quad (a^2 + 2ab + b^2) \\ 
&\& \&= a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4
\end{split}
\end{equation}
```

Because it is always used as the content of a single (logical) equation, a `split` does not itself produce any numbering tag and hence there is no starred variant. If needed, the outer display environment will provide any needed tags.

Apart from commands such as `\label` or `\notag` that produce no visible material, a `split` structure should normally constitute the entire body of the equation being split. It can consist of either a whole `equation` or `equation*` environment or one whole line of a `gather` or `gather*` environment; see Section 8.2.5.

When the `centertags` option is in effect (the default), the tag (and any other material in the equation outside the `split`) is centered vertically on the total height of the material from the `split` environment. When the `tbtags` option is specified, the tag is aligned with the last line of the `split` when the tag is on the right, and with the first line of the `split` when the tag is on the left.

$$\begin{aligned}
 (a+b)^3 &= (a+b)(a+b)^2 \\
 [8-2-8] \quad &= (a+b)(a^2 + 2ab + b^2) \\
 &= a^3 + 3a^2b + 3ab^2 + b^3 \quad (1)
 \end{aligned}$$

```
\usepackage[tbtags]{amsmath}
\begin{equation}
\begin{split}
(a+b)^3 &\&= (a+b) \quad (a+b)^2 \\ 
&\&= (a+b)(a^2 + 2ab + b^2) \\ 
&\& \&= a^3 + 3a^2b + 3ab^2 + b^3
\end{split}
\end{equation}
```

In the next example the command `\phantom` is used to adjust the horizontal positioning. It is first used in the preamble to define an “invisible relation symbol” of width equal to that of its argument (in this case, `=`). Within the example it is used to align certain lines by starting them with a “phantom, or invisible, sub-formula” (see Section 8.7.2 on page 503). The empty pair of braces `{}` is equivalent

to `\mathord{}` and provides an invisible zero-width “letter” that is needed to achieve the correct spacing of  $+ h$  (without the `{}` it would look like this:  $+h$ ).

```
\usepackage{amsmath}
\newcommand{\relphantom}[1]{\mathrel{\phantom{#1}}}
\newcommand{\ve{\varepsilon}}{\varepsilon}
\newcommand{\vf{\varphi}}{\varphi}
\newcommand{\bfE{\mathbf{E}}}{\mathbf{E}}
\begin{equation} \begin{aligned}
f_{h,\varepsilon}(x, y) &= \ve{\varepsilon} \mathbf{E}_{x,y} \int_0^{t_\varepsilon} L_{x,y_\varphi(\varepsilon u)} \varphi(x) du \\
&= h \int L_{x,z} \varphi(x) \rho_x(dz) \\
&\quad + h \left[ \frac{1}{t_\varepsilon} \left( \mathbf{E}_y \int_0^{t_\varepsilon} L_{x,y^x(s)} \varphi(x) ds - t_\varepsilon \int L_{x,z} \varphi(x) \rho_x(dz) \right) + \right. \\
&\quad \left. \frac{1}{t_\varepsilon} \left( \mathbf{E}_y \int_0^{t_\varepsilon} L_{x,y^x(s)} \varphi(x) ds - \mathbf{E}_{x,y} \int_0^{t_\varepsilon} L_{x,y_\varphi(\varepsilon s)} \varphi(x) ds \right) \right]
\end{aligned} \end{equation}
```

Note that the equation number tag has been moved to the line below the displayed material. Although this does not seem to be a very wise decision, it is as far as the automated expertise built into the system at this stage can take us.

$$\begin{aligned}
f_{h,\varepsilon}(x, y) &= \varepsilon \mathbf{E}_{x,y} \int_0^{t_\varepsilon} L_{x,y_\varphi(\varepsilon u)} \varphi(x) du \\
&= h \int L_{x,z} \varphi(x) \rho_x(dz) \\
&\quad + h \left[ \frac{1}{t_\varepsilon} \left( \mathbf{E}_y \int_0^{t_\varepsilon} L_{x,y^x(s)} \varphi(x) ds - t_\varepsilon \int L_{x,z} \varphi(x) \rho_x(dz) \right) + \right. \\
&\quad \left. \frac{1}{t_\varepsilon} \left( \mathbf{E}_y \int_0^{t_\varepsilon} L_{x,y^x(s)} \varphi(x) ds - \mathbf{E}_{x,y} \int_0^{t_\varepsilon} L_{x,y_\varphi(\varepsilon s)} \varphi(x) ds \right) \right]
\end{aligned} \tag{1}$$

8-2-9

### 8.2.5 Equation groups without alignment

The `gather` environment is used to put two or more equations into a single display without alignment between the equations. Each equation is separately centered

within the display width and has its individual number tag, if needed. Each line of a `gather` is a single (logical) equation.

<span style="border: 1px solid black; padding: 2px;">8-2-10</span>	$(a + b)^2 = a^2 + 2ab + b^2$ $(a + b) \cdot (a - b) = a^2 - b^2$	(1) $\begin{aligned} (a + b)^2 &= a^2 + 2ab + b^2 \\ (a + b) \cdot (a - b) &= a^2 - b^2 \end{aligned}$ (2) $\begin{aligned} (a + b) \cdot (a - b) &= a^2 - b^2 \end{aligned}$	<pre>\usepackage{amsmath} \begin{gather} (a + b)^2 = a^2 + 2ab + b^2 (a + b) \cdot (a - b) = a^2 - b^2 \end{gather}</pre>
--	---	---	---

Use `\notag` within the logical line to suppress the equation number for that line; or use `gather*` to suppress all equation numbers.

<span style="border: 1px solid black; padding: 2px;">8-2-11</span>	$D(a, r) \equiv \{z \in \mathbf{C}:  z - a  < r\}$ $\text{seg}(a, r) \equiv \{z \in \mathbf{C}: \Im z < \Im a,  z - a  < r\}$ $C(E, \theta, r) \equiv \bigcup_{e \in E} c(e, \theta, r)$	(1) (2)	<pre>\usepackage{amsmath} \begin{gather} D(a, r) \equiv \{z \in \mathbf{C}:  z - a  &lt; r\} \notag \\ \text{seg}(a, r) \equiv \{z \in \mathbf{C}: \Im z &lt; \Im a,  z - a  &lt; r\} \notag \\ C(E, \theta, r) \equiv \bigcup_{e \in E} c(e, \theta, r) \end{gather}</pre>
--	--	------------	---

### 8.2.6 Equation groups with simple alignment

The `align` environment should be used for two or more equations in a single display with vertical alignment. The simplest form uses a single ampersand (`&`) on each line to mark the alignment point (usually just before a Relation symbol).

<span style="border: 1px solid black; padding: 2px;">8-2-12</span>	$(a + b)^3 = (a + b)(a + b)^2$ $= (a + b)(a^2 + 2ab + b^2)$ $= a^3 + 3a^2b + 3ab^2 + b^3$	(1) (2) (3)	<pre>\usepackage{amsmath} \begin{align} (a + b)^3 &amp;= (a + b)(a + b)^2 \\ &amp;= (a + b)(a^2 + 2ab + b^2) \\ &amp;= a^3 + 3a^2b + 3ab^2 + b^3 \end{align}</pre>
	$x^2 + y^2 = 1$ $x = \sqrt{1 - y^2}$	(4) (5)	<pre>\begin{aligned} x^2 + y^2 &amp;= 1 \\ x &amp;= \sqrt{1 - y^2} \end{aligned}</pre>

### 8.2.7 Multiple alignments: align and flalign

An `align` environment can include more than one alignment point. The layout contains as many column-pairs as necessary and is similar to an `array` with preamble of the form `{rlrl...}`. If it consists of  $n$  such `rl` column-pairs, then the number

of ampersands per line will be  $2n - 1$ : one ampersand for alignment within each column-pair giving  $n$ ; and  $n - 1$  ampersands to separate the column-pairs.

Within the `align` environment, the material is spread out evenly across the display width. All extra (or white) space within the line is distributed equally “between consecutive `r1` column-pairs” and the two display margins.

This example has two column-pairs.

$$\begin{array}{ll} \text{Compare } x^2 + y^2 = 1 & x^3 + y^3 = 1 \\ x = \sqrt{1 - y^2} & x = \sqrt[3]{1 - y^3} \end{array} \quad (1) \quad (2)$$

This example has three column-pairs.

$$\begin{array}{lll} x = y & X = Y & a = b + c \\ x' = y' & X' = Y' & a' = b \\ x + x' = y + y' & X + X' = Y + Y' & a'b = c'b \end{array} \quad (3) \quad (4) \quad (5)$$

```
\usepackage{amsmath}
This example has two column-pairs.
\begin{align} \text{\texttt{Compare }} \\ x^2 + y^2 &= 1 & & \\ x^3 + y^3 &= 1 & & \\ x &= \sqrt{1-y^2} & & \\ x &= \sqrt[3]{1-y^3} & & \end{align}
```

```
This example has three column-pairs.
\begin{align} x &= y & & X &= Y & \\ a &= b+c & & & & \\ x' &= y' & & X' &= Y' & \\ a' &= b & & & & \\ x + x' &= y + y' & & & & \\ x + X' &= Y + Y' & & a'b &= c'b & \end{align}
```

8-2-13

This example has two column-pairs.

$$\begin{array}{ll} \text{Compare } x^2 + y^2 = 1 & x^3 + y^3 = 1 \\ x = \sqrt{1 - y^2} & x = \sqrt[3]{1 - y^3} \end{array} \quad (1) \quad (2)$$

This example has three column-pairs.

$$\begin{array}{lll} x = y & X = Y & a = b + c \\ x' = y' & X' = Y' & a' = b \\ x + x' = y + y' & X + X' = Y + Y' & a'b = c'b \end{array} \quad (3) \quad (4) \quad (5)$$

```
\usepackage{amsmath}
This example has two column-pairs.
\begin{flalign} \text{\texttt{Compare }} \\ x^2 + y^2 &= 1 & & \\ x^3 + y^3 &= 1 & & \\ x &= \sqrt{1-y^2} & & \\ x &= \sqrt[3]{1-y^3} & & \end{flalign}
```

```
This example has three column-pairs.
\begin{flalign} x &= y & & X &= Y & \\ a &= b+c & & & & \\ x' &= y' & & X' &= Y' & \\ a' &= b & & & & \\ x + x' &= y + y' & & & & \\ x + X' &= Y + Y' & & a'b &= c'b & \end{flalign}
```

8-2-14

In both cases the minimum space between column-pairs can be set by changing `\minalignsep`. Its default value is 10pt but, misleadingly, it is *not* a length

parameter. Thus, it must be changed by using `\renewcommand`. If we set it to zero for the first part of the example, Equation (2) gets squeezed onto a single line; if we set it to 15pt later, the label (3) gets forced onto a line by itself.

Unfortunately, there is no such simple parametric method for controlling the spacing at the margins.

This example has two column-pairs.

$$\begin{aligned} \text{Compare } x^2 + y^2 &= 1 & x^3 + y^3 &= 1 & (1) \\ x = \sqrt{1 - y^2} & & x = \sqrt[3]{1 - y^3} & & (2) \end{aligned}$$

This example has three column-pairs.

$$\begin{aligned} \boxed{8-2-15} \quad x = y & & X = Y & & a = b + c & \\ & & & & & (3) \\ x' = y' & & X' = Y' & & a' = b & (4) \\ x + x' = y + y' & & X + X' = Y + Y' & & a'b = c'b & (5) \end{aligned}$$

```
\usepackage{amsmath}
This example has two column-pairs.
\renewcommand{\minalignsep}{0pt}
\begin{align} \text{Compare } & \\ x^2 + y^2 & \&= 1 & & & \& \\ x^3 + y^3 & \&= 1 & & & \\ x & \&= \sqrt{1-y^2} & \& & \\ x & \&= \sqrt[3]{1-y^3} & & & \end{align}
\end{align}
This example has three column-pairs.
\renewcommand{\minalignsep}{15pt}
\begin{flalign} x & \&= y & \& X & \&= Y & \& \\ a & \&= b+c & & & & & \\ x' & \&= y' & \& X' & \&= Y' & \& \\ a' & \&= b & & & & & \\ x + x' & \&= y + y' & & & & & \\ X + X' & \&= Y + Y' & \& a'b & \&= c'b & \& \end{flalign}
```

The next example illustrates a very common use for `align`. Note the use of `\text` to produce normal text within the mathematical material.

$$\begin{aligned} \boxed{8-2-16} \quad x = y & & \text{by hypothesis} & & (1) \\ x' = y' & & \text{by definition} & & (2) \\ x + x' = y + y' & & \text{by Axiom 1} & & (3) \end{aligned}$$

```
\usepackage{amsmath}
\renewcommand{\minalignsep}{2em}
\begin{align} x & \&= y & \& \text{by hypothesis} \\ x' & \&= y' & \& \text{by definition} \\ x + x' & \&= y + y' & \& \text{by Axiom 1} \end{align}
```

### 8.2.8 Display environments as mini-pages

All the environments described so far produce material set to the full display width. A few of these environments have also been adapted to provide self-contained alignment structures, as if they were set as the only content of a `minipage` environment whose size, in both directions, is determined by its contents. The environment names are changed only slightly: to `aligned` and `gathered`. Note that an `aligned` environment avoids unnecessary space on the left and right; thus, it mostly resembles the `flalign` environment.

Like `minipage`, these environments take an optional argument that specifies the vertical positioning with respect to the material on either side. The default alignment of the box is centered ([c]). Of course, like `split` they are used only within equations and they never produce tags.

$$\begin{aligned} x^2 + y^2 &= 1 \\ x &= \sqrt{1 - y^2} \\ \text{and also } y &= \sqrt{1 - x^2} \end{aligned} \quad (1)$$

```
\usepackage{amsmath}
\begin{equation}
\begin{aligned}
x^2 + y^2 &= 1 \\
x &= \sqrt{1-y^2} \\
\text{and also } y &= \sqrt{1-x^2}
\end{aligned}
\end{equation}
```

8-2-17

The same mathematics can also be typeset, albeit not very beautifully, using different vertical alignments for the environments.

$$\begin{aligned} x^2 + y^2 &= 1 \\ x &= \sqrt{1 - y^2} \\ \text{and also } y &= \sqrt{1 - x^2} \end{aligned} \quad (1)$$

```
\usepackage{amsmath}
\begin{equation}
\begin{aligned}[t]
x^2 + y^2 &= 1 \\
x &= \sqrt{1-y^2} \\
\text{and also } y &= \sqrt{1-x^2}
\end{aligned}
\end{equation}
```

8-2-18

They may be used in many ways—for example, to do some creative and useful grouping of famous equations. Incidentally, these mini-page display environments are among the very few from `amsmath` that are robust enough to be used inside other definitions, as in the following example.

$$\left. \begin{aligned} \mathbf{B}' &= -c\nabla \times \mathbf{E} \\ \mathbf{E}' &= c\nabla \times \mathbf{B} - 4\pi\mathbf{J} \end{aligned} \right\} \quad \text{Maxwell's equations}$$

```
\usepackage{amsmath,bm}
\newenvironment{rcase}
{\left.\begin{aligned}}
{\end{aligned}\right\rbrace}
\begin{equation*}
\begin{rcase}
\bm{B}' &= -c\nabla \times \bm{E} \\
\bm{E}' &= c\nabla \times \bm{B} - 4\pi\bm{J},
\end{rcase}
\quad \text{Maxwell's equations}
\end{equation*}
```

8-2-19

You can also use the `\minalignsep` command to control the space between pairs of columns in an aligned environment, as shown in the next example.

**8-2-20**

$$\begin{aligned} V_j &= v_j & X_i &= x_i - q_i x_j & = u_j + \sum_{i \neq j} q_i & (1) \\ V_i &= v_i - q_i v_j & X_j &= x_j & U_i &= u_i \end{aligned}$$

```
\usepackage{amsmath}
\renewcommand{\minalignsep}{5pt}
\begin{equation} \begin{aligned} V_j &= v_j & X_i &= x_i - q_i x_j & = u_j + \sum_{i \neq j} q_i & (1) \\ V_i &= v_i - q_i v_j & X_j &= x_j & U_i &= u_i \end{aligned} \end{equation}
```

### 8.2.9 Interrupting displays: `\intertext`

The `\intertext` command is used for a short passage of text (typically at most a few lines) that appears between the lines of a display alignment. Its importance stems from the fact that all the alignment properties are unaffected by the text, which itself is typeset as a normal paragraph set to the display width; this alignment would not be possible if you simply ended the display and then started a new display after the text. This command may appear only immediately after a `\backslash` or `\backslash*` command.

Here the words “and finally” are outside the alignment, at the left margin, but all three equations are aligned.

**8-2-21**

and finally

$$\begin{aligned} A_1 &= N_0(\lambda; \Omega') - \phi(\lambda; \Omega') & (1) & \begin{aligned} \begin{aligned} A_1 &= N_0(\lambda; \Omega') - \phi(\lambda; \Omega') \\ A_2 &= \phi(\lambda; \Omega')\phi(\lambda; \Omega) \end{aligned} & \begin{aligned} \begin{aligned} A_1 &= N_0(\lambda; \Omega') - \phi(\lambda; \Omega') \\ A_2 &= \phi(\lambda; \Omega')\phi(\lambda; \Omega) \end{aligned} \end{aligned} \\ A_2 &= \phi(\lambda; \Omega')\phi(\lambda; \Omega) & (2) & \begin{aligned} A_2 &= \phi(\lambda; \Omega')\phi(\lambda; \Omega) \\ A_3 &= \mathcal{N}(\lambda; \omega) \end{aligned} \\ A_3 &= \mathcal{N}(\lambda; \omega) & (3) & \begin{aligned} A_3 &= \mathcal{N}(\lambda; \omega) \end{aligned} \end{aligned}$$

```
\usepackage{amsmath}
\begin{aligned} A_1 &= N_0(\lambda; \Omega') - \phi(\lambda; \Omega') & (1) & \begin{aligned} A_1 &= N_0(\lambda; \Omega') - \phi(\lambda; \Omega') \\ A_2 &= \phi(\lambda; \Omega')\phi(\lambda; \Omega) \end{aligned} \\ A_2 &= \phi(\lambda; \Omega')\phi(\lambda; \Omega) & (2) & \begin{aligned} A_2 &= \phi(\lambda; \Omega')\phi(\lambda; \Omega) \\ A_3 &= \mathcal{N}(\lambda; \omega) \end{aligned} \\ A_3 &= \mathcal{N}(\lambda; \omega) & (3) & \begin{aligned} A_3 &= \mathcal{N}(\lambda; \omega) \end{aligned} \end{aligned}
```

### 8.2.10 Vertical space and page breaks in and around displays

As is usual in L<sup>A</sup>T<sub>E</sub>X, the optional argument `\[dimension]` gives extra vertical space between two lines in all amsmath display environments (there must be no space between the `\backslash` and the `[` character delimiting the optional argument). The vertical spaces before and after each display environment are controlled by the following rubber lengths, where the values in parentheses are those for `\normalsize` with the (default) 10pt option in the standard L<sup>A</sup>T<sub>E</sub>X classes:<sup>1</sup>

`\abovedisplayskip`, `\belowdisplayskip`      The normal vertical space added above and below a mathematical display (default 10pt plus 2pt minus 5pt).

Space within the display...

... and around the display

<sup>1</sup>These defaults are very much improved by the *AMSLATEX* document classes.

`\abovedisplayshortskip, \belowdisplayshortskip` The (usually smaller) vertical space added above and below a “short display” (0pt plus 3pt and 6pt plus 3pt minus 3pt, respectively). A *short display* is one that starts to the right of where the preceding text line ends.

If you look closely, you can observe the results of these space parameters in the following example. The second equation is surrounded by less space because the text in front of it does not overlap with the formula.

We now have the following:

$$X = a \quad a = c$$

and thus we have

$$X = c \tag{1}$$

And now we don't get much space around the display!

`\usepackage{amsmath}`

We now have the following:

`\[ X = a \quad a = c \]`

and thus we have

`\begin{equation} X = c \end{equation}`

And now we don't get much space around the display!

8-2-22

Since the four parameters `\abovedisplay...` and `\belowdisplay...` depend on the current font size, they cannot be modified in the preamble of the document using `\setlength`. Instead, they must be changed by modifying `\normalsize`, `\small`, and similar commands—a job usually done in a document class.

*Page breaks around  
the display...*

Automatic page breaking before and after each display environment is controlled by the penalty parameters `\predisplaypenalty` (for breaking before a display; default 10000, i.e., no break allowed) and `\postdisplaypenalty` (for breaking after a display, default 0; i.e., breaks allowed). The defaults are already set in standard L<sup>A</sup>T<sub>E</sub>X and are not changed by amsmath.

*and within the  
display*

Unlike standard L<sup>A</sup>T<sub>E</sub>X, the amsmath display environments do not, by default, allow page breaks between lines of the display. The reason for this behavior is that correct page breaks in such locations depend heavily on the structure of the display, so they often require individual attention from the author.

With amsmath such individual control of page breaks is best achieved via the `\displaybreak` command, but it should be used only when absolutely necessary to allow a page break within a display. The command must go before the `\` at which a break may be taken, and it applies only to that line and can be used only within an environment that produces a complete display. Somewhat like standard L<sup>A</sup>T<sub>E</sub>X's `\pagebreak` (see Section 6.2.2 in [104]), `\displaybreak` takes an optional integer as its argument, with a value ranging from zero to four, denoting the desirability of the page break: `\displaybreak[0]` means “it is permissible to break here” without encouraging a break; `\displaybreak` with no optional argument is the same as `\displaybreak[4]` and forces a break. This command cannot be used to discourage or prevent page breaks. Note that it makes no sense to break within a “mini-page display”, as those environments will never be split over two pages.

This kind of adjustment is fine-tuning, like the insertion of line breaks and page breaks in text. It should therefore be left until your document is nearly

finalized. Otherwise, you may end up redoing the fine-tuning several times to keep up with changing document content.

The command `\allowdisplaybreaks`, which obeys the usual L<sup>A</sup>T<sub>E</sub>X scoping rules, is equivalent to putting `\displaybreak` before every line end in any display environment within its scope; it takes the same optional argument as `\displaybreak`. Within the scope of an `\allowdisplaybreaks` command, the `\!*` command can be used to prohibit a page break.

The effect of a `\displaybreak` command overrides both the default and the effect of an `\allowdisplaybreaks`.

Many authors wisely use empty lines between major structures in the document source to make it more readable. In most cases, such as before and after a heading, these empty lines do no harm. This is not universally true, however. Especially around and within mathematical display environments, one has to be quite careful: a blank line in front of such an environment will produce unexpected formatting because the empty line is in effect converted into a paragraph containing no text (and so containing just the invisible paragraph indentation box). The following display is consequently surrounded by spaces of size `\displayshortskip`. Thus, the combined result is quite a lot of (possibly too much) space before the display (a whole empty line plus the `\abovedisplayshortskip`) and a very small amount of space after the display, as this example shows.

 Be wary of empty  
lines around  
displays

<code>\usepackage{amsmath}</code>	
Empty line before display:	<code>\begin{array}{l} \text{Empty line before display:} \\ \\ \text{\qquad\qquad\qquad}\backslash[ a \neq b \backslash] \end{array}</code>
$a \neq b$	
In both cases, too much space before! ...	In both cases, too much space before! \ldots
$a \neq b$	<code>\begin{array}{l} \text{\qquad\qquad\qquad}\backslash[ a \neq b \backslash] \\ \text{(1)} \end{array}</code>
... and not a lot of space after!	<code>\begin{array}{l} \text{\qquad\qquad\qquad}\backslash[ a \neq b \backslash] \\ \text{\qquad\qquad\qquad}\backslash dots\ and\ not\ a\ lot\ of\ space\ after! \end{array}</code>

8-2-23

With the `amsmath` package loaded, this behavior is exhibited by all the display math environments. Strangely enough, with standard L<sup>A</sup>T<sub>E</sub>X the `\[` case comes out looking more or less right.

<code>\usepackage{amsmath}</code>	
Empty line before display:	<code>\begin{array}{l} \text{Empty line before display:} \\ \\ \text{\qquad\qquad\qquad}\backslash[ a \neq b \backslash] \end{array}</code>
$a \neq b$	
Enough space now, but don't rely on it!	Enough space now, but don't rely on it!
$a \neq b$	<code>\begin{array}{l} \text{\qquad\qquad\qquad}\backslash[ a \neq b \backslash] \\ \text{(1)} \end{array}</code>
Less space after in this case!	Less space after in this case!

8-2-24

To summarize, do not use empty lines around display environments!

### 8.2.11 Equation numbering and tags

In L<sup>A</sup>T<sub>E</sub>X the tags for equations are typically generated automatically and contain a printed representation of the L<sup>A</sup>T<sub>E</sub>X counter `equation`. This involves three processes: setting (normally by incrementing) the value of the `equation` counter; formatting the tag; and printing it in the correct position.

In practice, the first two processes are nearly always linked. Thus, the value of the `equation` counter is increased only when a tag containing its representation is automatically printed. For example, when a mathematical display environment has both starred and unstarred forms, the unstarred form automatically tags each logical equation while the starred form does not. Only in the unstarred form is the value of the `equation` counter changed.

Within the unstarred forms the setting of a tag (and the incrementing of the counter value) for any particular logical equation can be suppressed by putting `\notag` (or `\nonumber`<sup>1</sup>) before the `\backslash\backslash`. You can override the default automatic tag with one of your own design (or provide a new one) by using the command `\tag` before the `\backslash\backslash`. The argument of this command can be arbitrary normal text that is typeset (within the normal parentheses) as the tag for that equation.

Note that the use of `\tag` suppresses the incrementing of the counter value. Thus, the default tag setting is only visually the same as `\tag{\theequation}`; they are not equivalent forms. The starred form, `\tag*`, causes the text in its argument to be typeset without the parentheses (and without any other material that might otherwise be added with a particular document class).

```
\usepackage{amsmath}
\begin{aligned}
x^2 + y^2 &= z^2 & (1) & \begin{aligned} &x^2+y^2 &=& z^2 \label{eq:A} \\ &x^3+y^3 &=& z^3 \notag \\ &x^4+y^4 &=& r^4 &\tag{\$**\$} \\ &x^5+y^5 &=& r^5 &\tag{\$**\$} \\ &x^6+y^6 &=& r^6 &\tag{\ref{eq:A}$$} \end{aligned} & 8-2-25 \\
x^3 + y^3 &= z^3 & (*) & \begin{aligned} &x^2+y^2 &=& z^2 \label{eq:A} \\ &x^3+y^3 &=& z^3 \notag \\ &x^4+y^4 &=& r^4 &\tag{\$**\$} \\ &x^5+y^5 &=& r^5 &\tag{\$**\$} \\ &x^6+y^6 &=& r^6 &\tag{\ref{eq:A}$$} \end{aligned} \\
x^4 + y^4 &= r^4 & (*) & \begin{aligned} &x^2+y^2 &=& z^2 \label{eq:A} \\ &x^3+y^3 &=& z^3 \notag \\ &x^4+y^4 &=& r^4 &\tag{\$**\$} \\ &x^5+y^5 &=& r^5 &\tag{\$**\$} \\ &x^6+y^6 &=& r^6 &\tag{\ref{eq:A}$$} \end{aligned} \\
x^5 + y^5 &= r^5 & * & \begin{aligned} &x^2+y^2 &=& z^2 \label{eq:A} \\ &x^3+y^3 &=& z^3 \notag \\ &x^4+y^4 &=& r^4 &\tag{\$**\$} \\ &x^5+y^5 &=& r^5 &\tag{\$**\$} \\ &x^6+y^6 &=& r^6 &\tag{\ref{eq:A}$$} \end{aligned} \\
x^6 + y^6 &= r^6 & (1') & \begin{aligned} &x^2+y^2 &=& z^2 \label{eq:A} \\ &x^3+y^3 &=& z^3 \notag \\ &x^4+y^4 &=& r^4 &\tag{\$**\$} \\ &x^5+y^5 &=& r^5 &\tag{\$**\$} \\ &x^6+y^6 &=& r^6 &\tag{\ref{eq:A}$$} \end{aligned} \\
A_1 = N_0(\lambda; \Omega') - \phi(\lambda; \Omega') & & (2) & \begin{aligned} &A_1 &=& N_0(\lambda; \Omega') - \phi(\lambda; \Omega') \\ &&& - \phi(\lambda; \Omega') \end{aligned} \\
A_2 = \phi(\lambda; \Omega') \phi(\lambda; \Omega) & & \text{ALSO } (2) & \begin{aligned} &A_2 &=& \phi(\lambda; \Omega') \\ &&& \phi(\lambda; \Omega) \end{aligned} \\
A_3 = \mathcal{N}(\lambda; \omega) & & (3) & \begin{aligned} &A_3 &=& \mathcal{N}(\lambda; \omega) \end{aligned} \\
\end{aligned}
```

Notice this example's use of the `\label` and `\ref` commands to provide some kinds of "relative numbering" of equations.

To facilitate the creation of cross-references to equations, the `\eqref` command (used in Example 8-2-29 on page 485), automatically adds the parentheses around the equation number, adding an italic correction if necessary. See also Section 2.4 on page 66 for more general solutions to managing references.

<sup>1</sup>The command `\notag` is interchangeable with `\nonumber`.

### 8.2.12 Fine-tuning tag placement

Optimal placement of equation number tags can be a rather complex problem in multiple-line displays. These display environments try hard to avoid overprinting an equation number on the equation contents; if necessary, the number tag is moved down or up, onto a separate line. The difficulty of accurately determining the layout of a display can occasionally result in a tag placement that needs further adjustment. Here is an example of the kind of thing that can happen, and a strategy for fixing it. The automatic tag placement is clearly not very good.

```
\usepackage{amsmath}
\begin{equation} \begin{aligned}
\lvert I_2 \rvert &\leq \left| \left( u(a, t) - \int_{\gamma(t)}^a \frac{d\theta}{k}(\theta, t) \int_a^\theta c(\xi) u_t(\xi, t) d\xi \right) dt \right| \\
&\quad && \\ 
&\quad &\quad & \& C_6 \Bigg| \lvert f \int_\Omega \widetilde{S}_{a,-}^{-1,0} W_2(\Omega, \Gamma_l) \rvert \Bigg| \Bigg| \lvert u \rvert \overset{\circ}{\rightarrow} W_2^{\tilde{A}}(\Omega; \Gamma_r, T) \Bigg| \\
\end{aligned} \end{equation}
```

$$\boxed{8.2-26} \quad \begin{aligned}
\lvert I_2 \rvert &= \left| \int_0^T \psi(t) \left\{ u(a, t) - \int_{\gamma(t)}^a \frac{d\theta}{k}(\theta, t) \int_a^\theta c(\xi) u_t(\xi, t) d\xi \right\} dt \right| \\
&\leq C_6 \left| \left| f \int_\Omega \widetilde{S}_{a,-}^{-1,0} W_2(\Omega, \Gamma_l) \right| \right| \left| \lvert u \rvert \overset{\circ}{\rightarrow} W_2^{\tilde{A}}(\Omega; \Gamma_r, T) \right| \\
\end{aligned} \tag{1}$$

A fairly easy way to improve the appearance of such an equation is to use an `align` environment with a `\notag` on the first equation line:

```
\begin{aligned}
\lvert I_2 \rvert &\leq \left| \left( u(a, t) - \int_{\gamma(t)}^a \frac{d\theta}{k}(\theta, t) \int_a^\theta c(\xi) u_t(\xi, t) d\xi \right) dt \right| \\
&\quad \dots && \notag \\ 
&\quad &\quad & \& C_6 \Bigg| \lvert f \int_\Omega \widetilde{S}_{a,-}^{-1,0} W_2(\Omega, \Gamma_l) \rvert \Bigg| \Bigg| \lvert u \rvert \overset{\circ}{\rightarrow} W_2^{\tilde{A}}(\Omega; \Gamma_r, T) \Bigg| \\
\end{aligned}
```

This produces a good result but note that it misuses logical markup—it assumes the equation numbers to be on the right!

$$\begin{aligned}
 |I_2| &= \left| \int_0^T \psi(t) \left\{ u(a,t) - \int_{\gamma(t)}^a \frac{d\theta}{k}(\theta,t) \int_a^\theta c(\xi) u_t(\xi,t) d\xi \right\} dt \right| \\
 &\leq C_6 \left\| f \int_\Omega \left| \tilde{S}_{a,-}^{-1,0} W_2(\Omega, \Gamma_l) \right| \left\| |u| \overset{\circ}{\rightarrow} W_2^{\tilde{A}}(\Omega; \Gamma_r, T) \right\| \right\| \quad (1)
 \end{aligned}$$

8-2-27

A `\raisetag` command is available that will further adjust the vertical position of the current equation number but *only* when it has been automatically moved from its “normal position”. For example, to move such a tag upward<sup>1</sup> by 6pt, you could write `\raisetag{6pt}`. You can try adjusting the above equation with `\raisetag` but the correct value is not easy to divine: a value of `1.2\baselineskip` looks about right!

A more sensible use is shown in the next example, where `\raisetag` with a negative argument is used to move the tag on the left down into the display.

$$\begin{aligned}
 (1) \quad \text{The sign function: } S(x) = & \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases} \\
 & \begin{aligned}
 \usepackage[leqno]{amsmath} \\
 \begin{gathered}
 \text{\texttt{\textbackslash usepackage[leqno]\{amsmath}}} \\
 \text{\texttt{\textbackslash begin\{gather\} \textbackslash raisetag\{-10pt\}}} \\
 \text{\texttt{\textbackslash text\{The sign function: \}}} \\
 \text{\texttt{\textbackslash mathcal\{S\}(x) = \begin\{cases\}}} \\
 \text{\texttt{\textbackslash end\{cases\}}} \\
 \text{\texttt{\textbackslash end\{gather\}}}
 \end{gathered}
 \end{aligned}
 \end{aligned}$$

8-2-28

Here we used a `gather` environment with a single line because the `equation` environment is (the only) one within which `\raisetag` unfortunately has no effect (it is coded using low-level `TeX`).

These kinds of adjustment constitute “fine-tuning”, like line breaks and page breaks in text. They should therefore be left until your document is nearly finalized. Otherwise, you may end up redoing the fine-tuning several times to keep up with changing document content.

### 8.2.13 Subordinate numbering sequences

The `amsmath` package provides a `subequations` environment to support “equation sub-numbering” with tags of the form (2a), (2b), (2c), and so on. All the tagged equations within it use this sub-numbering scheme based on two normal `TeX` counters: `parentequation` and `equation`.

<sup>1</sup>The description in the file `amsmath.dtx` seems to indicate that a positive value should always move the tag toward the “normal position”—that is, downward for tags on the left, but the current implementation does not work in this way.

The next example demonstrates that the tag can be redefined to some extent, but note that the redefinition for `\theequation` must appear within the `subequations` environment! (Appendix A.1.4 discusses counter manipulations.)

```
\usepackage{amsmath}
\begin{subequations} \label{eq:1}
\begin{aligned} f &= g & (1a) & \begin{aligned} f &\triangleq g & \label{eq:1A} \\ f' &\triangleq g' & \label{eq:1B} \end{aligned} \\
f' &= g' & (1b) & \mathcal{L}f \triangleq \mathcal{L}g \label{eq:1C} \\
\mathcal{L}f &= \mathcal{L}g & (1c) & \end{aligned}
\end{subequations}
\begin{subequations} \label{eq:2}
\renewcommand{\theequation}{\theparentequation\,\text{roman}\{equation\}}
\begin{aligned} f &= g & (2i) & \begin{aligned} f &\triangleq g & \label{eq:2A} \\ f' &\triangleq g' & \label{eq:2B} \end{aligned} \\
f' &= g' & (2ii) & \mathcal{L}f \triangleq \mathcal{L}g + K \label{eq:2C} \\
\mathcal{L}f &= \mathcal{L}g + K & (2iii) & \end{aligned}
\end{subequations}
```

Note the relationship between (1) and (2): only 1c and 2iii differ.

8-2-29

Note the relationship between `\eqref{eq:1}` and `\eqref{eq:2}`: only `\ref{eq:1C}` and `\ref{eq:2C}` differ.

The `subequations` environment must appear *outside* the displays that it affects. Also, it should not be nested within itself. Each use of this environment advances the “main” equation counter by one. A `\label` command within the `subequations` environment but outside any individual (logical) equation will produce a `\ref` to the parent number (e.g., to 2 rather than 2i).

### 8.2.14 Resetting the equation counter

It is fairly common practice to have equations numbered within sections or chapters, using tags such as (1.1), (1.2), ..., (2.1), (2.2), .... With `amsmath` this can easily be set up by using the declaration `\numberwithin{equation}{section}`.<sup>1</sup>

For example, to get compound equation tags including the section number, with the equation counter being automatically reset for each section, put this declaration in the preamble: `\numberwithin{equation}{section}`.

## 8.3 Matrix-like environments

The `amsmath` package offers a number of matrix-like environments, all of which are similar to `array` in syntax and layout. Thinking of complex mathematical layouts in this way is a useful exercise, as quite a wide variety of two-dimensional mathematical structures and table-like layouts can be so described.

<sup>1</sup>As the name implies, `\numberwithin` can be applied to any pair of counters, but the results may not be satisfactory in all cases because of potential complications. See the discussion of the `\@addtoreset` command in Appendix A.1.4.

*Old commands disabled* Three of these environments replace old commands that are kept well hidden in standard L<sup>A</sup>T<sub>E</sub>X; `cases` (discussed in the next section) and `matrix` and `pmatrix` (discussed in the section after that). Because these old command forms use a totally different notation, they are not truly part of L<sup>A</sup>T<sub>E</sub>X and they cannot be mixed with the environment forms described here. Indeed, `amsmath` will produce an explanatory error message if one of the old commands is used (see page 907). If, contrariwise, you make the mistake of using the `amsmath` environment forms without loading that package, then you will most probably get this error message: “Misplaced alignment tab character &”.

### 8.3.1 The `cases` environment

Constructions like the following, where a single equation has a few variants, are very common in mathematics. To handle these constructions, `amsmath` provides the `cases` environment. It produces a decorated array with two columns, both left aligned.

$$P_{r-j} = \begin{cases} 0 & \text{if } r - j \text{ is odd,} \\ r! (-1)^{(r-j)/2} & \text{if } r - j \text{ is even.} \end{cases} \quad (1)$$

```
\usepackage{amsmath}
\begin{equation} P_{r-j} = \begin{cases} 0 & \text{\text{if } $r - j$ is odd,} \\ r! \, (-1)^{(r-j)/2} & \text{\text{if } $r - j$ is even.} \end{cases} \end{equation}
```

[ 8-3-1 ]

Notice the use of `\text` and the “embedded math mode” in the text strings. With the help of the `aligned` environment, other environments similar to `cases` can be defined, as in Example 8-2-19 on page 478.

### 8.3.2 The matrix environments

The matrix environments are similar to L<sup>A</sup>T<sub>E</sub>X’s `array`, except that they do not have an argument specifying the formats of the columns. Instead, a default format is provided: up to 10 centered columns. Also, the spacing differs slightly from the default in `array`. The example below illustrates the matrix environments `matrix`, `pmatrix`, `bmatrix`, `Bmatrix`, `vmatrix`, and `Vmatrix`.<sup>1</sup>

```
\usepackage{amsmath}
\begin{gather*}
\begin{matrix} 0 & 1 & \left( \begin{matrix} 0 & -i \\ i & 0 \end{matrix} \right) \\ 1 & 0 & \left( \begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix} \right) \\ \left[ \begin{matrix} 0 & -1 \\ 1 & 0 \end{matrix} \right] & \left\{ \begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix} \right\} \\ \left| \begin{matrix} a & b \\ c & d \end{matrix} \right| & \left\| \begin{matrix} i & 0 \\ 0 & -i \end{matrix} \right\| \end{matrix}
\begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix}
\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}
\begin{Bmatrix} 1 & 0 \\ 0 & -1 \end{Bmatrix}
\begin{vmatrix} a & b \\ c & d \end{vmatrix}
\begin{Vmatrix} i & 0 \\ 0 & -i \end{Vmatrix}
\end{gather*}
```

[ 8-3-2 ]

<sup>1</sup>Note the warning above about possible problems when using `matrix` and `pmatrix`.

The maximum number of columns in a matrix environment is determined by the counter `MaxMatrixCols`, which you can change using L<sup>A</sup>T<sub>E</sub>X's standard counter commands. As in standard arrays, the amount of space between the columns is given by the value of `\arraycolsep`, but no space is added on either side of the array. With more columns L<sup>A</sup>T<sub>E</sub>X has to work a little harder and needs slightly more resources. However, with today's typical T<sub>E</sub>X implementations such limits are less important, so setting it to 20 or even higher is possible without a noticeable change in processing speed.

8-3-3

```
\usepackage{amsmath}
\setcounter{MaxMatrixCols}{20}
\begin{Vmatrix}
a & b & c & d & e & f & g & h & i & j & \cdots \\
a & b & c & d & e & f & g & h & i & j & \cdots \\
a & b & c & d & e & f & g & h & i & j & \cdots \\
a & b & c & d & e & f & g & h & i & j & \cdots \\
\ddots & \ddots
\end{Vmatrix}
```

This example also demonstrates use of the command `\hdotsfor` to produce a row of dots in a matrix, spanning a given number of columns (here 5). The spacing of the dots can be varied by using the optional parameter (here 2) to specify a multiplier for the default space between the dots; the default space between dots is 3 math units (see Appendix A.1.5). The thin space and the brace group `\,{}()` at the end of each row simply make the layout look better; together they produce two thin spaces, about 6mu or 1/3em. (Spacing in formulas is discussed in more detail in Section 8.7.6 on page 507.)

To produce a small matrix suitable for use in text, use the `smallmatrix` environment. Note that the text lines are not spread apart even though the line before the small matrix contains words with descenders.

To show the effect of the matrix on surrounding lines inside a paragraph, we put it here:  $(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix})$  and follow it with enough text to ensure that there is at least one full line below the matrix.

8-3-4

```
\usepackage{amsmath}
To show the effect of the matrix on surrounding
lines inside a paragraph, we put it here:
$ \left( \begin{smallmatrix}
1 & 0 \\ 0 & -1
\end{smallmatrix} \right) $ and follow it with enough text to ensure that
there is at least one full line below the matrix.
```

### 8.3.3 Stacking in subscripts and superscripts

The `\substack` command is most commonly used to typeset several lines within a subscript or superscript, using `\backslash` as the row delimiter.

A slightly more general structure is the `subarray` environment, which allows you to specify that the lines should be left or right aligned instead of centered.

Note that both environments need to be surrounded by braces when they appear as a subscript or superscript.

$$\sum_{\substack{0 \leq i \leq m \\ 0 < j < n}} P(i, j) \quad (1)$$

$$\sum_{\substack{i \in \Lambda \\ 0 \leq i \leq m \\ 0 < j < n}} P(i, j) \quad (2)$$

```
\usepackage{amsmath}
\begin{gather}
\sum_{\substack{0 \leq i \leq m \\ 0 < j < n}} P(i, j) \\
\sum_{\substack{i \in \Lambda \\ 0 \leq i \leq m \\ 0 < j < n}} P(i, j)
\end{gather}
```

8-3-5

### 8.3.4 Commutative diagrams

Some commands for producing simple commutative diagrams based on arrays are available in a separate package, `amscd`. It provides some useful shorthand forms for specifying the decorated arrows and other connectors. However, it is very limited—for example, these connectors can be only horizontal and vertical.

The `picture` environment could be used for more complex commutative diagrams but for most serious work in this area you will need one of the more comprehensive packages. These include Kristoffer Rose's XY-pic system (see [57, chapter 5]) and its extension [11] by Michael Barr; the `diagram` system [22, 23] by Francis Borceux; and the `kuvio` package [155] by Anders Svensson.

In the `CD` environment the notations `@>>>`, `@<<<`, `@VVV`, and `@AAA` give right, left, down, and up arrows, respectively.<sup>1</sup> The following examples also show the use of the command `\DeclareMathOperator` (see Section 8.6.2).

$$\begin{array}{ccccc} \text{cov}(L) & \longrightarrow & \text{non}(K) & \longrightarrow & \text{cf}(K) \\ \downarrow & & \uparrow & & \uparrow \\ \text{add}(L) & \longrightarrow & \text{add}(K) & \longrightarrow & \text{cov}(K) \end{array}$$

```
\usepackage{amsmath, amscd}
\DeclareMathOperator{\add}{add}
\DeclareMathOperator{\cf}{cf}
\DeclareMathOperator{\cov}{cov}
\DeclareMathOperator{\non}{non}
\begin{CD}
@>>> @>>> @>>> \\
@AAA @AAA @AAA \\
@>>> @>>> @>>>
\end{CD}
```

8-3-6

Decorations on the arrows are specified as follows. For the horizontal arrows, material between the first and second `>` or `<` symbols will be typeset as a superscript, and material between the second and third will be typeset as a subscript. Similarly, material between the first and second, or second and third, As or Vs of vertical arrows will be typeset as left or right “side-scripts”; this format is used in the next example to place the operator `End P` to the right of the arrow.

The notations `@=` and `@|` give horizontal and vertical double lines.

<sup>1</sup>For keyboards lacking the characters `<` and `>`, the notations `@))` and `@((` are alternatives.

A “null arrow” (produced by `\cdot`) can be used instead of a visible arrow to fill out an array where needed.

[8-3-7]

$$\begin{array}{ccc} S^{W_\Lambda} \otimes T & \xrightarrow{j} & T \\ \downarrow & & \downarrow \text{End } P \\ (S \otimes T)/I & \xlongequal{\cdot} & (Z \otimes T)/J \end{array}$$

```
\usepackage{amsmath,amscd}
\DeclareMathOperator{\End}{End}
\begin{CD}
S^{\mathcal{W}_\Lambda} \otimes T @>j>> T \\
@VVV @VV{\text{End } P}V \\
(S \otimes T)/I @= (Z \otimes T)/J
\end{CD}
```

A similar layout, which does not look nearly as good, can be produced in standard L<sup>A</sup>T<sub>E</sub>X:

[8-3-8]

$$\begin{array}{ccc} S^{W_\Lambda} \otimes T & \xrightarrow{j} & T \\ \downarrow & & \downarrow \text{End } P \\ (S \otimes T)/I & = & (Z \otimes T)/J \end{array}$$

```
\begin{array}{ccc}
S^{\mathcal{W}_\Lambda} \otimes T & \xrightarrow{j} & T \\
\Big\downarrow & & \Big\downarrow \text{End } P \\
(S \otimes T)/I & = & (Z \otimes T)/J
\end{array}
```

This example shows clearly how much better the results are with the `amscd` package: the notation is enormously easier and, for example, the package produces longer horizontal arrows and much improved spacing between elements of the diagram. The more specialized packages will enable you to get even more beautiful results.

### 8.3.5 delarray—Delimiters surrounding an array

This section describes a useful general extension to the `array` package (see Section 5.2 on page 243) that allows the user to specify opening and closing extensible delimiters (see Section 8.5.3) to surround a mathematical `array` environment. The `delarray` package was written by David Carlisle, and its use is illustrated in the next, rather odd-looking, example (note that the `delarray` package is independent of `amsmath` but it automatically loads the `array` package if necessary).

[8-3-9]

$$\mathcal{Q} = \left( \begin{array}{cc} X & Y \end{array} \right) \left[ \begin{array}{cc} A & B \\ C & D \end{array} \right] \left( \begin{array}{c} L \\ M \end{array} \right)$$

```
\usepackage{delarray}
\mathcal{Q} = \left[ \begin{array}{cc} X & Y \end{array} \right] \left[ \begin{array}{cc} A & B \\ C & D \end{array} \right] \left[ \begin{array}{c} L \\ M \end{array} \right]
```

The delimiters are placed on either side of the “preamble declaration” (here `{cc}`). They must be delimiters from Table 8.3 on page 498.

The most useful feature of this package is also illustrated in the preceding example: the use of the [t] and [b] optional arguments, which are not available with `amsmath`'s matrix environments. These show that use of the `delarray` syntax is not equivalent to surrounding the `array` environment with `\left` and `\right`, since the delimiters are raised as well as the array itself.

## 8.4 Compound structures and decorations

This section presents some commands that produce a variety of medium-sized mathematical structures including decorated symbols and fraction-like objects.

### 8.4.1 Decorated arrows

The commands `\xleftarrow` and `\xrightarrow` produce horizontal relation arrows similar to those used for the commutative diagrams in Section 8.3.4; they are intended to have textual decorations above and/or below the arrow and the length of the arrow is chosen automatically to accommodate the text. These arrows are normally available in only one size. Thus, they will probably not be suited for use in fractions, subscripts, or superscripts, for example.

The textual decorations below and above the arrows are specified in an optional and a mandatory argument to the command.

```
\usepackage{amsmath}
0 \xleftarrow[\zeta]{\partial_0 \alpha(b)} E^{\partial_0 b} \quad 8-4-1
\[
  0 \xleftarrow[\zeta]{\partial_0 \alpha(b)} F \times \Delta(n - 1)
  \xrightarrow{\partial_0 b} \partial_0 \alpha(b) E^{\partial_0 b}
\]
```

### 8.4.2 Continued fractions

The `\cfrac` command produces fraction arrays known as “continued fractions”. By default, each numerator formula is centered; left or right alignment of a numerator is achieved by adding the optional argument [l] or [r].

```
\usepackage{amsmath}
\begin{equation*}
\sqrt{2} + \cfrac{1}{\sqrt{3} + \cfrac{1}{\sqrt{4} + \cfrac{1}{\sqrt{5} + \cfrac{1}{\sqrt{6} + \dots}}}} \quad 8-4-2
\begin{aligned}
&\quad \backslashbegin\{array}{c} 1 \\ \hline \end{array} \\
&\quad \backslashcfrac{1}{\sqrt{2}} + \backslashcfrac{1}{\sqrt{3}} + \backslashcfrac{1}{\sqrt{4}} + \backslashcfrac{1}{\sqrt{5}} + \backslashcfrac{1}{\sqrt{6}} + \dots
\end{aligned}
\end{equation*}
```

### 8.4.3 Boxed formulas

The command `\boxed` puts a box around its argument; it works just like `\fbox`, except that the contents are in math mode. See also the commands described in Section 10.1.

8-4-3 
$$W_t - F \subseteq V(P_i) \subseteq W_t \quad (1)$$

```
\usepackage{amsmath}
\begin{equation}
\boxed{W_t - F \subseteq V(P_i) \subseteq W_t}
\end{equation}
```

### 8.4.4 Limiting positions

Subscripts and superscripts on integrals, sums, or other operators can be placed either above and below the mathematical operator or in the normal sub/super positions on the right of the operator. They are said to “take limits” if the superscript and subscript material is placed (in the “limit positions”) above and below the symbol or operator name. Typically, no limits are used in text (to avoid spreading lines apart); in a display, the placement depends on the operator used. The default placements in L<sup>A</sup>T<sub>E</sub>X are illustrated in the following example.

8-4-4 
$$\sum_{i=1}^n \int_0^\infty \lim_{n \rightarrow 0} \quad \text{Text: } \sum_{i=1}^n, \int_0^\infty, \lim_{n \rightarrow 0}.$$

```
\sum_{i=1}^n \int_0^\infty \lim_{n \rightarrow 0}
\begin{bmatrix}
\sum_{i=1}^n & \int_0^\infty & \lim_{n \rightarrow 0}
\end{bmatrix}
```

The placement of subscripts and superscripts on integrals, sums, and other operators is often dictated by the house-style of a journal. Recognizing this fact, `amsmath` offers a long list of options for controlling the positioning. In the following summary, *default* indicates what happens when the `amsmath` package is used with a standard L<sup>A</sup>T<sub>E</sub>X class but without any of these options.<sup>1</sup>

**intlimits, nointlimits** In displayed equations only, place superscripts and subscripts of integration-type symbols above and below or at the side (default), respectively.

**sumlimits, nosumlimits** In displayed equations only, place superscripts and subscripts of summation-type symbols (also called “large operators”) above and below (default) or at the side, respectively. This option also affects other big operators— $\prod$ ,  $\coprod$ ,  $\otimes$ ,  $\oplus$ , and so forth—but not integrals.

**namelimits, nonamelimits** Like `sumlimits` or `nosumlimits` but for certain “operator names”, such as `det`, `inf`, `lim`, and `max`, `min`, that traditionally have subscripts placed underneath, at least when they occur in a displayed equation.

<sup>1</sup>But not necessarily when using the `AMSLATEX` document classes.

The positioning on individual symbols/names can be controlled directly by placing one of the following TeX primitive commands immediately after the symbol or operator name: `\limits`, `\nolimits`, or `\displaylimits`. This last command, which specifies that the operator “takes limits” only when the mathematical style is a display style, is the default whenever a symbol of class Operator<sup>1</sup> appears or a `\mathop` construction is used. If an operator is to “take limits” outside a display, then this must be declared individually using the `\limits` command. Compare the next example to Example 8-4-4, noting that some commands show no effect as they merely reinforce the default.

$$\sum_{i=1}^n \int_0^\infty \lim_{n \rightarrow 0} \quad \text{Text: } \sum_{i=1}^n \int_0^\infty \lim_{n \rightarrow 0}.$$

```
\[
  \sum\nolimits_{i=1}^n \qquad \int\limits_0^\infty \lim\limits_{n \rightarrow 0}
\]
Text: $\\sum\\nolimits_{i=1}^n$, $\\int\\limits_0^\\infty$,
      $\\lim\\limits_{n \\rightarrow 0}$.
```

8-4-5

#### 8.4.5 Multiple integral signs

The commands `\iint`, `\iiint`, and `\iiiiint` give multiple integral signs with well-adjusted spaces between them, in both running text and displays. The command `\idotsint` gives two integral signs with ellipsis dots between them. The following example also shows the use of `\limits` to override the default for integral constructions and place the limit  $V$  underneath the symbol.

$$\begin{aligned} & \iint_V \mu(v, w) du dv \\ & \iiint_V \mu(u, v, w) du dv dw \\ & \iiiiint_V \mu(t, u, v, w) dt du dv dw \\ & \int_V \dots \int \mu(z_1, \dots, z_k) dz \end{aligned} \quad \begin{aligned} & \text{\usepackage{amsmath}} \\ & \begin{aligned} & \text{\begin{gather*}} \\ & \quad \iint \limits_V \mu(v, w) \\ & \quad \backslash du \backslash dv \quad \backslash \backslash \\ & \quad \iiint \limits_V \mu(u, v, w) \\ & \quad \backslash du \backslash dv \backslash dw \quad \backslash \backslash \\ & \quad \iiiiint \limits_V \mu(t, u, v, w) \\ & \quad \backslash dt \backslash du \backslash dv \backslash dw \quad \backslash \backslash \\ & \quad \idotsint \limits_V \mu(z_1, \dots, z_k) \\ & \quad \backslash \mathbf{dz} \end{aligned} \\ & \text{\end{gather*}} \end{aligned}$$

8-4-6

#### 8.4.6 Modular relations

The commands `\mod`, `\bmod`, `\pmod`, and `\pod` are provided by the `amsmath` package to deal with the special spacing conventions of the “mod” notation for equivalence classes of integers. Two of these commands, `\mod` and `\pod`, are variants of `\pmod` that are preferred by some authors; `\mod` omits the parentheses, whereas

<sup>1</sup>See Section 8.9.1 on page 524 for a discussion of the various mathematical classes of symbols.

\pod omits the “mod” and retains the parentheses. With amsmath the spacing of \pmod is decreased within a non-display formula.

[8-4-7]

$$\begin{aligned} u &\equiv v + 1 \pmod{n^2} \\ u &\equiv v + 1 \bmod{n^2} \\ u &= v + 1 \pmod{n^2} \\ u &= v + 1 \pod{n^2} \end{aligned}$$

The in-text layout:  $u = v + 1 \pmod{n^2}$

$$\begin{aligned} (m \bmod n) &= k^2; & x &\equiv y \pmod b; \\ x &\equiv y \bmod c; & x &\equiv y \pod d. \end{aligned}$$

```
\usepackage{amsmath}
\begin{aligned*}
u &\equiv v + 1 \pmod{n^2} \\
u &\equiv v + 1 \bmod{n^2} \\
u &= v + 1 \pmod{n^2} \\
u &= v + 1 \pod{n^2}
\end{aligned*}
The in-text layout: $ u = v + 1 \pmod{n^2} $ \\
\begin{gather*}
(m \bmod n) = k^2 \ , \ ; \quad x \equiv y \pmod b \ , \ ; \quad \\
x \equiv y \bmod c \ , \ ; \quad x \equiv y \pod d \ .
\end{gather*}
```

#### 8.4.7 Fractions and generalizations

In addition to the common \frac, the amsmath package provides \dfrac and \tfrac as convenient abbreviations for {\displaystyle\frac ...} and {\textstyle\frac ...} (mathematical styles are discussed in more detail in Section 8.7.1 on page 502).

[8-4-8]

$$\frac{1}{k} \log_2 c(f) - \frac{1}{k} \log_2 c(f) \quad (1)$$

Text:  $\sqrt{\frac{1}{k} \log_2 c(f)} - \sqrt{\frac{1}{k} \log_2 c(f)}$ .

```
\usepackage{amsmath}
\begin{equation} \frac{1}{k} \log_2 c(f) - \frac{1}{k} \log_2 c(f) \end{equation}
Text: $ \sqrt{ \frac{1}{k} \log_2 c(f) } - \sqrt{ \frac{1}{k} \log_2 c(f) } $.
```

For binomial coefficients such as  $\binom{n}{k}$ , use the similar commands \binom, \dbinom, and \tbinom.

[8-4-9]

$$\binom{k}{2} 2^{k-1} + \binom{k-1}{2} 2^{k-2} \quad (1)$$

Text:  $\binom{k}{2} 2^{k-1} + \binom{k-1}{2} 2^{k-2}$ .

```
\usepackage{amsmath}
\begin{equation} \binom{k}{2} 2^{k-1} + \binom{k-1}{2} 2^{k-2} \end{equation}
Text: $ \binom{k}{2} 2^{k-1} + \binom{k-1}{2} 2^{k-2} $.
```

All of these \binom and \frac commands are special cases of the generalized fraction command \genfrac, which has six parameters.

\genfrac{l delim}{r delim}{thick}{style}{num}{denom}

The first two parameters, *l delim* and *r delim*, are the left and right delimiters, respectively. They must be either both empty or both non-empty; to place a single

<i>Style</i>	<i>Default Thickness (approximately)</i>
text/display	0.40pt
script	0.34pt
scriptscript	0.24pt

Table 8.2: Default rule thickness in different math styles

delimiter, use a period “.” on the “empty” side. The third parameter, *thick*, is used to override the default thickness of the fraction rule; for instance, `\binom` uses 0pt for this argument so that the line is invisible. If it is left empty, the line thickness has the default value specified by the font set-up in use for mathematical typesetting. The examples in this chapter use the defaults listed in Table 8.2 in the various styles (see also Section 8.7.1).

The fourth parameter, *style*, provides a “mathematical style override” for the layout and font sizes used. It can take integer values in the range 0–3 denoting `\displaystyle`, `\textstyle`, `\scriptstyle`, and `\scriptscriptstyle`, respectively. If this argument is left empty, then the style is selected according to the normal rules for fractions (described in Table 8.5 on page 502). The last two arguments are simply the numerator (*num*) and denominator (*denom*).

To illustrate, here is how `\frac`, `\tfrac`, and `\binom` might be defined:

```
\newcommand\frac [2]{\genfrac {}{}{0pt}{}{#1}{#2}}
\newcommand\tfrac[2]{\genfrac {}{}{0pt}{}{#1}{#2}}
\newcommand\binom[2]{\genfrac {}{}{0pt}{}{#1}{#2}}
```

Of course, if you want to use a particular complex notation (such as one implemented with `\genfrac`) repeatedly throughout your document, then you will do yourself (and your editor) a favor if you define a meaningful command name with `\newcommand` as an abbreviation for that notation, as in the examples above.

The old generalized fraction commands `\over`, `\overwithdelims`, `\atop`, `\atopwithdelims`, `\above`, and `\abovewithdelims` (inherited in standard L<sup>A</sup>T<sub>E</sub>X from primitive T<sub>E</sub>X) produce warning messages if they are used with the `amsmath` package.

#### 8.4.8 Dottier accents

The `\dot` and `\ddot` mathematical accents are supplemented by `\ddot{\dots}` and `\dddot{\dots}`, giving triple and quadruple dot accents, respectively.

```
\dot{S} \ddot{P} \ddot{\ddot{Q}} \ddot{\ddot{\ddot{R}}} \usepackage{amsmath} $ \dot{S} \quad \ddot{P} \quad \ddot{\ddot{Q}} \quad \ddot{\ddot{\ddot{R}}} $ | 8-4-10
```

If you want to set up your own mathematical accents, then you should probably use the `accents` package developed by Javier Bezos. It provides methods

of defining “faked” accents (see `\accentset` in the example) and general under-accents (`\underaccent`, `\undertilde`), along with other features. It can be used together with `amsmath`. For further details see [20].

8-4-11	$\hat{X}$	$\hat{\hat{h}}$	$\hat{\mathcal{M}}$	$\hat{\mathcal{C}}$	$\underline{M}$	$\underline{\underline{ABC}}$
	<pre>\usepackage{accents} \[\ \accentset{\ast}{X} \quad \hat{\hat{h}} \quad \hat{\mathcal{M}} \quad \hat{\mathcal{C}} \quad \underline{M} \quad \underline{\underline{ABC}}</pre>					
	<pre>\quad \hat{\hat{\hat{h}}} \quad \hat{\mathcal{M}} \quad \hat{\mathcal{C}} \quad \underline{\underline{M}} \quad \underline{\underline{\underline{ABC}}}</pre>					
	<pre>\quad \hat{\hat{\hat{\hat{h}}}} \quad \hat{\mathcal{M}} \quad \hat{\mathcal{C}} \quad \underline{\underline{\underline{M}}} \quad \underline{\underline{\underline{\underline{ABC}}}}</pre>					

#### 8.4.9 amsxtra—Accents as superscripts

One feature available with this package is a collection of simple commands for placing accents as superscripts to a sub-formula:

8-4-12	$(xyz)^{\cdots}$	$(xyz)^{\cdot\cdot}$	$(xyz)^{\cdot}$	$\usepackage{amsxtra}$
	$(xyz)^{\circ}$	$(xyz)^{\vee}$	$(xyz)^{\wedge}$	$\$(xyz)\spddot\$ \quad \$\$(xyz)\spdot\$ \quad \$\$(xyz)\spcheck\$ \\\$$
	$(xyz)^{\sim}$	$(xyz)^{\sim}$		$\$(xyz)\sphat\$ \quad \$\$(xyz)\stilde\$$

#### 8.4.10 Extra decorations

Standard  $\text{\LaTeX}$  provides `\stackrel` for placing a superscript above a Relation symbol. The `amsmath` package makes the commands `\overset` and `\underset` available as well. They can be used to place material above or below any Ordinary symbol or Binary operator symbol, in addition to Relation symbols; they are typeset just like the limits above and below a summation sign.

The command `\sideset` serves a special purpose, complementary to the others: it adds decorations additional to the “normal” limits (which are set above and below) to any Operator symbol such as  $\sum$  or  $\prod$ . These are placed in the subscript and superscript positions, on both the left and right of the Operator.

8-4-13	$\overset{*}{X} > \underset{*}{X} \iff \sum'_{a,b \in R^*} \frac{a}{b} = X$	$\usepackage{amsmath}$
		$\quad \backslash[\ \overset{*}{X} > \underset{*}{X}$
		$\quad \quad \quad \text{iff } \sideset{}{}{\sum_{a,b \in \mathbf{R}^*}} = X \]$

This more complex example shows how to fully decorate a product symbol.

8-4-14	$\prod_{i=1}^n \prod_{k>1}^m T_{i,j}^k$	$\usepackage{amsmath}$
		$\quad \backslash[\ \sideset{_i=1^n}{_j=2^m}{\prod_{k>1}} \mathcal{T}_{i,j}^k \]$

## 8.5 Variable symbol commands

Many  $\text{\LaTeX}$  commands are often thought of as producing a particular symbol when, in fact, the exact form is not fixed (even when the font and size are fixed). Certain

features of  $\text{\TeX}$ 's mathematical typesetting can even be used to produce structures that can, in principle, grow to whatever size is required.

Such context-dependent variability is very important in mathematical typesetting, and this section discusses some aspects of it. With a few clearly noted exceptions, the commands covered in this section are available in standard  $\text{\LaTeX}$ .

A well-known, but not very exciting, example of such variability entails the mathematical operator symbols, such as  $\text{\sum}$  and  $\text{\prod}$ , which typically come in just two sizes: a smaller size that is used in running text and a larger size that is used in displayed formulas. Such symbols appear in Table 8.25 on page 536.

### 8.5.1 Ellipsis ...

Standard  $\text{\LaTeX}$  provides several types of mathematical ellipsis dots:  $\text{\ldots}$ ,  $\text{\cdots}$ , and so on. When using  $\text{amsmath}$ , however, such ellipsis dots within math mode should almost always be marked up using simply  $\text{\dots}$ .<sup>1</sup>

The vertical position (on the baseline or centered) of the ellipsis, together with the space around it, are both automatically selected according to what kind of symbol follows  $\text{\dots}$ . For example, if the next symbol is a plus sign, the dots will be centered; if it is a comma, they will be on the baseline. In all cases, three dots are used but the spacing varies. These defaults from the  $\text{amsmath}$  package can be changed in a class file when different conventions are in use.

A series $H_1, H_2, \dots, H_n$ , a sum $H_1 + H_2 + \dots + H_n$ , an orthogonal product $H_1 \times H_2 \times \dots \times H_n$ .	$\begin{array}{l} \text{\usepackage{amsmath}} \\ \\ \text{A series } \$H\_1, H\_2, \text{\dots}, H\_n\$, \text{ a sum} \\ \$H\_1 + H\_2 + \text{\dots} + H\_n\$, \text{ an orthogonal product} \\ \$H\_1 \times H\_2 \times \text{\dots} \times H\_n\$. \end{array}$
--	--

| 8-5-1

If the dots fall at the end of a mathematical formula, the next object will be something like  $\text{\end}$  or  $\text{\rangle}$  or  $\text{\$}$ , which does not give any information about how to place the dots. In such a case, you must help by using  $\text{\dotsc}$  for “dots with commas”,  $\text{\dotsb}$  for “dots with Binary operator/Relation symbols”,  $\text{\dotsm}$  for “multiplication dots”,  $\text{\dotsi}$  for “dots with integrals”, or even  $\text{\dotso}$  for “none of the above”. These commands should be used only in such special positions: otherwise you should just use  $\text{\dots}$ .

In this example, low dots are produced in the first instance and centered dots in the other cases, with the space around the dots being nicely adjusted.

A series $H_1, H_2, \dots$ , a sum $H_1 + H_2 + \dots$ , an orthogonal product $H_1 \times H_2 \times \dots$ , and an infinite integral: $\int_{H_1} \int_{H_2} \dots -\Gamma d\Theta$	$\begin{array}{l} \text{\usepackage{amsmath}} \\ \\ \text{A series } \$H\_1, H\_2, \text{\dots}\$, \text{ a sum} \\ \$H\_1 + H\_2 + \text{\dots}\$, \text{ an orthogonal product} \\ \$H\_1 \times H\_2 \times \text{\dots}\$, \text{ and an infinite} \\ \text{integral: } \backslash[\text{\int}_{H\_1} \text{\int}_{H\_2} \text{\dotsi} \backslash; \\ \quad \{-\Gamma\}\backslash, d\Theta \backslash] \end{array}$
---	---

| 8-5-2

<sup>1</sup>The commands  $\text{\dots}$  and  $\text{\ldots}$  can also be used in text mode, where both always produce a normal text ellipsis.

You can customize the symbols and spacing produced by the `\dots` command in various contexts by redefining the commands `\dotsc`, `\dotsb`, `\dotsm`, and `\dotsi`; this would normally be done in a class file. Thus, for example, you could decide to use only two dots in some cases.

### 8.5.2 Horizontal extensions

In principle, any mathematical accent command can be set up to produce the appropriate glyph from a range of widths whenever these are provided by the available fonts. However, in standard L<sup>A</sup>T<sub>E</sub>X there are only two such commands: `\widehat` and `\widetilde`.

This section describes a few commands that produce constructions similar to these extensible accents. They all produce compound symbols of mathematical class Ordinary (see Section 8.9.1 on page 524) and are illustrated in this example.

		<pre>\usepackage{amsmath} \begin{aligned} \widehat{\psi_\delta(t) E_t h} &amp;= \widetilde{\psi_\delta(t) E_t h} \\ \overline{\psi_\delta(t) E_t h} &amp;= \underline{\psi_\delta(t) E_t h} \\ \overbrace{\psi_\delta(t) E_t h} &amp;= \underbrace{\psi_\delta(t) E_t h} \quad \text{Do not change style} \end{aligned}</pre>
8-5-3		<pre>\overrightarrow{\psi_\delta(t) E_t h} = \overleftarrow{\psi_\delta(t) E_t h} \quad \text{Do not change style without amsmath}</pre>
		<pre>\overrightarrow{\psi_\delta(t) E_t h} = \overleftarrow{\psi_\delta(t) E_t h} \quad \text{Do need amsmath}</pre>
		<pre>\overleftrightarrow{\psi_\delta(t) E_t h} = \overleftrightarrow{\psi_\delta(t) E_t h} \quad \text{Do need amsmath}</pre>

```
\begin{aligned}
\widehat{\psi_\delta(t) E_t h} &= \widetilde{\psi_\delta(t) E_t h} \\
\overline{\psi_\delta(t) E_t h} &= \underline{\psi_\delta(t) E_t h} \\
\overbrace{\psi_\delta(t) E_t h} &= \underbrace{\psi_\delta(t) E_t h} \\
&\quad \& \text{\texttt{\& \& \text{Do not change style}}} \\
&\quad \& \text{\texttt{\& \overrightarrow{\psi_\delta(t) E_t h}} \quad \text{Do not change style without amsmath}} \\
&\quad \& \text{\texttt{\& \& \text{Do not change style}}} \\
&\quad \& \text{\texttt{\& \& \text{without \textsf{amsmath}}}} \\
&\quad \& \text{\texttt{\& \underrightarrow{\psi_\delta(t) E_t h}}} \\
&\quad \& \text{\texttt{\& \overleftarrow{\psi_\delta(t) E_t h}}} \\
&\quad \& \text{\texttt{\& \underleftarrow{\psi_\delta(t) E_t h}}} \\
&\quad \& \text{\texttt{\& \overleftarrow{\psi_\delta(t) E_t h}}} \\
&\quad \& \text{\texttt{\& \overrightarrow{\psi_\delta(t) E_t h}}} \\
&\quad \& \text{\texttt{\& \overleftarrow{\psi_\delta(t) E_t h}}} \\
&\quad \& \text{\texttt{\& \text{Do need \textsf{amsmath}}}} \\
\end{aligned}
```

Further details of the availability and properties of these commands are unfortunately somewhat complex but they are summarized in the example. Here, “change style” means that the symbol employed is affected by the mathematical style in use so that they will look right when used, for example, in fractions or subscripts/superscripts (see Section 8.7.1 on page 502). Those that do not change style are suitable for use only at the top level of displayed mathematics.

Another horizontally extensible feature of L<sup>A</sup>T<sub>E</sub>X is the bar in a radical sign; it is described at the end of the next subsection.

$( )$	$\langle \rangle$	$\{ \}$	$\  \ $	$\backslash lVert \backslash rVert$
$\langle \rangle$	$\backslash langle \backslash rangle$	$\{ \}$	$\  \ $	$\backslash lvert \backslash rvert$
$\langle \rangle$	$\backslash lgroup \backslash rgroup$	$[]$	$ $	
$\langle \rangle$	$\backslash lmoustache \backslash rmoustache$	$[]$	$\backslash vert$	
$\Downarrow$	$\backslash Downarrow$	$[]$	$\backslash arrowvert$	
$\Uparrow$	$\backslash Uparrow$	$[]$	$\backslash bracevert$	
$\Updownarrow$	$\backslash Updownarrow$	$[]$	$\backslash Arrowvert$	
$\downarrow$	$\backslash downarrow$	$/$	$\backslash  $	
$\uparrow$	$\backslash uparrow$	$\backslash backslash$	$\backslash Vert$	
$\Updownarrow$	$\backslash updownarrow$	.	$\sqrt{\phantom{x}}$	$\backslash sqrtsign$

Symbols in blue require either the `amsmath` package or, if additionally denoted with (*STY*), the `stmaryrd` package.

A period ( $.$ ) is not itself an extensible symbol but it can be used to produce an “invisible” delimiter.

The `\sqrtsign` symbol cannot be used with `\left`, `\right`, or `\middle`.

Synonyms:  $\left[ \right]$   $\left\{ \right\}$   $\left\langle \right\rangle$   $\left| \right|$   $\left\lvert \right\rvert$   $\left\lVert \right\rVert$   $\left\lvert\right\rvert$

Table 8.3: Vertically extensible symbols

### 8.5.3 Vertical extensions

There is a much larger range available with vertical extensions. All of the symbols depicted in Table 8.26 on page 537 are potentially extensible, as are a few others. The full list is given in Table 8.3. These symbols become extensible only in certain usages; they must all be based on a construction of the following form:<sup>1</sup>

`\left<ext-Open> <sub-formula> \right<ext-Close>`

<sup>1</sup>If L<sup>A</sup>T<sub>E</sub>X is using the eT<sub>E</sub>X program, then you can also use these extensible symbols with `\middle`.

Here  $\langle ext-Open \rangle$  and  $\langle ext-Close \rangle$  can be any of the symbols (except  $\sqrt$ ) listed in Table 8.3, or possibly others if additional packages are loaded. They must be symbols that have been set up to be extensible using the methods described in [109], which is part of every L<sup>A</sup>T<sub>E</sub>X distribution; thus, a symbol must be available to represent the absence of an actual glyph. This symbol, which is sometimes called the *null delimiter*, was chosen to be the period (.). The sizes of the actual glyphs used to typeset the extensible symbols are chosen to fit with the vertical size (height and depth) of the typeset *sub-formula* that lies in between them; the exact details of how this is done, and of the parameters that affect the process, can be found in Chapter 17 and Appendix G (Rule 19) of *The T<sub>E</sub>Xbook* [82]. One can also request specific sizes for such symbols (see Section 8.7.3 on page 504).

The radical sign  $\sqrt$  is even more amazing—it grows both vertically and horizontally to fit the size of its argument. In L<sup>A</sup>T<sub>E</sub>X it is typically accessed via the  $\sqrt$  command, which is discussed further in Section 8.7.4 on page 504.

8-5-4 
$$1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{x}}}}}$$

```
\[ \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{x}}}}}} ]
```

## 8.6 Words in mathematics

### 8.6.1 The $\text{\\text}$ command

Math font-changing commands such as  $\mathit{mathrm}$  are not intended for putting normal text inside mathematics; even for single words this task is often best carried out with the  $\text$  command, which is similar to the L<sup>A</sup>T<sub>E</sub>X command  $\text{mbox}$  but is much better, ensuring that the text is set using the correct font size. The font will be the text font in use outside the current mathematical material.

8-6-1 Also, if  $\Delta_{\max \text{ up}} = \Delta_{\min \text{ down}}$   
(for all ups and downs) then

$$\Delta_{\sum \text{ of ups}} = \Delta_{\sum \text{ of downs}} \quad (1)$$

```
\usepackage{amsmath}
\begin{gather}
\text{Also, if } \Delta_{\max \text{ up}} = \Delta_{\min \text{ down}} \notag \\
\text{(for all ups and downs) then} \notag \\
\Delta_{\sum \text{ of ups}} = \Delta_{\sum \text{ of downs}} \notag
\end{gather}
```

### 8.6.2 Operator and function names

The names of many well-known mathematical functions (such as  $\log$  and  $\sin$ ) and operators (such as  $\max$  and  $\lim$ ) are traditionally typeset as words (or abbreviations) in Roman type so as to visually distinguish them from shorter variable

arccos	\arccos	arcsin	\arcsin	arctan	\arctan
arg	\arg	cos	\cos	cosh	\cosh
cot	\cot	coth	\coth	csc	\csc
deg	\deg	det	\det^{(\ell)}	dim	\dim
exp	\exp	gcd	\gcd^{(\ell)}	hom	\hom
inf	\inf^{(\ell)}	inj lim	\injlim^{(\ell)}	ker	\ker
lg	\lg	lim	\lim^{(\ell)}	lim inf	\liminf^{(\ell)}
lim sup	\limsup^{(\ell)}	ln	\ln	log	\log
max	\max^{(\ell)}	min	\min^{(\ell)}	Pr	\Pr^{(\ell)}
proj lim	\projlim^{(\ell)}	sec	\sec	sin	\sin
sinh	\sinh	sup	\sup^{(\ell)}	tan	\tan
tanh	\tanh	$\lim_{\rightarrow}$	\varinjlim^{(\ell)}	<u>lim</u>	\varliminf^{(\ell)}
$\overline{\lim}$	\varlimsup^{(\ell)}	$\lim_{\leftarrow}$	\varprojlim^{(\ell)}		

Blue functions require the `amsmath` package.  $(\ell)$  indicates that the operator takes limits in displays.

Table 8.4: Predefined operators and functions

names that are set in “math italic”. The most common function names have predefined commands to produce the correct typographical treatment; see Table 8.4. Most functions are available in standard L<sup>A</sup>T<sub>E</sub>X; those listed in blue in the table require loading `amsmath`. The functions marked with  $(\ell)$  may “take limits” in display formulas (see Section 8.4.4).

```
\usepackage[fleqn]{amsmath}
\newcommand\abs[1]{\lvert#1\rvert}
\setlength\mathindent{0pt}
\begin{gather*}
\lim_{x \rightarrow 0} \frac{\sin^2(x)}{x^2} = 1 \\
\lim_{n \rightarrow \infty} |a_{n+1}| / |a_n| = 0 \\
\varliminf_{A/p \rightarrow \lambda(A)} (m_i^\lambda \cdot M)^* \leq \varprojlim_{A/p} A_p \leq 0
\end{gather*} 8-6-2
```

New functions of this type are needed frequently in mathematics, so the `amsmath` package provides a general mechanism for defining new “operator names”.

```
\DeclareMathOperator*[cmd]{text} \operatornamename*[text]
```

The `\DeclareMathOperator` defines *cmd* to produce *text* in the appropriate font for “textual operators”. If the new function being named is an operator that should, when used in displays, “take limits” (so that any subscripts and superscripts are placed in the “limits” positions, above and below, as with, for example,  $\lim$ ,  $\sup$ ,

or min), then use the starred form `\DeclareMathOperator*`. In addition to using the proper font, `\DeclareMathOperator` sets up good spacing on either side of the function name when necessary. For example, it gives  $A \text{ meas } B$  instead of  $A \text{meas} B$ . The *text* argument is processed using a “pseudo-text mode” in which

- The hyphen character `-` will print as a text hyphen (not as a minus sign); see `\supminus` in the next example.
- The asterisk character `*` will print as a raised text asterisk (not centered).
- Otherwise, the text is processed in math mode so that spaces are ignored and you can use subscripts, superscripts, and other elements.

The related command `\operatorname` (and its `*-form`) simply turns its argument into a function name, as in Example 8-2-11 on page 475. It is useful for “one-off” operators.

The next example shows how to provide the command `\meas` for the new function name “meas” (short for measure) and the operator functions `\esssup` and `\supminus`, both of which take limits.

**8-6-3**

$\ f\ _\infty = \text{ess sup}_{x \in R^n}  f(x) $ $\text{meas}_1 \{u \in R_+^1 : f^*(u) > \alpha\} =$ $\text{ess sup}_{x \in R^n} \text{meas}_i \{u \in R^n :  f(u)  \geq \alpha\}$ $(\forall \alpha \in \supminus^* R_{*+})$	<pre>\usepackage[fleqn]{amsmath} \DeclareMathOperator \meas {meas} \DeclareMathOperator*{\esssup}{ess \, sup} \DeclareMathOperator*{\supminus}{sup - minus*} \newcommand\abs[1]{\lvert#1\rvert} \newcommand\norm[1]{\lVert#1\rVert}  \begin{gather*} \ f\ _\infty = \text{esssup}_{x \in R^n} \abs{f(x)} \\ \text{meas}_1 \{u \in R_+^1 : f^*(u) &gt; \alpha\} = \\ \text{esssup}_{x \in R^n} \text{meas}_i \{u \in R^n :  f(u)  \geq \alpha\} \\ (\forall \alpha \in \supminus^* R_{*+}) \end{gather*}</pre>
--	---

Unfortunately, such declarations must appear in the preamble so it is not possible to change a declaration temporarily. In fact, `\DeclareMathOperator` works only for command names that have not been used previously, so it is not possible to overwrite an existing command directly. To do so, you must first remove the previous definition (in this case, of `\csc`) before redeclaring it; this removal is accomplished by using low-level TeX coding, as L<sup>A</sup>T<sub>E</sub>X provides no method for completing this task.

**8-6-4**

$\lim_{n \rightarrow \infty} \overline{Q}(u_n, u_n - u^\#) \geq \text{cosec}(Q'(u^\#))$	<pre>\usepackage{amsmath} % Low-level TeX needed here to cancel % the old definition of \csc: \let \csc \relax \DeclareMathOperator{\csc}{cosec} \newcommand\calQ{\mathcal{Q}} \varlimsup_{n \rightarrow \infty} \calQ(u_n, u_n - u^{\#}) \geq \csc(\calQ'(u^{\#}))</pre>
---	---

<i>Style</i>	<i>Superscript</i>	<i>Subscript</i>	<i>Numerator</i>	<i>Denominator</i>
$D$	$S$	$S'$	$T$	$T'$
$D'$	$S'$	$S'$	$T'$	$T'$
$T$	$S$	$S'$	$S$	$S'$
$T'$	$S'$	$S'$	$S'$	$S'$
$S, SS$	$SS$	$SS'$	$SS$	$SS'$
$S', SS'$	$SS'$	$SS'$	$SS'$	$SS'$

Table 8.5: Mathematical styles in sub-formulas

## 8.7 Fine-tuning the mathematical layout

Although L<sup>A</sup>T<sub>E</sub>X generally does a good job of laying out the elements of a formula, it is sometimes necessary to fine-tune the positioning. This section describes how to achieve some of the many detailed adjustments to the layout that are used to produce mathematical typography that is just a little bit better. Most of this section applies to all L<sup>A</sup>T<sub>E</sub>X mathematical material, but a few features are available only with the `amsmath` package; these will be clearly labeled.

### 8.7.1 Controlling the automatic sizing and spacing

Letters and mathematical symbols normally get smaller, and are more tightly spaced, when they appear in fractions, superscripts, or subscripts. In total, T<sub>E</sub>X has eight different styles in which it can lay out formulas:

$D, D'$	<code>\displaystyle</code>	Displayed on lines by themselves
$T, T'$	<code>\textstyle</code>	Embedded in text
$S, S'$	<code>\scriptstyle</code>	In superscripts or subscripts
$SS, SS'$	<code>\scriptscriptstyle</code>	In all higher-order superscripts or subscripts

The prime versions ( $D', T'$ , etc.) represent the so-called *cramped* styles, which are similar to the normal styles except that superscripts are not raised so much.

T<sub>E</sub>X uses only three type sizes for mathematics in these styles: text size (also used in `\displaystyle`), script size, and scriptscript size. The size of each part of a formula can be determined according to the following scheme.

<i>A symbol in style</i>	<i>Will be typeset in</i>	<i>And produces</i>
$D, D', T, T'$	text size	(text size)
$S, S'$	script size	(script size)
$SS, SS'$	scriptscript size	(scriptscript size)

In L<sup>A</sup>T<sub>E</sub>X, the top-level part of a formula set in running text (within a \$ pair or between `\(...\)`) is typeset using text style (style  $T$ ). A displayed formula

(e.g., one between  $\{ \dots \}$ ) will be typeset in display style (style D). The kind of style used in a sub-formula can then be determined from Table 8.5 on the facing page, where the last two columns describe the styles used in the numerator and the denominator of a fraction.

The various styles can be seen in this example:

	<code>\normalsize</code>	%% Style:
	<code>\[ b</code>	%% D
	<code>^0</code>	%% S
	<code>+</code>	%% D
	<code>\frac{(k + p)</code>	%% T
	<code>_{{j'}}</code>	%% S'
	<code>% \displaystyle</code>	
8-7-1	<code>\pm</code>	%% T [D]
	<code>\frac{(f + q)</code>	%% S [T]
	<code>_{{(pk)}}</code>	%% SS [S]
	<code>^y</code>	%% SS
	<code>_{{j'}}}}</code>	%% SS'
	<code>{(h + y)}</code>	%% S' [T']
	<code>^{{(pk)}}}</code>	%% T'
		%% S'
	<code>\]</code>	

You can change the layout of this example by explicitly specifying the style to be used in each part. For example, if you remove the comment character in front of `\displaystyle`, then some of the styles will change to those shown in brackets. The result looks like this:

$$8-7-2 \quad b^0 + \frac{(k + p)_{j'} \pm (f + q)^{(pk)}_{j'}}{(l + q)^{(pk)}}$$

Section 3.1.4 describes other ways to change the style of an individual symbol.

### 8.7.2 Sub-formulas

Whereas in text a pair of braces can simply indicate a group to which the effects of some declaration should be confined, within mathematics they do more than this. They delimit a sub-formula, which is always typeset as a separate entity that is added to the outer formula. As a side effect, sub-formulas are always typeset at their natural width and will not stretch or shrink horizontally when TeX tries to fit a formula in a paragraph line during line-breaking. As shown earlier, the sub-formula from a simple brace group is treated as if it was just a single symbol (of class Ordinary). An empty brace group, therefore, generates an invisible symbol that can affect the spacing. The exact details can be found in Chapters 17 and 18 and Appendix G of *The TeXbook* [82].

The contents of subscripts/superscripts and the arguments of many (but not all) commands, such as `\frac` and `\mathrel`, are also sub-formulas and get this same special treatment. Important examples of arguments that are not necessarily set as sub-formulas include those of `\bm` (see Section 8.8.2). If a group is needed only to limit the scope of a declaration (i.e., where a separately typeset sub-formula would be wrong), then `\begingroup` and `\endgroup` should be used. Note that specialized mathematical declarations such as style changes apply until the end of the current sub-formula, irrespective the presence of any other groups.

### 8.7.3 Big-g delimiters

To provide direct control of the sizes of extensible delimiters, L<sup>A</sup>T<sub>E</sub>X offers four commands: `\big`, `\Big`, `\bigg`, and `\Bigg`. These take a single parameter, which *must* be an extensible delimiter, and they produce ever-larger versions of the delimiter, from 1.2 to 3 times as big as the base size.

Three extra variants exist for each of the four commands, giving four sizes of Opening symbol (e.g., `\bigl`); four sizes of Relation symbol (e.g., `\Bigm`); and four sizes of Closing symbol (e.g., `\Biggr`).<sup>1</sup> All 16 of these commands can (and must) be used with any symbol that can come after either `\left`, `\right`, or (with eTEX) `\middle` (see Table 8.3 on page 498).

In standard L<sup>A</sup>T<sub>E</sub>X the sizes of these delimiters are fixed. With the `amsmath` package, however, the sizes adapt to the size of the surrounding material, according to the type size and mathematical style in use, as shown in the next example. The same is true when you load the `exscale` package (see Section 7.5.5), or when you use a font package that implements the `exscale` functionality as an option (e.g., most of the packages discussed in Sections 7.6 and 7.7).

$$\left( \mathbf{E}_y \int_0^{t_\varepsilon} L_{x,y^x(s)} \varphi(x) ds \right) \quad \text{\usepackage{amsmath}} \\ \left[ \biggl( \mathbf{E}_y \int_0^{t_\varepsilon} L_{x,y^x(s)} \varphi(x) ds \biggr) \right] \\ \left( \mathbf{E}_y \int_0^{t_\varepsilon} L_{x,y^x(s)} \varphi(x) ds \right) \quad \text{\Large} \\ \left[ \biggl( \mathbf{E}_y \int_0^{t_\varepsilon} L_{x,y^x(s)} \varphi(x) ds \biggr) \right]$$

#### 8.7.4 Radical movements

In standard L<sup>A</sup>T<sub>E</sub>X, the placement of the index on a radical sign is sometimes not good. With amsmath, the commands `\leftroot` and `\uproot` can be used within the optional argument of the `\sqrt` command to adjust the positioning of this index. Positive integer arguments to these commands move the root index to the left and up, respectively, while negative arguments move it right and down. These

<sup>1</sup>See Section 8.9.1 on page 524 for the various mathematical classes of symbols.

arguments are given in terms of math units (see Section 8.7.6), which are quite small, so these commands are useful for fine adjustments.

```
8-7-4 \usepackage{amsmath}
\[
\sqrt[k]{k} \quad \sqrt[3]{k} \quad \sqrt[4]{k}
\] \sqrt[\beta]{\beta} \qquad \sqrt[\beta]{\leftroot{2}\uproot{4}\beta} \qquad \sqrt[\beta]{\leftroot{1}\uproot{3}\beta}
```

### 8.7.5 Ghostbusters™

To get math spacing and alignment “just right”, it is often best to make creative use of some of primitive TEX’s unique and sophisticated typesetting abilities. These features are accessed by a collection of commands related to `\phantom` and `\smash`; and they can be used in both mathematical and other text.

For instance, the large alignment example (Example 8-2-9 on page 474) uses lots of phantoms to get the alignment just right. Each of these phantoms produces an invisible “white box” whose size (width and total height plus depth) is determined by typesetting the text in its argument and measuring its size.

Conversely, the command `\smash` typesets its contents (in an LR-box) but then ignores both their height and depth, behaving as if they were both zero. The standard TEX command `\hphantom` is a combination of these, producing the equivalent of `\smash{\phantom{a truly busted ghost!}}`: an invisible box with zero height and depth but the width of the phantom contents.

The `\vphantom` command makes the width of the phantom zero but preserves its total height plus depth. An example is the command `\mathstrut`, which is defined as “`\vphantom(`” so that it produces a zero-width box of height and depth equal to that of a parenthesis.

The `amsmath` package provides an optional argument for `\smash`, used as follows: `\smash[t]{...}` ignores the height of the box’s contents, but retains the depth, while `\smash[b]{...}` ignores the depth and keeps the height. Compare these four lines, in which only the handling of  $\sqrt{y}$  varies:

```
8-7-5 \usepackage{amsmath}
\sqrt{x} + \sqrt{y} + \sqrt{z}      $ \sqrt{x} + \sqrt{y} + \sqrt{z} \\ 
\sqrt{x} + \sqrt{y} + \sqrt{z}      $ \sqrt{x} + \sqrt{y} + \mathstrut \sqrt{z} \\ 
\sqrt{x} + \sqrt{y} + \sqrt{z}      $ \sqrt{x} + \sqrt{y} + \smash{y} \sqrt{z} \\ 
\sqrt{x} + \sqrt{y} + \sqrt{z}      $ \sqrt{x} + \sqrt{y} + \smash[b]{y} \sqrt{z}
```

To get the three radical signs looking pleasantly similar, it seems that the thing to do may be to give the  $y$  some extra height with a strut—but that only makes things worse! The best solution turns out to be to smash the bottom of the  $y$  (but not the whole of it!).

In the next example, the top of the large fraction in the second line appears correctly at its normal height, while neither this height nor the depth of the  $p$  in

the denominator on the first line affects the vertical space between the two lines. This, of course, would bring the two lines in this example confusingly close together. For this reason, another `\strut` was added. Nevertheless, more moderate use of smashing is often of benefit to such unbalanced displays.

```
\usepackage{amsmath}
\[
f_p(x) = \begin{cases} \frac{1}{p} & x = p \\ \frac{\frac{(1-x)}{2}}{\sqrt{1-p} \cos(x-p)} & x \neq p \end{cases} \quad \text{8-7-6}
\begin{array}{l}
\text{f\_p (x) =} \\
\text{\begin{cases}} \\
\text{\frac{1}{p} \quad x = p} \\
\text{\frac{\frac{(1-x)}{2}}{\sqrt{1-p} \cos(x-p)} \quad x \neq p} \\
\text{\end{cases}} \\
\text{\end{array}}
\]

```

Another collection of examples illustrates a very common application of smashing: using a partial `\smash` to give fine control over the height of surrounding delimiters. It also shows that smashing can lead to problems because the real height of the line needs to be known; this is restored by `\vphantom`. In the following code, `\Hmjdf` is the compound symbol defined by

```
\newcommand{\Hmjdf}{\widetilde{\mathcal{H}^2_{MJD}(\chi)}}
```

To show the resulting vertical space we added some rules:

<i>Appearance</i>	<i>Code</i>	<i>Comment</i>
$\widetilde{(\mathcal{H}^2_{MJD}(\chi))}$	<code>\left( \right. \left. \vphantom{\mathcal{H}^2_{MJD}(\chi)} \right)</code>	<i>Outer brackets too large</i>
$(\mathcal{H}^2_{MJD}(\chi))$	<code>\left( \right. \left. \smash{\mathcal{H}^2_{MJD}(\chi)} \right)</code>	<i>Outer brackets too small and rules too close</i>
$(\mathcal{H}^2_{MJD}(\chi))$	<code>\left( \right. \left. \smash[t]{\mathcal{H}^2_{MJD}(\chi)} \right) \vphantom{\mathcal{H}^2_{MJD}(\chi)}</code>	<i>Just right!</i>
$(\mathcal{H}^2_{MJD}(\chi))$	<code>\left( \right. \left. \smash[t]{\mathcal{H}^2_{MJD}(\chi)} \right)</code>	<i>Both vphantom and partial smash are needed</i>

A word of warning: in a few places, deficiencies in the very low-level TEX processing may cause errors in the fine details of typesetting. These possibilities are of particular concern in mathematical layouts where (1) a sub-formula (such as the numerator/denominator of a fraction or subscripts/superscripts) consists of exactly one LR-box, or a similarly constructed mathematical box, and also (2) that

*Smashes being ignored by TEX*

box does not have its natural size, as with the more complex forms of `\makebox`, `\smash`, and some phantoms. As an example look at the following:

```
8-7-7 \sqrt{\frac{a+b}{x_j}} \sqrt{\frac{a+b}{x_j}} \sqrt{\frac{a+b}{x_j}} \sqrt{\frac{a+b}{x_j+b}}
          \sqrt{\frac{a+b}{\frac{x_j}{\smash{x_j}}}} \sqrt{\frac{a+b}{\frac{x_j}{\smash{\{x_j\}}}}} \sqrt{\frac{a+b}{\frac{x_j}{\smash{\{\}x_j}}}} \sqrt{\frac{a+b}{\frac{x_j}{\smash{x_j+b}}}}
```

To shorten the depth of the radical, a `\smash` was added in the second radical, but without any effect. With an empty brace group (third radical), it suddenly worked. On the other hand, no workaround was needed for the forth radical.<sup>1</sup> For the same reason the `\strut` or an empty brace group was actually necessary in Example 8-7-6 on the facing page to see any effects from the `\smash` commands there. In summary, whenever you find that a `\smash` does not work, try adding an empty math sub-formula (from `{}`) before the lonely box, to keep it from being mistreated.

### 8.7.6 Horizontal spaces

Even finer, and more difficult, tuning requires the explicit spacing commands shown in Table 8.6 on the next page. Both the full and short forms of these commands are robust, and they can also be used outside math mode in normal text. They are related to the thin, medium, and thick spaces available on the machines used to typeset mathematics in the mid-20th century.

The amounts of space added by these `\dotspace` commands are, in fact, defined by the current values of the three parameters `\thinmuskip`, `\medmuskip`, and `\thickmuskip`; the table lists their default values with `amsmath`. These very low-level TeX parameters require values in “mu” (*math units*). They must therefore be set only via low-level TeX assignments (as shown in Example 8-9-2 on page 525) and not by `\setlength` or similar commands. Moreover, in normal circumstances their values should not be modified because they are used internally by TeX’s mathematical typesetting (see Table 8.7 on page 525).

 *Do not change the parameter values*

One math unit (1mu) is 1/18 of an em in the current mathematical font size (see also Table A.1 on page 855). Thus, the absolute value of a math unit varies with the mathematical style, giving consistent spacing whatever the style.

These *math units* can be used more generally to achieve even better control over space within mathematics. This is done via the `amsmath` command `\mspace`, which is like `\hspace` except that it can be used only within mathematics and its length argument must be given in math units (e.g., `\mspace{0.5mu}`). Thus, to get a negative `\quad` within a mathematical formula, you could write `\mspace{-18.0mu}`; this will, for example, normally give about half the space

<sup>1</sup>Technically this is due to the denominator being wider than the nominator in this case, so that it was not reboxed by TeX.

<i>Positive Spaces</i>			<i>Negative Spaces</i>			<i>Amount</i>
<i>Short</i>	<i>Space</i>	<i>Full</i>	<i>Short</i>	<i>Space</i>	<i>Full</i>	
\,	$\Rightarrow \Leftarrow$	\thinspace	\!	$\Rightarrow \Leftarrow$	\negthinspace	3mu
\:	$\Rightarrow \Leftarrow$	\medspace		$\Rightarrow \Leftarrow$	\negmedspace	4mu plus 2mu minus 4mu
\;	$\Rightarrow \Leftarrow$	\thickspace		$\Rightarrow \Leftarrow$	\negthickspace	5mu plus 5mu
	$\Rightarrow \Leftarrow$	\enskip				0.5em
	$\Rightarrow \Leftarrow$	\quad				1em
	$\Rightarrow \Leftarrow$	\quad\quad				2em

*Note: The “Amount” column is discussed in the text.*

Table 8.6: Mathematical spacing commands

in a double subscript size as it does in the basic mathematical size. In contrast, `\hspace{-1em}` will produce the same amount of space whatever the mathematical font size (but `\text{\hspace{-1em}}` will produce variable-sized space).

## 8.8 Fonts in formulas

For most symbols in a formula, the font used for a glyph cannot be changed by a font declaration as it can be in text. Indeed, there is no concept of, for example, an italic plus sign or a small caps less than sign.

One exception involves the letters of the Latin alphabet, whose appearance can be altered by the use of math alphabet identifier commands such as `\mathcal{}`. The commands provided by standard L<sup>A</sup>T<sub>E</sub>X for this purpose are discussed in Section 7.4; this section introduces a few more. Another exception relates to the use of bold versions of arbitrary symbols to produce distinct symbols with new meanings. This potentially doubles the number of symbols available, as boldness can be a recognizable attribute of a glyph for nearly every shape: depending on the font family, even “<” is noticeably different from “<”. Although there is a `\mathbf{}` command, the concept of a math alphabet identifier cannot be extended to cover bold symbols; a better solution is discussed in Section 8.8.2.

To change the overall appearance of the mathematics in a document, the best approach is to replace all the fonts used to typeset formulas. This is usually done in the preamble of a document by loading a (set of) suitable packages, such as those discussed in Sections 7.6 and 7.7.

At the end of this section we showcase the effects of such extensive changes, made with but a few keystrokes, on a sample page of mathematics. Section 8.8.3 contains the same material typeset with both Computer Modern Math fonts (the default in L<sup>A</sup>T<sub>E</sub>X) and 15 other font families for text and mathematics. All of the fonts used are readily available and about half of them are provided free of charge.

### 8.8.1 Additional math font commands

By loading the `amsfonts` (or `amssymb`) package, the Euler Fraktur alphabet (`\mathfrak{A}`) and a Blackboard Bold alphabet (`\mathbb{A}`) become available.

**8-8-1**  $\forall n \in \mathbb{N} : \mathfrak{M}_n \leq \mathfrak{A}$  `\usepackage{amsfonts}`  
`$ \forall n \in \mathbb{N} : \mathfrak{M}_n \leq \mathfrak{A}`

As an example of small-scale changes to the mathematical typesetting, those who prefer a visually distinct Blackboard Bold alphabet can load one from the Math Pazo fonts. See Section 7.6.3 for more information on the Math Pazo fonts and Section 7.4.1 for details on `\DeclareMathAlphabet`. In this example we first load the `amsfonts` package and then overwrite its definition of `\mathbb{A}`.

**8-8-2**  $\{n, m \in \mathbb{N} \mid \mathfrak{N}_{n,m}\}$  `\usepackage{amsfonts}`  
`\DeclareMathAlphabet{\mathbb{U}}{fplmbb}{m}{n}`  
`$ \{n, m \in \mathbb{N} \mid \mathfrak{N}_{n,m}\}`

This example shows how to include arbitrary alphabets from your  $\text{\LaTeX}$  distribution as math alphabets, with the crucial part being the arguments of `\DeclareMathAlphabet`. Although getting these right may appear to be a tricky matter, it is not so difficult once you know where to look. Fonts suitable for inclusion need to have an `.fd` file; that is, given a font family name in the Berry naming convention (see Section 7.10.2), there should be a file `<enc><name>.fd`. For example,

*the (commercial) Lucida Handwriting font*

has the family name `hlcw`. It is available in several encodings, including T1, so one possible file to look at is `t1hlcw.fd`. In that file you will find the remaining arguments for the declaration. The font is available only in series `m` and shape `it`. All other font shapes contain substitutions (see Section 7.10.6 for details on the file format for `.fd` files). Putting all this together enables us to provide a `\mathscr{A}` command. Another possibility is to use this alphabet as a replacement for the standard `\mathcal{A}` command.

**8-8-3**  $A_B \neq \mathscr{A}_B \neq \mathcal{A}_B$  `\DeclareMathAlphabet{\mathscr{T1}{hlcw}{m}{it}}`  
`$ A_B \neq \mathscr{A}_B \neq \mathcal{A}_B $`

Of course, the presence of the file `t1hlcw.fd` (and other support files) on your system does not mean that the previous example will run there. To achieve this goal, you must also install the corresponding commercial font. Most modern  $\text{\LaTeX}$  installations contain such support files for various commercial font sets, so that you can use these fonts the moment you have bought them and added them to your system. In this case you would need a file called `hlcritw8a.pfb`.

In truth, you probably do not need to buy any fonts, because the freely available fonts already include a huge choice. The `nfssfont.tex` program can provide

valuable help in choosing a font, by producing samples and character tables for the fonts available to your installation (see Section 7.5.7).

### 8.8.2 `bm`—Making bold

For bold Latin letters only, you can use the command `\mathbf`; for everything else, there is the `bm` package. Although `amsmath` provides `\boldsymbol` and `\pmb`, the rules about when to use which command, and many of the restrictions on when they work, can now be avoided: just load the `bm` package and use `\bm` to make any formula as bold and beautiful as the available fonts allow.

The example below shows many ways to use the `\bm` and `\mathbf` commands and a strategy for defining shorthand names for frequently occurring bold symbols, using both standard L<sup>A</sup>T<sub>E</sub>X's `\newcommand` and `\bmdefine`, which is provided by `bm`. Note that `\mathbf{xy}` is not identical to `\bm{xy}`: the former produces bold Roman "xy" and the latter produces "xy" (i.e., bold math italic).

```
\usepackage{amsmath, amssymb, bm}
\newcommand\bfB{\mathbf{B}} \newcommand\bfx{\mathbf{x}}
\bmdefine\bp{\pi} \bmdefine\binf{\infty}
\section{The bold equivalence}
$ \sum_{j < B} \prod_\lambda : \bm{(\sum_{x_j} \prod_\lambda)} \\
\begin{gather}
B_\infty + \pi B_1 \sim \bfB_\infty \bm{+} \bfB_1 \sim B_\infty + \pi B_1 \\
\bm{+} \bm{\sim} B_\infty + \pi B_1 \\
B_\infty + \bm{\partial} B_1 \bm{\partial} \infty \bm{+} \bm{\partial} B_1 \\
(\bfB, \bfx) : \frac{\partial \bfB}{\partial x} \bm{=} \frac{\partial \bfx}{\partial x} \\
\bm{+} \bm{\approx} \bm{\approx} \bm{+} \bm{\approx} \bm{\approx} \\
\end{gather}
\end{gather}
```

## 1 The bold equivalence $\sum_{j < B} \prod_\lambda : \sum_{x_j} \prod_\lambda$

$$B_\infty + \pi B_1 \sim \bfB_\infty + \bfB_1 \sim B_\infty + \pi B_1 \quad (1)$$

$$B_\infty + \pi B_1 \in \left\{ (\bfB, \bfx) : \frac{\partial \bfB}{\partial x} \bm{\approx} \bf1 \right\} \quad (2)$$

In the above example `bm` tries its best to fulfill the requests for bold versions of individual symbols and letters, but if you look closely you will see that the results are not always optimal. For example,  $\sum$ ,  $\prod$ , and  $\approx$  are all made bold by use of a technique known as *poor man's bold*, in which the symbol is overprinted three times with slight offsets. Also, the  $\{$  is not made bold in any way. Such deficiencies are unavoidable because for some symbols there is simply no bold variant available when using the Computer Modern math fonts.

The situation changes when the `txfonts` are loaded by changing the first line of the previous example to `\usepackage{amsmath, amssymb, txfonts, bm}`. This

family of fonts contains bold variants for *all* symbols from standard L<sup>A</sup>T<sub>E</sub>X and `amssymb`. It produces the following output:

## 1 The bold equivalence $\sum_{j < B} \prod_{\lambda} : \Sigma_{x_j} \prod_{\lambda}$

8-8-5

$$B_\infty + \pi B_1 \sim \mathbf{B}_\infty + \boldsymbol{\pi} \mathbf{B}_1 \sim \mathbf{B}_\infty + \boldsymbol{\pi} \mathbf{B}_1 \quad (1)$$

$$B_\infty + \pi B_1 \in \left\{ (\mathbf{B}, \mathbf{x}) : \frac{\partial \mathbf{B}}{\partial \mathbf{x}} \not\leq \mathbf{1} \right\} \quad (2)$$

What are the precise rules used by `\bm` to produce bold forms of the symbols in its argument? In a nutshell, it makes use of the fact that L<sup>A</sup>T<sub>E</sub>X includes a **bold** math version (accessible via `\boldmath`) for typesetting a whole formula in bold (provided suitable bold fonts are available and set up). For each symbol, the `\bm` command looks at this math version to see what would be done in that version. If the font selected for the symbol is different from the one selected in the normal math version, it then typesets the symbol in this bold font, obtaining a perfect result (assuming that the bold math version was set up properly). If the fonts in both versions are identical, it assumes that there is no bold variant available and applies poor man's bold (see above).

 Load the `bm` package after packages that change the existing math font 'set-up'

With delimiters, such as `\biggl\lbrace` in the example, the situation is even more complex: a delimiter in T<sub>E</sub>X is typically typeset by a glyph chosen to match a requested height from a sequence of different sizes (see Section 8.5.3 on page 498). Moreover, these glyphs can live in different fonts and a particular size may or may not have bold variants, making it impossible for `\bm` to reliably work out whether it needs to apply poor man's bold. It therefore essentially typesets the delimiter using whatever fonts the bold math version offers. With the Computer Modern ath fonts, only the smallest delimiter size is available in bold; all other sizes come from fonts that have no bold variants.

8-8-6

```
\usepackage{bm}
$\bm{\Biggl\{\Biggl\{\Bigl\{\mathcal{Q}\Bigr\}\Bigl\}\Biggr\}\Biggr\}}
```

```
\usepackage{bm}
$ \bm{ \Biggl\{ \Biggl\{ \Bigl\{ \mathcal{Q} \Bigr\} \Bigr\} \Biggr\} }
```

This situation can be improved by use of the `txfonts` (as in Example 8-8-5) or use of another font set with full bold variants, such as the `pxfonts` shown here:

8-8-7

```
\usepackage{pxfonts,bm}
$\bm{\Biggl\{\Biggl\{\Bigl\{\mathcal{Q}\Bigr\}\Bigl\}\Biggr\}\Biggr\}}
```

```
\usepackage{pxfonts,bm}
$ \bm{ \Biggl\{ \Biggl\{ \Bigl\{ \mathcal{Q} \Bigr\} \Bigr\} \Biggr\} }
```

Normally, `\bm` requires that if a command that itself takes arguments is within its argument, then that command must be fully included (i.e., both the command and its arguments must appear) in the argument of `\bm`; as a result, all parts of the

typeset material will be in bold. If you really need the output of a command with arguments to be only partially bold, then you have to work harder. You should place the symbol(s) that should not be bold in an `\mbox` and explicitly reset the math version within the box contents using `\unboldmath`. TeX considers an `\mbox` to be a symbol of class Ordinary (see Section 8.9.1); hence, to get the spacing right, you may have to surround it by a `\mathbin`, `\mathrel`, or `\mathop`.

```
\usepackage{amsmath,bm}
$ \bm{\sqrt[2]{x \times \alpha}} $ but  $\sqrt[2]{x \times \alpha}$  8-8-8
or the similar  $\sqrt{x \times \alpha}$ 
$ \bm{\sqrt[2]{x \mathbin{\mbox{\unboldmath$\times$}} \alpha}} $ or the similar
$ \bm{\sqrt{\bm{x} \times \bm{\alpha}}} $
```

Fortunately, such gymnastics are seldom needed. In most cases involving commands with arguments, only parts of the arguments need to be made bold, which can be achieved by using `\bm` inside those arguments. As with `\sqrt{sign}` in the example above, for the common case of bold accents `\bm` is specially programmed to allow the accent's argument to be outside its own argument. However, if you need such accents regularly, it is wise to define your own abbreviation using `\bmdefine`, as in the next example.

*Speeding up the processing* Although `\bmdefine\bp{i}{\pi}` appears to be simply shorthand for `\newcommand\bp{i}{\bm{\pi}}`, in fact almost the opposite is true: `\bm` defines a new hidden temporary command using `\bmdefine` and then immediately uses this temporary command to produce the bold symbol. In other words, `\bmdefine` does all the hard work! If you frequently use, for example, something that is defined via `\bm{\alpha}`, then a new `\bmdefine` is executed at every use. If you set things up by doing `\bmdefine\balph{a}{\alpha}`, then `\bmdefine` does its time-consuming work only once, however many times `\balph{a}` is used.

```
\hat{a} \neq \hat{\hat{a}} \neq \hat{\hat{a}} = \hat{a} \neq \hat{\hat{a}} 8-8-9
$ \hat{a} \neq \hat{\hat{a}} \neq \hat{\hat{a}} = \hat{a} \neq \hat{\hat{a}} $
```

This example also shows that the variable-width accents (e.g., `\widehat`) share a deficiency with the delimiters: in the Computer Modern math set-up they come from a font for which no bold variant is available.

*Dealing with strange errors* The `bm` package tries very hard to produce the correct spacing between symbols (both inside and outside the argument of `\bm`). For this effort to work, `\bm` has to “investigate” the definitions of the commands in its argument to determine the correct mathematical class to which each of the resulting symbols belongs (see Section 8.9.1 on page 524). It is possible that some strange constructions could confuse this investigation. If this happens then L<sup>A</sup>T<sub>E</sub>X will almost certainly stop with a strange error. Ideally, this problem should not arise with constructs from standard L<sup>A</sup>T<sub>E</sub>X or the A<sub>M</sub>S-L<sup>A</sup>T<sub>E</sub>X distributions, but proper parsing in TeX is extremely difficult and the odd overlooked case might still be present. For instance, the author got trapped when writing this section by the fact that `\bm` was trying

to process the argument of `\hspace` instead of producing the desired space (this problem is fixed in version 1.1a).

If some command does produce an error when used inside `\bm`, you can always surround it *and all its arguments* with an extra level of braces—for example, writing `\bm{..{\cmd..}..}` rather than simply `\bm{..\cmd..}`. The `\bm` command will not attempt to parse material surrounded by braces but will use the `\boldmath` version to typeset the whole of the formula within the braces. The resulting bold sub-formula is then inserted as if it were a “symbol” of class Ordinary. Thus, to obtain the right spacing around it, you may have to explicitly set its class; for instance, for a relation you would use `\bm{..\mathrel{\cmd..}}` (see Section 8.9.1 on page 524).

### 8.8.3 A collection of math font set-ups

In this section we show a sample text typeset with different font set-ups for math and text. Figure 8.1 shows the sample text typeset in Computer Modern text and math fonts—the default font set-up in L<sup>A</sup>T<sub>E</sub>X. Figures 8.2 to 8.16 on pages 514–523 (with blue captions to visually separate caption and sample) have also been generated by typesetting this sample text, each time loading different support packages for text and math fonts. These packages do all the work required to modify L<sup>A</sup>T<sub>E</sub>X’s internal tables. For other set-ups and additional information see [24].

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_Q f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_G} \partial(\tilde{X}_\gamma)$ ; and also on display:

$$\begin{aligned} \iiint_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\stackrel{8-8-10}{\approx} \bigcup_{Q \in \bar{Q}} \left[ f^* \left( \frac{\int Q(t)}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\vartheta} \end{aligned} \quad (1)$$

For  $x$  in the open interval  $] -1, 1 [$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.1: Sample page typeset with Computer Modern fonts

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_Q f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_C} \partial(\tilde{X}_\gamma)$ ; and also on display:

$$\begin{aligned} \iiint_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\approx \bigcup_{Q \in \bar{Q}} \left[ f^* \left( \frac{\int Q(t)}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\vartheta} \end{aligned} \quad \text{8-8-1}$$

(1)

For  $x$  in the open interval  $] -1, 1 [$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.2: Sample page typeset with Concrete fonts

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_Q f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_C} \partial(\tilde{X}_\gamma)$ ; and also on display:

$$\begin{aligned} \iiint_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\approx \bigcup_{Q \in \bar{Q}} \left[ f^* \left( \frac{\int Q(t)}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\vartheta} \end{aligned} \quad \text{8-8-1}$$

(1)

For  $x$  in the open interval  $] -1, 1 [$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.3: Sample page typeset with Concrete and Euler fonts

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\int\int\int_{\mathcal{Q}} f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_{\tilde{C}}} \partial(\tilde{X}_{\gamma})$ ; and also on display:

8-8-13

$$\begin{aligned} \iiint_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\lesssim \bigcup_{\mathbb{Q} \in \bar{\mathbb{Q}}} \left[ f^* \left( \frac{\int \mathbb{Q}(t)}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\theta} \end{aligned} \quad (1)$$

For  $x$  in the open interval  $(-1, 1]$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.4: Sample page typeset with Fourier fonts

The Concrete Roman text fonts were designed by Donald Knuth, matching math fonts were designed by Ulrik Vieth; see Section 7.7.2. They are shown in Figure 8.2, which was produced by adding `\usepackage{boldsans}{ccffonts}` to the preamble of the sample document. Note that Concrete fonts have no boldface, so that the  $\partial Q$  subscript on the integral comes out in poor man's bold.

Figure 8.3 combines Concrete Roman with Euler Math (designed by Hermann Zapf). This combination was produced with

```
\usepackage{ccffonts} \usepackage[euler-digits]{eulervm}
```

and shows no deficiencies with bold symbols in math; see also Section 7.7.10. You will probably want to design different headings, as the default (Computer Modern boldface extended) does not blend very well with Concrete Roman.

In Figure 8.4 we see Utopia combined with Fourier Math fonts (designed by Michel Bovani). This combination has been discussed in Section 7.7.7 and was produced by adding `\usepackage{fourier}` to the preamble. Again, the boldface subscript shows deficiencies, but these are expected to be addressed in a future release of the fonts.

The METAFONT versions of Concrete, both Roman and Math, are freely available. Scalable outlines can be purchased from MicroPress.<sup>1</sup> The Fourier set-up is freely available in Type 1 format.

<sup>1</sup><http://www.micropress-inc.com>

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\mathop{\iiint}\limits_Q f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_{\tilde{C}}} \partial(\tilde{X}_\gamma)$ ; and also on display:

$$\begin{aligned} \mathop{\iiint}\limits_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\gtrsim \bigcup_{Q \in \bar{Q}} \left[ f^* \left( \frac{\int Q(t)}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\vartheta} \end{aligned} \quad (1) \quad \boxed{8-8-14}$$

For  $x$  in the open interval  $] -1, 1 [$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.5: Sample page typeset with Times and Symbol

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\mathop{\iiint}\limits_Q f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_{\tilde{C}}} \partial(\tilde{X}_\gamma)$ ; and also on display:

$$\begin{aligned} \mathop{\iiint}\limits_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\gtrsim \bigcup_{Q \in \bar{Q}} \left[ f^* \left( \frac{\int Q(t)}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\vartheta} \end{aligned} \quad (1) \quad \boxed{8-8-15}$$

For  $x$  in the open interval  $] -1, 1 [$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.6: Sample page typeset with Times and TX fonts

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_Q f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_C} \partial(\tilde{X}_\gamma)$ ; and also on display:

8-8-16

$$\begin{aligned} \iiint_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\gtrapprox \bigcup_{Q \in \bar{Q}} \left[ f^* \left( \frac{\int Q(t)}{\sqrt{1 - t^2}} \right) \right]_{t=\alpha}^{t=\delta} \end{aligned} \quad (1)$$

For  $x$  in the open interval  $] -1, 1 [$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1 - x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.7: Sample page typeset with Times and TM Math fonts

This page spread shows three math font set-ups for use with Times Roman as a body font. With Times Roman being one of the predominant fonts in use today, several solutions have been developed to provide support for it.

Figure 8.5 shows a free solution devised by Alan Jeffrey and others (discussed in Section 7.6.2), which was produced by adding `\usepackage{mathptmx}` to the preamble. It deploys Adobe's Symbol font for most mathematical symbols and due to a missing set of bold symbols for math, shows the typical deficiencies in this respect. In contrast to other font solutions it does not offer its own shapes for the extended AMS symbol set but uses the standard Computer Modern shapes.

Figure 8.6 also shows a freely available implementation deploying the TX fonts (designed by Young Ryu). It offers the full range of mathematical symbols including boldface variants, but uses exceptionally tight spacing so that sometimes symbols in formulas touch each other; see Section 7.7.5 for details. It can be activated by adding `\usepackage{txfonts}` in the preamble.

In Figure 8.7 we see the commercially available TM Math solution by MicroPress,<sup>1</sup> which uses considerably wider spacing in formulas. It comprises bold symbols and offers its own shapes for the AMS extended symbol set. It can be activated through `\usepackage{tmmath, tmams}` in the preamble.

Other commercial Math fonts in Type 1 format for use with Times Roman are MathTime and MathTime Professional (designed by Michael Spivak), available through Y&Y<sup>2</sup> and PC<sub>T</sub>E<sub>X</sub>,<sup>3</sup> respectively.

<sup>1</sup><http://www.micropress-inc.com>    <sup>2</sup><http://www.YandY.com>    <sup>3</sup><http://www.pctex.com>

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_Q f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_{\bar{C}}} \partial(\tilde{X}_{\gamma})$ ; and also on display:

$$\begin{aligned} \iiint_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\gtrapprox \bigcup_{Q \in \bar{Q}} \left[ f^* \left( \frac{\int Q(t)}{\sqrt{1 - t^2}} \right) \right]_{t=\alpha}^{t=\vartheta} \end{aligned} \quad \boxed{8-8-17}$$

(1)

For  $x$  in the open interval  $] -1, 1 [$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1 - x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.8: Sample page typeset with Palatino and Math Pazo

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_Q f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_{\bar{C}}} \partial(\tilde{X}_{\gamma})$ ; and also on display:

$$\begin{aligned} \iiint_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\gtrapprox \bigcup_{Q \in \bar{Q}} \left[ f^* \left( \frac{\int Q(t)}{\sqrt{1 - t^2}} \right) \right]_{t=\alpha}^{t=\vartheta} \end{aligned} \quad \boxed{8-8-18}$$

(1)

For  $x$  in the open interval  $] -1, 1 [$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1 - x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.9: Sample page typeset with Palatino and PX fonts

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_Q f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_C} \partial(\tilde{X}_\gamma)$ ; and also on display:

$$\begin{aligned} \iiint_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\stackrel{8-8-19}{\approx} \biguplus_{Q \in \mathbb{Q}} \left[ f^* \left( \frac{\int Q(t)}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\delta} \end{aligned} \quad (1)$$

For  $x$  in the open interval  $] -1, 1 [$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.10: Sample page typeset with Palatino and PA Math fonts

The typeface Palatino was designed by Hermann Zapf for the Stempel foundry in 1948 based on lettering from the Italian Renaissance. Since then it has become one of the most widely used typefaces, and probably the most popular Old Style revival in existence. A number of math font set-ups are available for use with Palatino as the text font.

Figure 8.8 shows the freely available Math Pazo fonts (designed by Diego Puga), which can be activated with `\usepackage{mathpazo}`. It offers boldface symbols and a matching blackboard bold alphabet, but does not contain specially designed shapes for the AMS symbol set; see also Section 7.6.3.

In contrast, the free PX fonts (designed by Young Ryu) comprise the complete symbol set. They are shown in Figure 8.9. Just like the TX fonts, they are very tightly spaced; see Section 7.7.6 for details. This set-up can be activated with `\usepackage{pxfonts}`.

Figure 8.10 shows the commercial solution offered by MicroPress.<sup>1</sup> It provides a similar range of symbols as the Math Pazo solution with roughly the same running length, though with noticeably different shapes. This set-up can be activated with `\usepackage{pamath}`.

<sup>1</sup><http://www.micropress-inc.com>

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_Q f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_C} \partial(\widetilde{X}_\gamma)$ ; and also on display:

$$\begin{aligned} \iiint_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\lesssim \bigcup_{Q \in \bar{Q}} \left[ f^* \left( \frac{\int Q(t)}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\delta} \end{aligned} \quad [8-8-20] \quad (1)$$

For  $x$  in the open interval  $] -1, 1[$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.11: Sample page typeset with Baskerville fonts

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_Q f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_C} \partial(\widetilde{X}_\gamma)$ ; and also on display:

$$\begin{aligned} \iiint_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\lesssim \bigcup_{Q \in \bar{Q}} \left[ f^* \left( \frac{\int Q(t)}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\delta} \end{aligned} \quad [8-8-21] \quad (1)$$

For  $x$  in the open interval  $] -1, 1[$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.12: Sample page typeset with Charter fonts

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_Q f(x, y, z) dx dy dz$  and  $\prod_{y \in \Gamma_C} \partial(\tilde{X}_y)$ ; and also on display:

$$\begin{aligned} \iiint_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\stackrel{8.8.22}{\lesssim} \bigcup_{Q \in \bar{Q}} \left[ f^* \left( \frac{|Q(t)|}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=9} \end{aligned} \quad (1)$$

For  $x$  in the open interval  $] -1, 1 [$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.13: Sample page typeset with Lucida Bright

Figure 8.11 deploys the Baskerville typeface as a text font. This “transitional” typeface was originally designed by John Baskerville (1706–1775) and can be obtained from many font vendors. The math fonts are BA Math from MicroPress<sup>1</sup>—their distribution also contains a variant of the Baskerville text fonts used here. The BA Math fonts include bold weights but do not contain shapes for the AMS symbol set. Note that although the individual symbols do not look very large, the display formulas take more vertical space than in other examples. The font set-up is activated with `\usepackage{ba}`.

Figure 8.12 shows the use of the commercial CH Math fonts (also from MicroPress<sup>1</sup>). Their distribution has been designed to work with the freely available Charter fonts; see Section 7.6.1. The CH Math fonts comprise the full set of mathematical symbols including the AMS additions and are activated by adding the preamble line `\usepackage{chmath, chams}`.

The Lucida Bright and Lucida New Math fonts are displayed in Figure 8.13. This set of commercial text and math fonts has been designed by Charles Bigelow and Kris Holmes and can be obtained from Y&Y.<sup>2</sup> The font set-up covers all standard mathematical symbols including AMS additions and is activated by loading the `lucidabr` package. As you will notice, the formulas run very wide, which enhances legibility at the cost of space. The body font in this book is Lucida Bright. However, for the examples, we usually used Computer Modern to make them come out as in standard L<sup>A</sup>T<sub>E</sub>X.

<sup>1</sup><http://www.micropress-inc.com>    <sup>2</sup><http://www.YandY.com>

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_Q f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_{\tilde{c}}} \partial(\tilde{X}_{\gamma})$ ; and also on display:

$$\begin{aligned} \iiint_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\lesssim \biguplus_{Q \in \bar{Q}} \left[ f^* \left( \frac{\int Q(t) dt}{\sqrt{1 - t^2}} \right) \right]_{t=\alpha}^{t=\vartheta} \end{aligned} \quad [8-8-23]$$

(1)

For  $x$  in the open interval  $] -1, 1 [$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.14: Sample page typeset with CM Bright fonts

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_Q f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_{\tilde{c}}} \partial(\tilde{X}_{\gamma})$ ; and also on display:

$$\begin{aligned} \iiint_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\approx \biguplus_{Q \in \bar{Q}} \left[ f^* \left( \frac{\int Q(t) dt}{\sqrt{1 - t^2}} \right) \right]_{t=\alpha}^{t=\vartheta} \end{aligned} \quad [8-8-24]$$

(1)

For  $x$  in the open interval  $] -1, 1 [$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.15: Sample page typeset with Helvetica Math fonts

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_Q f(x, y, z) dx dy dz$  and  $\prod_{Y \in \tilde{\mathcal{C}}} \partial(\tilde{X}_Y)$ ; and also on display:

8-8-25

$$\begin{aligned} \iiint_Q f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial Q} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\approx \bigcup_{Q \in \tilde{\mathcal{Q}}} \left[ f^* \left( \frac{\int Q(t)}{\sqrt{1-t^2}} \right) \right]_{t=a}^{t=\ell} \end{aligned} \quad (1)$$

For  $x$  in the open interval  $] -1, 1 [$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8.16: Sample page typeset with Informal Math fonts

This page spread shows two sans serif set-ups and an “informal” math font set-up. The solutions involving sans serif fonts can be usefully deployed in many circumstances, such as conventional articles, presentations (e.g., slides, reports), online documentation, or magazines. On the other hand, the Informal Math solution should probably be confined to announcements, fliers, and similar material.

Figure 8.14 shows the Computer Modern Bright set of fonts (designed by Walter Schmidt), which are based on the Computer Modern font design. The solution offers the full range of math symbols in normal and bold weights and is activated by loading the `cmbright` package; see Section 7.7.3. The fonts are freely available in METAFONT format, and the Type 1 versions are commercially available from MicroPress.<sup>1</sup>

Figure 8.15 shows a math font set-up for use with Helvetica (originally designed by Max Miedinger). The HV math fonts have been designed at MicroPress<sup>1</sup> and comprise the full set of mathematical symbols. The set-up is activated by loading the packages `hvmath` and `hvams` (for the AMS symbol set). While the Type 1 fonts are only commercially available, you can obtain 300dpi bitmapped fonts free of charge from MicroPress.

Finally, Figure 8.16 shows the Informal Math solution also offered by MicroPress.<sup>1</sup> The font design is loosely based on Adobe’s Tekton family of fonts. The set-up is activated by loading the `infomath` package. Note that the text fonts are only available in OT1 and that the AMS symbol set is not supported.

<sup>1</sup><http://www.micropress-inc.com>

## 8.9 Symbols in formulas

The tables at the end of this section advertise the large range of mathematical symbols provided by the  $\mathcal{AM}$ S fonts packages, including the command to use for each symbol. They also include the supplementary symbols from the St Mary Road Font, which was designed by Alan Jeffrey and Jeremy Gibbons. This font extends the Computer Modern and  $\mathcal{AM}$ S symbol font collections; the corresponding `stmaryrd` package should normally be loaded in addition to `amssymb`, but always after it. It provides extra symbols for fields such as functional programming, process algebra, domain theory, linear logic, and many more. For a wealth of information about an even wider variety of symbols, see the *Comprehensive L<sup>A</sup>T<sub>E</sub>X Symbol List* by Scott Pakin [134].

The tables indicate which extra packages need to be loaded to use each symbol command. They are organized as follows: symbols with command names in black are available in standard L<sup>A</sup>T<sub>E</sub>X without loading further packages; symbols in blue require loading either `amsmath`, `amssymb`, or `stmaryrd`, as explained in the table notes. If necessary, further classification is given by markings: <sup>(SM)</sup> signals a symbol from `stmaryrd` when the table also contains symbols from other packages; <sup>(kernel)</sup> identifies symbols that are available in standard L<sup>A</sup>T<sub>E</sub>X but only by combining two or more glyphs, whereas a single glyph exists in the indicated package; and <sup>(var)</sup> marks “Alphabetic characters/symbols” (of type `\mathalpha`; see Table 7.30 on page 435) that change appearance when used within the scope of a math alphabet identifier (see Section 7.4).

### 8.9.1 Mathematical symbol classes

The symbols are classified primarily by their “mathematical class”, occasionally called their “math symbol type”. This classification is related to their “meaning” in standard technical usage, but its importance for mathematical typography is that it influences the layout of a formula. For example, T<sub>E</sub>X’s mathematical formatter adjusts the horizontal space on either side of each symbol according to its mathematical class. There are also some finer distinctions made, for example, between accents and simple symbols and in breaking up the enormous list of Relation symbols into several tables.

The set-up for mathematics puts each symbol into one of these classes: Ordinary (Ord), Operator (Op), Binary (Bin), Relation (Rel), Opening (Open), Closing (Close), or Punctuation (Punct). This classification can be explicitly changed by using the commands `\mathord`, `\mathop`, `\mathbin`, `\mathrel`, `\mathopen`, `\mathclose`, and `\mathpunct`, thereby altering the surrounding spacing. In this example, `\#` and `\top` (both Ord by default) are changed into a Rel and an Op.

$a \# \top^\alpha_x x_b^\alpha$	<code>\usepackage[fleqn]{amsmath}</code>	
$a \# \frac{\alpha}{x} x_b^\alpha$	<code>\begin{array}{l} \begin{array}{ll} \# &amp; \top^\alpha_x x_b^\alpha \\ \mathrel{\{\#}} &amp; \mathop{\{\top\}}^\alpha_x x_b^\alpha \end{array} \end{array}</code>	8-9-1

		Right Object							
		Ord	Op	Bin	Rel	Open	Close	Punct	Inner
<i>Left Object</i>	Ord	0	1	(2)	(3)	0	0	0	(1)
	Op	1	1	*	(3)	0	0	0	(1)
	Bin	(2)	(2)	*	*	(2)	*	*	(2)
	Rel	(3)	(3)	*	0	(3)	0	0	(3)
	Open	0	0	*	0	0	0	0	0
	Close	0	1	(2)	(3)	0	0	0	(1)
	Punct	(1)	(1)	*	(1)	(1)	(1)	(1)	(1)
	Inner	(1)	1	(2)	(3)	(1)	0	(1)	(1)

*0 = no space, 1 = \thinmuskip, 2 = \medmuskip, 3 = \thickmuskip, \* = impossible*

*Entries in (blue) are not added when in the mathematical “script styles” (see also Sections 8.7.1 and 8.7.6).*

Table 8.7: Space between symbols

A symbol can be declared to belong to one of the above classes using the mechanism described in Section 7.10.7. In addition, certain sub-formulas—most importantly fractions, and those produced by `\left` and `\right`—form a class called Inner; it is explicitly available through the `\mathinner` command.

In TeX, spacing within formulas is done simply by identifying the class of each object in a formula and then adding space between each pair of adjacent objects as defined in Table 8.7; this table is unfortunately hard-wired into TeX’s mathematical typesetting routines and so cannot be changed by macro packages.<sup>1</sup> In this table 0, 1, 2, and 3 stand for no space, a thin space (`\,`), a medium space (`\:`), and a thick space (`\;`), respectively. The exact amounts of space used are listed in Section 8.7.6 on page 507.

A Binary symbol is turned into an Ordinary symbol whenever it is not preceded and followed by symbols of a nature compatible with a binary operation; for this reason, some entries in the table are marked with a star to indicate that they are not possible. For example, `$+x$` gives `+x` (a “unary plus”) and not `+ x`; the latter can be produced by `$\{}+x$\code>.`

Finally, an entry in (blue) in Table 8.7 indicates that the corresponding space is not inserted when the style is script or scriptscript.

As an example of applying these rules, consider the following formula (the default values are deliberately changed to show the added spaces more clearly):

8-9-2	$a - b = - \max\{x, y\}$	$\begin{array}{l} \backslash \text{thinmuskip}=10mu \backslash \text{medmuskip}=17mu \backslash \text{thickmuskip}=30mu \\ \quad \backslash [ \\ \quad \quad a - b = -\max \{ x , y \} \\ \quad \backslash ] \end{array}$
-------	--------------------------	---

<sup>1</sup>Although a few of the entries in the table are questionable, on the whole it gives pleasing results.

<i>A B C D E F G H I J K L M N O P Q R S T U V W X Y Z</i>
<i>a b c d e f g h i j k l m n o p q r s t u v w x y z</i>
<i>0 1 2 3 4 5 6 7 8 9</i>

Table 8.8: Latin letters and Arabic numerals

$\text{\TeX}$  identifies the objects as Ord, Bin, Ord, and so on, and then inserts spaces as follows:

```
a      -      b      =      -      \max  \{ x , y \}
Ord \: Bin \: Ord \; Rel \; Ord \; Op Open Ord Punct \; Ord Close
```

The minus in front of `\max` is turned into an Ordinary because a Binary cannot follow a Relation.

Table 8.7 reveals a difference<sup>1</sup> between a “`\left...` `\right`” construction, in which the entire sub-formula delimited by the construction becomes a single object of class Inner (see Section 8.5.3 on page 498), and commands like `\Bigl` and `\Bigr` that produce individual symbols of the classes Opening and Closing, respectively. Although they may result in typesetting delimiters of equal vertical size, spacing differences can arise depending on adjacent objects in the formula. For example, Ordinary followed by Opening gets no space, whereas Ordinary followed by Inner is separated by a thin space. The spaces inside the sub-formula within a “`\left...` `\right`” construction are as expected, beginning with an Opening symbol and ending with a Closing symbol. In this example we again use larger spaces to highlight the difference.

$a(\sum x)$	$\neq$	$a (\sum x)$
		<small><code>\thinmuskip=10mu \medmuskip=17mu \thickmuskip=30mu</code></small>
		<small><code>\[ a \Bigl( \sum x \Bigr) \neq a \left( \sum x \right) \]</code></small>

8-9-3

In summary, it is not enough to look up a symbol in the tables that follow; rather, it is also advisable to check that the symbol has the desired mathematical class to ensure that it is properly spaced when used. Example 8-9-4 on page 528 shows how to define new symbols that differ only in their mathematical class from existing symbols.

### 8.9.2 Letters, numerals, and other Ordinary symbols

The unaccented ASCII Latin letters and Arabic numeral digits (see Table 8.8) all referred to as “Alphabetic symbols”. The font used for them can vary: in mathematical formulas, the default font for Latin letters is italic whereas for the Arabic digits it is upright/Roman. Alphabetic symbols are all of class Ordinary.

<sup>1</sup>Another important distinction is that the material within a “`\left...` `\right`” construction is processed separately as a sub-formula (see Section 8.7.2 on page 503).

$\Delta$	<code>\Delta<sup>(var)</sup></code>	$\Gamma$	<code>\Gamma<sup>(var)</sup></code>	$\Lambda$	<code>\Lambda<sup>(var)</sup></code>	$\Omega$	<code>\Omega<sup>(var)</sup></code>	$\Phi$	<code>\Phi<sup>(var)</sup></code>
$\Pi$	<code>\Pi<sup>(var)</sup></code>	$\Psi$	<code>\Psi<sup>(var)</sup></code>	$\Sigma$	<code>\Sigma<sup>(var)</sup></code>	$\Theta$	<code>\Theta<sup>(var)</sup></code>	$\Upsilon$	<code>\Upsilon<sup>(var)</sup></code>
$\Xi$	<code>\Xi<sup>(var)</sup></code>	$\alpha$	<code>\alpha</code>	$\beta$	<code>\beta</code>	$\chi$	<code>\chi</code>	$\delta$	<code>\delta</code>
$\digamma$	<code>\digamma</code>	$\epsilon$	<code>\epsilon</code>	$\eta$	<code>\eta</code>	$\gamma$	<code>\gamma</code>	$\iota$	<code>\iota</code>
$\kappa$	<code>\kappa</code>	$\lambda$	<code>\lambda</code>	$\mu$	<code>\mu</code>	$\nu$	<code>\nu</code>	$\omega$	<code>\omega</code>
$\phi$	<code>\phi</code>	$\pi$	<code>\pi</code>	$\psi$	<code>\psi</code>	$\rho$	<code>\rho</code>	$\sigma$	<code>\sigma</code>
$\tau$	<code>\tau</code>	$\theta$	<code>\theta</code>	$\upsilon$	<code>\upsilon</code>	$\varepsilon$	<code>\varepsilon</code>	$\varkappa$	<code>\varkappa</code>
$\varphi$	<code>\varphi</code>	$\varpi$	<code>\varpi</code>	$\varrho$	<code>\varrho</code>	$\varsigma$	<code>\varsigma</code>	$\vartheta$	<code>\vartheta</code>
$\xi$	<code>\xi</code>	$\zeta$	<code>\zeta</code>						

Symbols in blue require the `amssymb` package. <sup>(var)</sup> indicates a variable Alphabetic symbol.

Table 8.9: Symbols of class `\mathord` (Greek)

Unlike the Latin letters, the mathematical Greek letters are no longer closely related to the glyphs used for typesetting normal Greek text. Due to an interesting 18th-century happenstance, in the major European tradition of mathematical typography the default font for lowercase Greek letters in mathematical formulas is italic whereas for uppercase Greek letters it is upright/Roman. (In other fields, such as physics and chemistry, the typographical traditions are slightly different.)

The capital Greek letters in the first columns of Table 8.9 are also Alphabetic symbols whose font varies, with the default being upright/Roman. Those capital Greek letters not present in this table are the letters that have the same appearance as some Latin letter (e.g., *A* and *Alpha*, *B* and *Beta*, *K* and *Kappa*, *O* and *Omicron*). Similarly, the list of lowercase Greek letters contains no omicron because it would be identical in appearance to the Latin *o*. Thus, in practice, the Greek letters that have Latin look-alikes are not used in mathematical formulas.

Table 8.10 lists other letter-shaped symbols of class `Ordinary`. The first four are Hebrew letters. Table 8.11 lists the remaining symbols in the `Ordinary` class,

$\aleph$	<code>\aleph</code>	$\beth$	<code>\beth</code>	$\daleth$	<code>\daleth</code>	$\gimel$	<code>\gimel</code>
$\$$	<code>\\$</code>	$\Im$	<code>\Im</code>	$\Re$	<code>\Re</code>	$\Bbbk$	<code>\Bbbk</code>
$\circledS$	<code>\circledS</code>	$\complement$	<code>\complement</code>	$\ell$	<code>\ell</code>	$\eth$	<code>\eth</code>
$\Finv$	<code>\Finv</code>	$\Game$	<code>\Game</code>	$\hbar$	<code>\hbar</code>	$\hslash$	<code>\hslash</code>
$\imath$	<code>\imath</code>	$\jmath$	<code>\jmath</code>	$\mathdollar$	<code>\mathdollar</code>	$\mathparagraph$	<code>\mathparagraph</code>
$\mathsection$	<code>\mathsection</code>	$\mathsterling$	<code>\mathsterling</code>	$\mathmo$	<code>\mathmo</code>	$\mathP$	<code>\mathP</code>
$\partial$	<code>\partial</code>	$\mathpounds$	<code>\mathpounds</code>	$\mathS$	<code>\mathS</code>	$\wp$	<code>\wp</code>

Symbols in blue require the `amssymb` package.

Synonyms:  $\$$  `\mathdollar`,  $\$$  `\mathparagraph`,  $\mathP$  `\mathP`,  $\mathsection$  `\mathsection`,  $\mathS$  `\mathS`,  $\mathsterling$  `\mathsterling`,  $\mathpounds$  `\mathpounds`

Table 8.10: Symbols of class `\mathord` (letter-shaped)

!	!	.	.	/	/
?	?	@	@		
#	\#	%	\%	&	\&
-	\_	=	\	\angle	\angle^{(kernel)}
//	\Arrowvert		\arrowvert	\backprime	\blacklozenge
\	\backslash	\bigstar	\bigstar	\blacktriangle	\blacktriangledown
\blacksquare	\blacksquare	\bracevert	\bracevert	\clubsuit	\clubsuit
\bot	\bot	\diagdown	\diagdown	\exists	\exists
\copyright	\copyright	\emptyset	\emptyset	\forall	\forall
\diamondsuit	\diamondsuit	\flat	\flat	\heartsuit	\heartsuit
\flat	\flat	\forall	\forall	\lnot	\lnot
\infty	\infty	\lightning^{(STM)}	\lightning^{(STM)}	\nabla	\nabla
\lozenge	\lozenge	\measuredangle	\measuredangle	\exists .	\exists .
\natural	\natural	\neg	\neg	\sharp	\sharp
\prime	\prime	\sharp	\sharp	\spadesuit	\spadesuit
\sphericalangle	\sphericalangle	\square	\square	\surd	\surd
\top	\top	\triangle	\triangle	\triangledown	\triangledown
\varcopyright^{(STM)}	\varcopyright^{(STM)}	\varnothing	\varnothing	\Vert	\Vert
\vert	\vert				

Symbols in blue require either the `amssymb` package or, if flagged with `(STM)`, the `stmaryrd` package.

Note that the exclamation sign, period, and question mark are not treated as punctuation in formulas.

Synonyms: `\lnot`, `\neg` | `\vert`, `|` || `\Vert`, `\|`

Table 8.11: Symbols of class `\mathord` (miscellaneous)

including some common punctuation. These behave like letters and digits, so they never get any extra space around them.

A common mistake is to use the symbols from Table 8.11 directly as Binary operator or Relation symbols, without using a properly defined math symbol command for that type. Thus, if you use commands such as `\#`, `\square`, or `\&`, check carefully that you get the correct inter-symbol spaces or, even better, define your own symbol command.

$a \neg b$	$x \square y + z$	<code>\usepackage[fleqn]{amsmath} \usepackage{amssymb}</code> <code>\DeclareMathSymbol\bneg {\mathbin}{symbols}{3A}</code> <code>\DeclareMathSymbol\rsquare{\mathrel}{AMSA}{03}</code>	
$a \neg b$	$x \square y + z$	<code>\[ a \neg b \qquad x \square y + z \]</code>	8-9-4 ---
$a \neg b$	$x \square y + z$	<code>\[ a \mathbin{\neg} b \qquad x \mathrel{\square} y + z \]</code>	

The `\DeclareMathSymbol` declaration is explained in Section 7.10.7. The correct values for its arguments are most easily found by looking at the definitions

$\acute{x}$	<code>\acute{x}</code>	$\bar{x}$	<code>\bar{x}</code>	$\breve{x}$	<code>\breve{x}</code>	$\check{x}$	<code>\check{x}</code>
$\ddot{x}$	<code>\ddot{x}</code>	$\ddot{\bar{x}}$	<code>\ddot{\bar{x}}</code>	$\ddot{\breve{x}}$	<code>\ddot{\breve{x}}</code>	$\dot{x}$	<code>\dot{x}</code>
$\grave{x}$	<code>\grave{x}</code>	$\hat{x}$	<code>\hat{x}</code>	$\mathring{x}$	<code>\mathring{x}</code>	$\tilde{x}$	<code>\tilde{x}</code>
$\vec{x}$	<code>\vec{x}</code>	$\widehat{xyz}$	<code>\widehat{xyz}</code>	$\widetilde{xyz}$	<code>\widetilde{xyz}</code>		

Accents in blue require the `amsmath` package.

The last two accents are available in a range of widths, the largest suitable one being automatically used.

Table 8.12: Mathematical accents, giving sub-formulas of class `\mathord`

in the file `amssymb.sty` or `fontmath.ltx` (for the core symbols). For example, we looked up `\neq` and `\square`, replaced the `\mathord` in each case, and finally gave the resulting symbol a new name.

### 8.9.3 Mathematical accents

The accent commands available for use in formulas are listed in Table 8.12. Most of them are already defined in standard L<sup>A</sup>T<sub>E</sub>X. See Section 8.4.8 for ways to define additional accent commands and Section 8.5.2 for information about extensible accents. Adding a mathematical accent to a symbol always produces a symbol of class Ordinary. Thus, without additional help, one cannot use the accents to produce new Binary or Relation symbols.

<span style="border: 1px solid black; padding: 2px;">8-9-5</span>	<pre>\usepackage{amstext} a = b but a \tilde{=} b which is not a \tilde{=} b \text{ but } a \mathrel{\tilde{=}} b \text{ which is not } a \mathrel{\tilde{=}} b</pre>
---	---

Other ways to place symbols over Relation symbols are shown in Section 8.4.10. When adding an accent to an *i* or *j* in mathematics, it is best to use the dotless variants `\imath` and `\jmath`; for example, use `\hat{\jmath}` to get  $\hat{j}$ .

### 8.9.4 Binary operator symbols

There are more than 100 symbols of class Binary operators from which to choose. Most of these Binary symbols are shown in Table 8.13 on the next page. Some of them are also available, under different names, as Relation symbols.

The `amssymb` package offers a few box symbols for use as Binary operators; many more are added by `stmaryrd`. These are shown in Table 8.14.

The `stmaryrd` package can be loaded with the option `heavycircles`. It causes each circle symbol command in Table 8.15 on page 531 that starts with `\var` to swap its definition with the corresponding command without the “var”; for example, the symbol `\varodot` becomes `\odot`, and vice versa.

*	*	+	+	-	-
\amalg	\ast	\bbslash <sup>(StM)</sup>	\Cap	\baro <sup>(StM)</sup>	\bigtriangledown
\barwedge	\bigtriangleup	\Cap	\cup	\cap	\bigtriangleleft
\bigtriangleup	\bigtriangleup	\cup	\dag	\curlyvee	\bigtriangledown
\Cup	\Cap	\dag	\ddagger	\dagger	\dagger
\curlywedge	\cup	\ddagger	\divideontimes	\diamond	\diamond
\ddag	\divideontimes	\doublecup	\dotplus	\gtrdot	\fatbslash <sup>(StM)</sup>
\div	\doublecup	\fatslash <sup>(StM)</sup>	\interleave <sup>(StM)</sup>	\land	\fatdot
\doublecap	\fatslash <sup>(StM)</sup>	\interleave <sup>(StM)</sup>	\leftslice <sup>(StM)</sup>	\leftthreetimes	\leftthreetimes
\fatsemi <sup>(StM)</sup>	\fatslash <sup>(StM)</sup>	\leftslice <sup>(StM)</sup>	\lor	\ltimes	\ltimes
\intercal	\fatslash <sup>(StM)</sup>	\minuso <sup>(StM)</sup>	\minuso <sup>(StM)</sup>	\moo <sup>(StM)</sup>	\mp
\lbag <sup>(StM)</sup>	\leftslice <sup>(StM)</sup>	\nplus <sup>(StM)</sup>	\nplus <sup>(StM)</sup>	\pm	\pm
\lessdot	\lor	\rightslice <sup>(StM)</sup>	\setminus	\rightthreetimes	\rightthreetimes
\merge <sup>(StM)</sup>	\minuso <sup>(StM)</sup>	\rightslice <sup>(StM)</sup>	\sqcup	\smallsetminus	\sslash <sup>(StM)</sup>
\mp	\nplus <sup>(StM)</sup>	\sqcup	\talloblong <sup>(StM)</sup>	\times	\times
\rbag <sup>(StM)</sup>	\setminus	\talloblong <sup>(StM)</sup>	\triangleright	\uplus	\uplus
\rtimes	\sqcup	\triangleright	\varbigtriangleup <sup>(StM)</sup>	\varcurlyvee <sup>(StM)</sup>	\vee
\sqcap	\sqcup	\varbigtriangleup <sup>(StM)</sup>	\vartimes <sup>(StM)</sup>	\vee	\vee
\star	\star	\vartimes <sup>(StM)</sup>	\wedge	\wr	\wr
\triangleleft	\star	\wedge	\left <sup>(StM)</sup>	\leftright <sup>(StM)</sup>	\right <sup>(StM)</sup>
\varbigtriangledown <sup>(StM)</sup>	\triangleleft	\left <sup>(StM)</sup>	\veebar	\wr	\wr
\varcurlywedge <sup>(StM)</sup>	\varbigtriangleup <sup>(StM)</sup>	\veebar	\wedge	\wr	\wr
\veebar	\varbigtriangleup <sup>(StM)</sup>	\wedge	\leftright <sup>(StM)</sup>	\wr	\wr
\Ydown <sup>(StM)</sup>	\varbigtriangledown <sup>(StM)</sup>	\leftright <sup>(StM)</sup>	\wr	\wr	\wr
\Yup <sup>(StM)</sup>	\wr	\wr	\wr	\wr	\wr

Symbols in blue require either the `amssymb` package or, if flagged with <sup>(StM)</sup>, the `stmaryrd` package.

The left and right triangles are also available as Relation symbols.

The `stmaryrd` package confusingly changes the Binary symbols `\bigtriangleup` and `\bigtriangledown` into Operators, leaving only the synonyms `\varbigtriangleup` and `\varbigtriangledown` for the Binary operator forms.

Synonyms:  $\wedge \backslash land, \wedge \backslash wedge$      $\vee \backslash lor, \vee \backslash vee$      $\sqcup \backslash doublecup, \backslash Cup$      $\sqcap \backslash doublecap, \backslash Cap$

\* `\ast`, \*     $\dag \backslash dag, \ddagger \backslash ddagger$

Table 8.13: Symbols of class `\mathbin` (miscellaneous)

\boxast <sup>(StM)</sup>	\boxbar <sup>(StM)</sup>	\boxbox <sup>(StM)</sup>	\boxbslash <sup>(StM)</sup>
\boxcircle <sup>(StM)</sup>	\boxdot	\boxempty <sup>(StM)</sup>	\boxminus
\boxplus	\boxslash <sup>(StM)</sup>	\boxtimes	\oblong <sup>(StM)</sup>

All symbols require either the `amssymb` package or, if flagged with <sup>(StM)</sup>, the `stmaryrd` package.

Table 8.14: Symbols of class `\mathbin` (boxes)

○	\bigcirc	• \bullet
•	\centerdot	◦ \cdot
◎	\circledcirc	⊛ \circledast
⊖	\obar <sup>(StM)</sup>	⊸ \oast <sup>(StM)</sup>
⊕	\odot	⊸ \ocircle <sup>(StM)</sup>
⊖	\ominus	⊸ \olessthan <sup>(StM)</sup>
⊗	\otimes	⊗ \oslash
○	\varbigcirc <sup>(StM)</sup>	⊛ \owedge <sup>(StM)</sup>
⊗	\varobslash <sup>(StM)</sup>	⊖ \varobar <sup>(StM)</sup>
⊖	\varogreaterthan <sup>(StM)</sup>	⊕ \varodot <sup>(StM)</sup>
⊕	\varoplus <sup>(StM)</sup>	⊖ \varominus <sup>(StM)</sup>
⊗	\varovee <sup>(StM)</sup>	⊗ \varotimes <sup>(StM)</sup>
		⊛ \varowedge <sup>(StM)</sup>

*Symbols in blue require either the `amssymb` package or, if flagged with <sup>(StM)</sup>, the `stmaryrd` package.*

*Option `heavycircles` of the `stmaryrd` package affects all commands starting with `\var` and their normal variants.*

*Synonyms:* ⊛ `\oast`, `\circledast` ⊛ `\ocircle`, `\circledcirc`

Table 8.15: Symbols of class `\mathbin` (circles)

### 8.9.5 Relation symbols

The class of binary Relation symbols forms a collection even larger than that of the Binary operators. The lists start with symbols for equality and order (Table 8.16 on the next page). You can put a slash through any Relation symbol by preceding it with the `\not` command; this negated symbol represents the complement (or negation) of the relation.

8-9-6	$u \not\prec v$ or $a \notin A$	$\$ u \not< v \$$ or $\$ a \not\in A \$$
-------	---------------------------------	--

Especially with larger symbols, this generic method of negating a Relation symbol does not always give good results because the slash will always be of the same size, position, and slope. Therefore, some specially designed “negated symbols” are also available (see Table 8.17 on the following page). If a choice is available, the designed glyphs are usually preferable. To see why, compare the symbols in this example.

8-9-7	$\not\leq$ $\not\geq$ $\not\approx$	<pre>\usepackage{amssymb} \$ \not\leq \not\succ \not\sim \$ \par \$ \not\leq \not\succ \not\sim \$</pre>
-------	---	--

Next come the Relation symbols for sets and inclusions, and their negations (see Tables 8.18 and 8.19).

<	<	=	=	>	>	>	>	≈	≈	\approx
≈	≈	\approxeq	≈	\asymp	≈	\backsim	≈	\backsimeq	≈	\backsimeq
△	△	\Bumpeq	△	\bumpeq	△	\circeq	△	\cong	△	\cong
≲	≲	\curlyeqprec	≲	\curlyeqsucc	≲	\Doteq	≲	\doteq	≲	\doteq
≷	≷	\doteqdot	≷	\eqcirc	≷	\eqsim	≷	\eqslantgtr	≷	\eqslantgtr
≸	≸	\eqslantless	≸	\equiv	≸	\fallingdotseq	≸	\ge	≸	\ge
≹	≹	\geq	≹	\geqq	≹	\geqslant	≹	\gg	≹	\gg
≺	≺	\ggg	≺	\ggtr	≺	\gtrapprox	≺	\gtreqless	≺	\gtreqless
≻	≻	\gtreqqless	≻	\gtless	≻	\gtrsim	≻	\leftarrowrightarroweq <sup>(SMT)</sup>	≻	\leftarrowrightarroweq <sup>(SMT)</sup>
≴	≴	\leq	≴	\leqq	≴	\leqslant	≴	\lessapprox	≴	\lessapprox
≵	≵	\lesseqgtr	≵	\lesseqgtr	≵	\lessgtr	≵	\lessim	≵	\lessim
≶	≶	\le	≶	\ll	≶	\lll	≶	\llless	≶	\llless
≷	≷	\prec	≷	\precapprox	≷	\preccurlyeq	≷	\preceq	≷	\preceq
≸	≸	\precsim	≸	\risingdotseq	≸	\sim	≸	\simeq	≸	\simeq
≹	≹	\succ	≹	\succapprox	≹	\succcurlyeq	≹	\succeq	≹	\succeq
≺	≺	\succsim	≺	\thickapprox	≺	\thicksim	≺	\triangleq	≺	\triangleq

Symbols in blue require either the `amssymb` package or, if flagged with <sup>(SMT)</sup>, the `stmaryrd` package.

Synonyms:  $\leq \leq \leq \leq \leq \leq \leq \leq \leq \leq \leq$   $\geq \geq \geq \geq \geq \geq \geq \geq \geq \geq$   $\approx \approx \approx \approx \approx \approx \approx \approx \approx \approx$

Table 8.16: Symbols of class `\mathrel{}` (equality and order)

≷	≷	\gnapprox	≷	\gneq	≷	\gneqq	≷	\gnsim
≸	≸	\gvertneqq	≸	\lnapprox	≸	\lneq	≸	\lneqq
≶	≶	\lnsim	≶	\lvertneqq	≶	\ncong	≶	\ne
≷	≷	\neq	≷	\ngeq	≷	\ngeqq	≷	\ngeqslant
≸	≸	\ngtr	≸	\nleq	≸	\nleqq	≸	\nleqslant
≶	≶	\nless	≶	\nprec	≶	\npreceq	≶	\nsim
≷	≷	\nsucc	≷	\nsuccapprox	≷	\precnapprox	≷	\precneqq
≸	≸	\precnsim	≸	\succnapprox	≸	\succneqq	≸	\succnsim

Symbols in blue require either the `amssymb` package or, flagged with <sup>(SMT)</sup>, the `stmaryrd` package.

Synonyms:  $\neq \neq \neq \neq \neq \neq \neq \neq \neq \neq$

Table 8.17: Symbols of class `\mathrel{}` (equality and order—negated)

$\blacktriangleleft$	$\blacktriangleright$	$\in$
$\inplus^{(StM)}$	$\ni$	$\niplus^{(StM)}$
$\ntrianglelefteqslant^{(StM)}$	$\ntrianglerighteqslant^{(StM)}$	$\owns$
$\sqsubset$	$\sqsubseteq$	$\sqsupset$
$\sqsupseteq$	$\Subset$	$\subset$
$\subseteq$	$\subseteqqq$	$\subsetplus^{(StM)}$
$\subsetplus^{(StM)}$	$\Supset$	$\supset$
$\supseteq$	$\supseteqqq$	$\supsetplus^{(StM)}$
$\supsetplus^{(StM)}$	$\trianglelefteq$	$\trianglelefteqslant^{(StM)}$
$\trianglerighteq$	$\trianglerighteqslant^{(StM)}$	$\vartriangle$
$\vartriangleleft$	$\vartriangleright$	

*Symbols in blue require either the `amssymb` package or, if flagged with  $^{(StM)}$ , the `stmaryrd` package.*

*Synonyms:*  $\ni \equiv \owns, \in \ni$

Table 8.18: Symbols of class `\mathrel` (sets and inclusion)

$\notin$	$\not\in$	$\not\subseteq$	$\not\subseteqqq$	$\not\supset$	$\not\supsetplus^{(StM)}$
$\not\supseteq$	$\not\supseteqqq$	$\not\supseteq$	$\not\supseteqqq$	$\not\trianglelefteq$	$\not\trianglelefteqslant^{(StM)}$
$\ntrianglelefteq$	$\ntrianglerighteq$	$\ntriangleright$	$\ntrianglerighteq$	$\ntrianglerighteq$	$\ntrianglerighteqslant^{(StM)}$
$\subsetneq$	$\subsetneqq$	$\subsetneq$	$\subsetneqq$	$\supsetneq$	$\supsetneqq$
$\supsetneq$	$\supsetneqq$	$\varsubsetneq$	$\varsubsetneqq$	$\varsupsetneq$	$\varsupsetneqq$
$\varsupsetneq$	$\varsupsetneqq$				

*Symbols in blue require the `amssymb` package.*

Table 8.19: Symbols of class `\mathrel` (sets and inclusion—negated)

They are followed by Relation symbols that are arrow-shaped (see Tables 8.20 and 8.21). Some extensible arrow constructions that produce compound Relation symbols are described in Section 8.5.2 on page 497.

In addition to `\not`, used to negate general Relation symbols, other building blocks have been especially designed to negate or extend arrow-like symbols; these are collected in Table 8.22.

8-9-8:	$\leftrightarrow \not\leftrightarrow$	$\usepackage{stmaryrd}$
		$\$ \Longarrownot \longleftrightarrow \qquad \arrownot \hookleftarrow \$$

Finally, in Table 8.23 on page 535 you will find a miscellaneous collection of Relation symbols.

○ \circlearrowleft	○ \circlearrowright	▽ \curlyveeuparrow <sup>(STMD)</sup>
▽ \curlyveeuparrow <sup>(STMD)</sup>	△ \curlywedgedownarrow <sup>(STMD)</sup>	△ \curlywedgeuparrow <sup>(STMD)</sup>
↷ \curvearrowleft	↷ \curvearrowright	↔ \dasharrow
←-- \dashleftarrow	→-- \dashrightarrow	↓ \Downarrow
↓ \downarrow	↓\downarrow	↓ \downharpoonright
← \gets	← \hookleftarrow	→ \hookrightarrow
⇐ \Leftarrow	⇐ \leftarrow	⇒ \leftarrowtail
← \leftarrowtriangle <sup>(STMD)</sup>	↔ \leftrightarrowtriangle <sup>(STMD)</sup>	← \leftharpoondown
← \leftharpoonup	⇐ \leftleftarrows	⇒ \Leftrightarrow
↔ \leftrightarrow	⇒ \leftrightarrows	⇒ \leftrightharpoons
↔ \leftrightsquigarrow	⇒ \Lleftarrow	⇒ \Longleftarrow
← \longleftarrow	↔ \Longleftrightarrow	↔ \longleftarrow
⇐ \Longmapsfrom <sup>(STMD)</sup>	↔ \longmapsfrom <sup>(STMD)</sup>	⇒ \Longmapsto <sup>(STMD)</sup>
→ \longmapsto	⇒ \Longrightarrow	→ \longrightarrow
↔ \looparrowleft	↔ \looparrowright	↑ \Lsh
⇐ \Mapsfrom <sup>(STMD)</sup>	↔ \mapsfrom <sup>(STMD)</sup>	⇒ \Mapsto <sup>(STMD)</sup>
→ \mapsto	→ \multimap	↗ \nearrow
↗ \nnearrow <sup>(STMD)</sup>	↗ \nnarrow <sup>(STMD)</sup>	↖ \nwarrow
↑ \restriction	⇒ \Rrightarrow	→ \rightarrow
→ \rightarrowtail	→ \rightarrowtriangle <sup>(STMD)</sup>	→ \rightharpoondown
→ \rightharpoonup	⇒ \rightleftarrows	⇒ \rightleftharpoons
⇒ \rightarrows	↔ \rightsquigarrow	⇒ \Rrightarrow
↑ \Rsh	↘ \searrow	↓ \shortdownarrow
← \shortleftarrow <sup>(STMD)</sup>	→ \shortrightarrow <sup>(STMD)</sup>	↑ \shortuparrow
↓ \ssearrow <sup>(STMD)</sup>	↙ \ssarrow <sup>(STMD)</sup>	↖ \swarrow
→ \to	↔ \twoheadleftarrow	→ \twoheadrightarrow
↑ \Uparrow	↑ \uparrow	↔ \Updownarrow
↓ \updownarrow	↓ \upharpoonleft	↑ \upharpoonright
↑\upuparrows		

Symbols in blue require either the `amssymb` package or, if flagged with <sup>(STMD)</sup>, the `stmaryrd` package.

Synonyms:  $\leftarrow \gets, \leftarrow \leftarrowtriangle$     $\rightarrow \rightarrowtail, \rightarrow \rightarrowtriangle$     $\restriction \restriction, \upharpoonright$   
 $\dashrightarrow \dashrightarrow, \dashleftarrow$

Table 8.20: Symbols of class `\mathrel` (arrows)

≯ \nLeftarrow	≮ \nleftarrow	≯ \nLeftrightarrow
≰ \nlefrightharpoonup	≰ \nRightarrow	≰ \nrightarrow

Symbols in blue require the `amssymb` package.

Table 8.21: Symbols of class `\mathrel` (arrows—negated)

$/ \backslash Arrownot^{(StM)}$	$\backslash arrownot^{(StM)}$	$\leftarrow \lhook$	$\backslash Longarrownot^{(StM)}$
$\backslash longarrownot^{(StM)}$	$\mid \backslash Mapsfromchar^{(StM)}$	$\mid \backslash mapsfromchar^{(StM)}$	$\mid \backslash Mapstochar^{(StM)}$
$\mid \backslash mapstochar$	$/ \backslash not$	$\rightarrow \rhook$	

*Symbols in blue require the stmaryrd package.*

*These symbols are for combining, mostly with arrows; e.g.,  $\backslash longarrownot \backslash longleftarrow$  gives  $\longleftrightarrow$ .*

*Use  $\backslash joinrel$  to “glue” relational symbols together, e.g.,  $\lhook \backslash joinrel \backslash longrightarrow$  gives  $\longleftrightarrow$ .*

*The dimensions of these symbols make them unsuitable for other uses.*

Table 8.22: Symbols of class `\mathrel` (negation and arrow extensions)

$:$	$\because$	$\backslash backepsilon$	$\therefore$	$\backslash because$	$\between$	$\backslash between$
$\bowtie$	$\dashv$	$\backslash dashv$	$\frown$	$\backslash frown$	$\Join$	$\backslash Join$
$\mid$	$\models$	$\backslash models$	$\nmid$	$\backslash nmid$	$\nparallel$	$\backslash nparallel$
$\nshortmid$	$\nshortparallel$	$\backslash nshortparallel$	$\nVdash$	$\backslash nVDash$	$\nVdash$	$\backslash nVdash$
$\nvDash$	$\nvDash$	$\backslash nvDash$	$\parallel$	$\backslash parallel$	$\perp$	$\backslash perp$
$\pitchfork$	$\propto$	$\backslash propto$	$\shortmid$	$\backslash shortmid$	$\shortparallel$	$\backslash shortparallel$
$\smallfrown$	$\smallsmile$	$\backslash smallsmile$	$\smile$	$\backslash smile$	$\therefore$	$\backslash therefore$
$\varpropto$	$\vdash$	$\backslash vdash$	$\vDash$	$\backslash vDash$	$\vdash$	$\backslash vdash$
$\Vdash$						

*Relation symbols in blue require the amssymb package.*

*$\therefore$  is a Relation symbol, so its spacing may not be as expected in common uses.*

Table 8.23: Symbols of class `\mathrel` (miscellaneous)

### 8.9.6 Punctuation

The symbols of class Punctuation appear in Table 8.24, together with some other punctuation-like symbols. Note that some of the typical punctuation characters (i.e., “. ! ?”) are not set up as mathematical punctuation but rather as symbols of class Ordinary. This can cause unexpected results for common uses of these symbols, especially in the cases of ! and ?. Some of the dots symbols listed here are of class Inner; Section 8.5.1 on page 496 provides information about using dots for mathematical ellipsis.

The `:` character produces a colon with class Relation—not a Punctuation symbol. As an alternative, standard L<sup>A</sup>T<sub>E</sub>X offers the command `\colon` as the Punctuation symbol. However, the amsmath package makes unfortunate major changes to the spacing produced by the command `\colon`, so that it is useful only for a particular layout in constructions such as `f\colon A\rightarrow B` where it produces  $f: A \rightarrow B$ . It is therefore wise to always use `\mathpunct{:}` for the simple punctuation colon in mathematics.

,	,	...	<code>\cdots</code>
;	;	:	<code>\colon</code>
			‘‘.
			<code>\ddots</code>
			:
			<code>\vdots</code>

*Punctuation symbols in blue require the amsmath package.*

*The logical amsmath commands normally used to access `\cdots` and `\ldots` are described in Section 8.5.1.*

*The `\colon` command is redefined in amsmath, making it unsuitable for use as a general punctuation character.*

*Synonyms:* ... `\hdots`, `\ldots` ... `\mathellipsis`, `\ldots`

Table 8.24: Symbols of class `\mathpunct`, `\mathord`, `\mathinner` (punctuation)

$\int\int$	<code>\int</code>	$\oint\oint$	<code>\oint</code>	$\square\square$	<code>\bigbox</code> <sup>(StM)</sup>
$\cap\cap$	<code>\bigcap</code>	$\cup\cup$	<code>\bigcup</code>	$\curlyvee\curlyvee$	<code>\bigcurlyvee</code> <sup>(StM)</sup>
$\curlywedge\curlywedge$	<code>\bigcurlywedge</code> <sup>(StM)</sup>	$\ \ /\ /\ $	<code>\biginterleave</code> <sup>(StM)</sup>	$\oplus\oplus$	<code>\bignplus</code> <sup>(StM)</sup>
$\odot\odot$	<code>\bigodot</code>	$\oplus\oplus$	<code>\bigoplus</code>	$\otimes\otimes$	<code>\bigotimes</code>
$\parallel\parallel$	<code>\bigparallel</code> <sup>(StM)</sup>	$\sqcap\sqcap$	<code>\bigsqcap</code> <sup>(StM)</sup>	$\sqcup\sqcup$	<code>\bigsqcup</code>
$\nabla\nabla$	<code>\bigtriangledown</code> <sup>(StM)</sup>	$\triangle\triangle$	<code>\bigtriangleup</code> <sup>(StM)</sup>	$\uplus\uplus$	<code>\biguplus</code>
$\vee\vee$	<code>\bigvee</code>	$\wedge\wedge$	<code>\bigwedge</code>	$\coprod\coprod$	<code>\coprod</code>
$\prod\prod$	<code>\prod</code>	$\smallint\smallint$	<code>\smallint</code>	$\sum\sum$	<code>\sum</code>

*Operator symbols in blue require the stmaryrd package.*

*The stmaryrd package confusingly changes the Binary symbols `\bigtriangleup` and `\bigtriangledown` into Operators, but there are alternative commands for the Binary operator forms.*

*Note that `\smallint` does not change size.*

Table 8.25: Symbols of class `\mathop`

### 8.9.7 Operator symbols

The Operator symbols typically come in two sizes, for text and display uses; most of them are related to similar Binary operator symbols. Whether an Operator symbol takes limits in displays depends on a variety of factors (see Section 8.4.4). The available collection is shown in Table 8.25.

$\llbracket \rrbracket$	$\llbracket \rrbracket$	$\{ \}$	$\{ \} \backslash\{ \}$	$\llbracket \rrbracket$	$\backslash lVert \backslash rVert$
$\llbracket \rrbracket$	$\backslash lbrack \backslash rbrack$	$\{ \}$	$\backslash lbrace \backslash rbrace$	$\  \ \!$	$\backslash lvert \backslash rvert$
$\llbracket \rrbracket$	$\backslash lceil \backslash rceil$	$( )$	$( )$	$( )$	$\backslash lgroup \backslash rgroup$
$\llbracket \rrbracket$	$\backslash lfloor \backslash rfloor$	$\langle \rangle$	$\langle \rangle \backslash langle \backslash rangle$	$\  \ \!$	$\backslash lmoustache \backslash rmoustache$
$\llbracket \rrbracket$	$\backslash llbracket \backslash rrbracket^{(StM)}$				

*Delimiters in blue require either the `amsmath` package or, if flagged with <sup>(StM)</sup>, the `stmaryrd` package.*

*Synonyms:*  $\llbracket \rrbracket$ ,  $\llbracket \rrbracket \backslash lbrack \backslash rbrack$ ,  $\{ \}$ ,  $\{ \} \backslash lbrace \backslash rbrace$ ,  $\langle \rangle$

Table 8.26: Symbol pairs of class `\mathopen` and `\mathclose` (extensible)

$\llbracket \rrbracket$	$\backslash lceil \backslash rceil^{(StM)}$	$\& \&$	$\backslash binampersand \backslash bindnasrepma^{(StM)}$	$\  \ \!$	$\backslash Lbag \backslash Rbag^{(StM)}$
$\llbracket \rrbracket$	$\backslash lfloor \backslash rrfloor^{(StM)}$	$\langle \rangle$	$\backslash llparenthesis \backslash rrparenthesis^{(StM)}$		

*All these pairs of symbols require the `stmaryrd` package and are not extensible.*

Table 8.27: Symbol pairs of class `\mathopen` and `\mathclose` (non-extensible)

### 8.9.8 Opening and Closing symbols

The paired extensible delimiters, when used on their own (i.e., without a preceding `\left`, `\right`, or `\middle`), produce symbols of class Opening or Closing; these pairs are listed in Table 8.26. See Section 8.5.3 on page 498 for further information about the extensible symbols.

To improve the flexibility of the vertical bar notation, `amsmath` defines some new pairs of paired extensible delimiter commands: `\lvert`, `\rvert`, `\lVert`, and `\rVert`. These commands are comparable to standard  $\text{\LaTeX}$ 's `\langle` and `\rangle` commands.

The `stmaryrd` package adds a collection of non-extensible paired symbols of class Opening and Closing, which are listed in Table 8.27.



## C H A P T E R 9

# L<sup>A</sup>T<sub>E</sub>X in a Multilingual Environment

This chapter starts with a short introduction to the technical problems that must be solved if you want to use (L)A<sub>T</sub>E<sub>X</sub> with a non-English language. Most of the remaining part of the chapter discusses the `babel` system, which provides a convenient way of generating documents in different languages. We look in particular how we can typeset documents in French, German, Russian, Greek, and Hebrew, as the typesetting of those languages illustrates various aspects of the things one has to deal with in a non-English environment. Section 9.5 explains the structure of `babel`'s language definition files for the various language options. Finally, we say a few words about how to handle other languages, such as Arabic and Chinese, that are not supported by `babel`.

### 9.1 T<sub>E</sub>X and non-English languages

Due to its popularity in the academic world, T<sub>E</sub>X spread rapidly throughout the world and is now used not only with the languages based on the Latin alphabet, but also with languages using non-Latin alphabetic scripts, such as Russian, Greek, Arabic, Persian, Hebrew, Thai, Vietnamese, and several Indian languages. Implementations also exist for Chinese, Japanese, and Korean, which use Kanji-based ideographic scripts.

With the introduction of 8-bit T<sub>E</sub>X and METAFONT, which were officially released by Donald Knuth in March 1990, problems of multilingual support could be more easily addressed for the first time. Nevertheless, by themselves, these

versions do not solve all the problems associated with providing a convenient environment for using L<sup>A</sup>T<sub>E</sub>X with multiple and/or non-English languages.

To achieve this goal, T<sub>E</sub>X and its companion programs should be made truly international, and the following points should be addressed:

1. Adjust all programs to the particular language(s):
  - Support typesetting in different directions, this ability is offered by several programs (e.g., eT<sub>E</sub>X, Omega) [27, 97],
  - Create proper fonts containing national symbols [137],
  - Define standard character set encodings, and
  - Generate patterns for the hyphenation algorithm.
2. Provide a translation for the language-dependent strings, create national layouts for the standard documents, and provide T<sub>E</sub>X code to treat the language-dependent typesetting rules automatically [120].
3. Support processing of multilingual documents (more than one language in the same document) and work in international environments (one language per document, but a choice between several possibilities). For instance, the sorting of indexes and bibliographic references should be performed in accordance with a given language's alphabet and collating sequence; see the discussion on `xindy` in Section 11.3.

At the same time, you should be able to conveniently edit, view, and print your documents using any given character set, and L<sup>A</sup>T<sub>E</sub>X should be able to successfully process files created in this way. There exist, however, almost as many different character encoding schemes as there are languages (for example, IBM PC personal computers have dozens of code pages). In addition, several national and international standards exist, such as the series ISO 8859-x [67]. Therefore, some thought should be given to the question of compatibility and portability. If a document is to be reproducible in multiple environments, issues of standardization become important. In particular, sending 8-bit encoded documents via electronic mail generated problems at one time, because some mail gateways dropped the higher-order bit, rendering the document unprocessable. The e-mail problem is more or less solved now that almost all mailers adhere to the Multipart Internet Mail Extensions (MIME) standard, in which the use of a particular encoding standard (e.g., ISO 8859-x) is explicitly declared in the e-mail's header. The fact remains, however, that it is necessary to know the encoding in which a document was produced. For this purpose L<sup>A</sup>T<sub>E</sub>X offers the `inputenc` package, described in Section 7.11.3 on page 443.

Document encoding problems will ultimately be solved when new standards that can encode not only the alphabetic languages, but also ideographic scripts like Chinese, Japanese, and Korean are introduced. Clearly, 8 bits are not sufficient to represent even a fraction of the “characters” in those scripts. Multi-byte elec-

tronic coding standards have been developed to serve this need—in particular, “16-bit” Unicode [165], which is a subset of the multi-byte ISO 10646 [69, 70]. Unicode will likely become the base encoding of most operating systems in the near future. Moreover, Unicode lies at the very heart of the XML [26] meta-language, on which all recently developed markup languages of the Internet are based. Thus, the integrity of electronic documents and data—structural as well as content-wise—can be fully guaranteed. L<sub>A</sub>T<sub>E</sub>X supports a restricted version of Unicode’s UTF-8 representation through the `inputenc` option `utf8` discussed in Section 7.5.2.

At its Portland, Oregon, meeting in July 1992, TUG’s Technical Council set up the Technical Working Group on Multiple Language Coordination (TWGMLC), chaired by Yannis Haralambous. This group was charged with promoting and co-ordinating the standardization and development of T<sub>E</sub>X-related software adapted to different languages. Its aim was to produce for each language or group of languages a package that would facilitate typesetting. Such a package should contain details about fonts, input conventions, hyphenation patterns, a L<sub>A</sub>T<sub>E</sub>X option file compatible with the `babel` concept (see Section 9.1.3), possibly a preprocessor, and, of course, documentation in English and the target language.

### 9.1.1 Language-related aspects of typesetting

When thinking about supporting typesetting documents in languages other than English, a number of aspects that need to be dealt with come to mind.

First and foremost is the fact that other languages have different rules for hyphenation, something that T<sub>E</sub>X accommodates through its support for multiple hyphenation patterns. In some languages, however, certain letter combinations change when they appear at a hyphenation point. T<sub>E</sub>X does not support this capability “out of the box”.

Some languages need different sets of characters to be properly typeset. This issue can vary from the need for additional “accented letters” (as is the case with many European languages) to the need for a completely different alphabet (as is the case with languages using the Cyrillic or Greek alphabet). When non-European languages need to be supported, the typesetting direction might be different as well (such as right to left for Arabic and Hebrew texts) or so many characters might be needed (as is the case with the Kanji script, for instance) that T<sub>E</sub>X’s standard mechanisms cannot deal with them.

A more “subtle” problem turns up when we look at the standard document classes that each L<sub>A</sub>T<sub>E</sub>X distribution supplies. They were designed for the Anglo-American situation. A specific example where this preference interferes with supporting other languages is the start of a chapter. For some languages it is not enough to just translate the word “Chapter”; the order of the word and the denomination of the chapter needs to be changed as well, solely on the basis of grammatical rules. Where the English reader expects to see “Chapter 1”, the French reader expects to see “1<sup>er</sup> Chapitre”.

### 9.1.2 Culture-related aspects of typesetting

An even more thorny problem when faced with the need to support typesetting of many languages is the fact that typesetting rules differ, even between countries that use the same language. For instance, hyphenation rules differ between British English and American English. Translations of English words might vary between countries, just as they do for the German spoken in Germany and the German spoken (and written) in Austria.

Typographic rules may differ between countries, too. No worldwide standard tells us how nested lists should be typeset; on the contrary, their appearance may differ for different languages, or countries, or even printing houses. With these aspects we enter the somewhat fuzzy area comprising the boundary between language aspects of typesetting and cultural aspects of typesetting. It is not clear where that boundary lies. When implementing support for typesetting documents written in a specific language, this difference needs to be taken into account. The language-related aspects can be supported on a general level, but the cultural aspects are more often than not better (or more easily) handled by creating specific document classes.

### 9.1.3 Babel—L<sup>A</sup>T<sub>E</sub>X speaks multiple languages

The L<sup>A</sup>T<sub>E</sub>X distribution contains a few standard document classes that are used by most users. These classes (article, report, book, and letter) have a certain American look and feel, which not everyone likes. Moreover, the language-dependent strings, such as “Chapter” and “Table of Contents” (see Table 9.2 on page 547 for a list of commands holding language-dependent strings), come out in English by default.

The *babel* package developed by Johannes Braams [25] provides a set of options that allow the user to choose the language(s) in which the document will be typeset. It has the following characteristics:

- Multiple languages can be used simultaneously.
- The hyphenation patterns, which are loaded when INIT<sub>E</sub>X is run to produce the L<sup>A</sup>T<sub>E</sub>X format, can be defined dynamically via an external file.
- Translations for the language-dependent strings and commands for facilitating text input are provided for more than 20 languages (see Table 9.1 on the facing page).

In the next section we describe the user interface of the *babel* system. We then discuss the additional commands for various languages and describe the support for typesetting languages using non-Latin alphabets. Finally, we discuss ways to tailor *babel* to your needs and go into some detail about the structure of the *language definition files* (.ldf) that implement the language-specific commands in *babel*. Throughout the sections, examples illustrate the use of various languages supported by *babel*.

<i>Language</i>	<i>Option</i>	<i>Language</i>	<i>Option</i>
Bahasa	bahasa	Icelandic	icelandic
Basque	basque	Interlingua	interlingua
Breton	breton	Irish Gaelic	irish
Bulgarian	bulgarian	Italian	italian
Catalan	catalan	Latin	latin
Croatian	croatian	Lower Sorbian	lowersorbian
Czech	czech	North Sami	samin
Danish	danish	Norwegian	norsk, nynorsk
Dutch	dutch, afrikaans	Polish	polish
English	english, USenglish, ( <i>american</i> , <i>canadian</i> ), UKenglish ( <i>british</i> ), <i>australian</i> ( <i>newzealand</i> )	Portuguese	portuges ( <i>portuguese</i> ), <i>brazilian</i> ( <i>brazil</i> )
Esperanto	esperanto	Romanian	romanian
Estonian	estonian	Russian	russian
Finnish	finnish	Scottish Gaelic	scottish
French	french ( <i>frenchb</i> , <i>francais</i> , <i>acadian</i> , <i>canadien</i> )	Serbian	serbian
Galician	galician	Slovakian	slovak
German	german ( <i>germanb</i> ), ngerman, <i>austrian</i> , <i>naustrian</i>	Slovenian	slovene
Greek	greek, polutonikogreek	Spanish	spanish
Hebrew	hebrew	Swedish	swedish
Hungarian	magyar ( <i>hungarian</i> )	Turkish	turkish
		Ukrainian	ukrainian
		Upper Sorbian	uppersorbian
		Welsh	welsh

*Options typeset in parentheses are alias names for the preceding option.*

*Other options for a single language typically differ in hyphenation rules, date handling, or language-dependent strings.*

*The option english combines American hyphenation patterns with a British date format.*

Table 9.1: Language options supported by the `babel` system

## 9.2 The babel user interface

Any language that you use in your document should be declared as an option when loading the `babel` package. Alternatively, because the language(s) in which a document is written constitute a global characteristic of the document, the languages can be indicated as *global options* on the `\documentclass` command. This strategy makes them available to any package that changes behavior depending on the language settings of the document. Currently supported options are enumerated in Table 9.1. For example, the following declaration prepares for typesetting

in the languages German (option `ngerman` for new hyphenation rules) and Italian (option `italian`):

```
\usepackage[ngerman,italian]{babel}
```

The last language appearing on the `\usepackage` command line will be the default language used at the beginning of the document. In the above example, the language-dependent strings, the hyphenation patterns (if they were loaded for the given language when the L<sup>A</sup>T<sub>E</sub>X format was generated with INIT<sub>E</sub>X; see the discussion on page 580), and possibly the interpretation of certain language-dependent commands (such as the date) will be for Italian from the beginning of the document up to the point where you choose a different language.

If one decides to make `ngerman` and `italian` global options, then other packages can also detect their presence. For example, the following code lets the package `variorref` (described in Section 2.4.2 on page 68) detect and use the options specified on the `\documentclass` command:

```
\documentclass[ngerman,italian]{article}
\usepackage{babel}
\usepackage{variorref}
```

If you use more than one language in your document and you want to define your own language-dependent strings for the `variorref` commands, you should use the methods described in Section 9.5 on page 579 and not those discussed in Section 2.4.2.

### 9.2.1 Setting or getting the current language

Within a document it is possible to change the current language in several ways. For example, you can change all language-related settings including translations for strings like “Chapter”, the typesetting conventions, and the set-up for short-hand commands. Alternatively, you can keep the translations unchanged but modify everything else (e.g., when typesetting short texts in a foreign language within the main text). Finally, you can change only the hyphenation rules.

```
\selectlanguage{language} \begin{otherlanguage}{language}
```

A change to all language-related settings is implemented via the command `\selectlanguage`. For instance, if you want to switch to German, you would use the command `\selectlanguage{german}`. The process is similar for switching to other languages. Each language must have been declared previously as a language option in the preamble as explained earlier. The `\selectlanguage` command calls the macros defined in the language definition file (see Section 9.5) and activates the special definitions for the language in question. It also updates the setting of T<sub>E</sub>X’s `\language` primitive used for hyphenation.

The environment `otherlanguage` provides the same functionality as the `\selectlanguage` declaration, except that the language change is local to the environment. For mixing left-to-right typesetting with right-to-left typesetting, the use of this environment is a prerequisite. The argument *language* is the language one wants to switch to.

```
\foreignlanguage{language}{phrase} \begin{otherlanguage*}{language}
```

The command `\foreignlanguage` typesets *phrase* according to the rules of *language*. It switches only the extra definitions and the hyphenation rules for the language, *not* the names and dates. Its environment equivalent is `otherlanguage*`.

The expansion of fixed document element names depends on the language, e.g., in English we have “References” or “Chapter”. Auf Deutsch ergibt sich “Literatur” oder “Kapitel”. Voici en français : “Références” ou “Chapitre”. But in short phrases “Références” does not change!

9-2-1

```
\usepackage[german,french,english]{babel}
\raggedright

The expansion of fixed document element names
depends on the language, e.g., in English
we have “\refname” or “\chaptername”. \par
\selectlanguage{german} Auf Deutsch ergibt sich
“\refname” oder “\chaptername”. \par
\begin{otherlanguage*}{french} Voici en
fran\c cais: “\refname” ou “\chaptername”.
\par\foreignlanguage{english}{But in short
phrases “\refname” does not change!}
\end{otherlanguage*}
```

```
\begin{hyphenrules}{language}
```

For the contents of the environment `hyphenrules`, *only* the hyphenation rules of *language* to be used are changed; `\languagename` and all other settings remain unchanged. When no hyphenation rules for *language* are loaded into the format, the environment has no effect.

As a special application, this environment can be used to prevent hyphenation altogether, provided that in `language.dat` the “language” `nohyphenation` is defined (by loading `zerohyph.tex`, as explained in Section 9.5.1 on page 580).

This text shows the effect of hyphenation.  
This text shows the effect of hyphenation.

9-2-2

```
\usepackage[english]{babel}
\begin{minipage}{5cm}
This text shows the effect of hyphenation.\par
\begin{hyphenrules}{nohyphenation}
This text shows the effect of hyphenation.
\end{hyphenrules}
\end{minipage}
```

Note that this approach works even if the “language” `nohyphenation` is not specified as an option to the `babel` package.

If more than one language is used, it might be necessary to know which language is active at a specific point in the document. This can be checked by a call to `\iflanguage`:

```
\iflanguage{language}{true-clause}{false-clause}
```

The first argument in this syntax, *language*, is the name of a language, which is first checked to see whether it corresponds to a language declared to `babel`. If the *language* is known, the command compares it with the current language. If they are the same, the commands specified in the *true-clause* are executed; otherwise, the commands specified in the third argument, *false-clause*, are executed.

This step is actually carried out by comparing the `\l@{language}` commands that point to the hyphenation patterns used for the two languages (see Section 9.5.1 on page 580). Thus, two “languages” are considered identical if they share the same patterns (e.g., dialects<sup>1</sup> of a language such as `austrian`), especially with languages for which no patterns are loaded.

English and Austrian use different while German and Austrian use the same hyphenation patterns.

```
\usepackage[german,english]{babel}
English and Austrian use \iflanguage{austrian}{the same}{different}
\foreignlanguage{german}{while German}
      and Austrian use \iflanguage{austrian}{the same}{different}
hyphenation patterns.
```

9-2-3

`\languagename`

The control sequence `\languagename` contains the name of the current language.

- (1) The language is `english`.
- (2) The language is `german`.
- (3) The language is `french`.
- (4) The language is `english`.
- (5) Pas en français.
- (6) The language is `german`.

```
\usepackage[german,french,english]{babel}
\par(1) The language is \languagename.
\par(2) \selectlanguage{german}%
The language is \languagename.
\par(3) \begin{otherlanguage}{french}
      The language is \languagename.
      \end{otherlanguage}
\par(4) \foreignlanguage{english}{%
      The language is \languagename.}
\par(5) \iflanguage{french}{En fran\c cais.}%
      {Pas en fran\c cais.}
\par(6) The language is \languagename.
```

9-2-4

Most document classes available in a L<sup>A</sup>T<sub>E</sub>X installation define a number of commands that are used to store the various language-dependent strings. Table 9.2 on the facing page presents an overview of these commands, together with their default text strings.

<sup>1</sup>Only in the implementation in `babel!` Some languages are implemented as “dialects” of the others for TeXnical reasons; no discrimination is intended.

<i>Command</i>	<i>English String</i>	<i>Command</i>	<i>English String</i>
\abstractname	Abstract	\indexname	Index
\alsofname	see also	\listfigurename	List of Figures
\appendixname	Appendix	\listtablename	List of Tables
\bibname	Bibliography	\pagename	Page
\ccname	cc	\partname	Part
\chaptername	Chapter	\prefacename	Preface
\contentsname	Contents	\proofname	Proof
\enclname	encl	\refname	References
\figurename	Figure	\seename	see
\glossaryname	Glossary	\tablename	Table
\headtoname	To (letter class)		

Table 9.2: Language-dependent strings in babel (English defaults)

### 9.2.2 Handling shorthands

For authors who write in languages other than English, it is sometimes awkward to type the input needed to produce the letters of their languages in the final document. More often than not, they need letters with accents above or below—sometimes even more than one accent. When you need to produce such glyphs and do not have the ability to use 8-bit input, but rather have to rely on 7-bit input encodings, an easier way to type those instructions would be welcome. For this reason (among others, as will be discussed later), babel supports the concept of “shorthands”. A “shorthand” is a one- or two-character sequence, the first character of which introduces the shorthand and is called the “shorthand character”. For a two-character shorthand, the second character specifies the behavior of the shorthand.

Babel knows about three kinds of shorthands—those defined by “the system”, “the language”, and “the user”. A system-defined shorthand sequence can be overridden by a shorthand sequence defined as part of the support for a specific language; a language-defined shorthand sequence can be overridden by a user-defined one.

#### Document-level commands for shorthands

This section describes the shorthand commands that can be used in the document and various aspects of the shorthand concept. Language-level or system-level shorthands are declared in language definition files; see Section 9.5 on page 579.

```
\useshorthands{char}
```

The command `\useshorthands` initiates the definition of user-defined shorthand sequences. The argument *char* is the character that starts these shorthands.

```
\defineshorthand{charseq}{expansion}
```

The command `\defineshorthand` defines a shorthand. Its first argument, *charseq*, is a one- or two-character sequence; the second argument, *expansion*, is the code to which the shorthand should expand.

```
\aliasshorthand{char1}{char2}
```

The command `\aliasshorthand` lets you use another character, *char2*, to perform the same functions as the default shorthand character, *char1*. For instance, if you prefer to use the character `|` instead of `"`, you can enter `\aliasshorthand{"|"}{|}`.

```
\usepackage[english]{babel}      \useshorthands{}  
\defineshorthand{"a"}{\{a\}}    \defineshorthand{"i"}{\{\i\}}  
\aliasshorthand{"|"}{|}
```

This shows the use and effect of "a: ä and "i: ii.

This shows the use and effect of |a: ä and |i: ii.

This shows the use and effect of `\verb="a=: "a` and `\verb="i=: "i.`

This shows the use and effect of `\verb=|a=: |a` and `\verb=|i=: |i.` 9-2-5

```
\languageshorthands{language}
```

The command `\languageshorthands` is used to switch between shorthands for the *language* specified as an argument. The *language* must have been declared to babel for the current document. When switching languages, the language definition files usually issue this command for the language in question. For example, the file `frenchb.lfd` contains the following command:

```
\languageshorthands{french}
```

Sometimes it is necessary to temporarily switch off the shorthand action of a given character because it needs to be used in a different way.

```
\shorthandon{chars} \shorthandoff{chars}
```

The command `\shorthandoff` sets the `\catcode` for each of the characters in its argument *chars* to “other” (12). Conversely, the command `\shorthandon` sets the `\catcode` to “active” (13) for its argument *chars*. Both commands only act on “known” shorthand characters. If a character is not known to be a shorthand character, its category code will be left unchanged.

For instance, the language definition file `german.lfd` defines two commands, `\mdqoff` and `\mdqon`, that turn the shorthand action of the character `"` off and on, respectively. They are defined as follows:

```
\newcommand\mdqon{\shorthandon{""}  
\newcommand\mdqoff{\shorthandoff{""}}
```

The language definition file for French (`frenchb.ldf`) makes the “double” punctuation characters “?”, “!”, “:”, and “;” active. One can eliminate this behavior by specifying each as an argument to a `\shorthandoff` command. This step is necessary with certain packages, where the same characters have a special meaning. Below is an example with the `xy` package, where the use of “;” and “?” as shorthand characters is turned off inside `xy`’s `xy` environment [57, Chapter 5], because these characters have a functional meaning there.

```
\usepackage{xy}    \usepackage[french]{babel}
Voici un exemple avec xypic : \[ \shorthandoff{;?}
Voici un exemple avec xypic : \begin{xy} (0,0)*{\bullet}, (0,0) ; (10,0),
                                **\dir {-} ?>* \dir {>} , (12,0)*{x}, \end{xy}
                                \]
•————→ x Quelle belle fl\`eche ! .
```

9-2-6

Quelle belle fl\`eche !

### 9.2.3 Language attributes

Sometimes the support for language-dependent typesetting needs to be tailored for different situations. In such a case it is possible to define attributes for the particular language. Two examples of the use of attributes can be found in the support for typesetting of Latin texts. When the attribute `medieval` is selected, certain document element names are spelled differently; also, the letters “u” and “V” are defined to be a lowercase and uppercase pair. The attribute `withprosodicmarks` can be used when typesetting grammars, dictionaries, teaching texts, and the like, where prosodic marks are important for providing complete information on the words or the verses. This attribute makes special shorthands available for breve and macron accents that may interfere with other packages.

```
\languageattribute{language}{langattrs}
```

The command `\languageattribute` declares which attributes are to be used for a given language. It must be used in the preamble of the document following the command `\usepackage[...]{babel}` that loads the `babel` package. The command takes two arguments: *language* is the name of a language, and *langattrs* is a comma-separated list of attributes to be used for that language. The command checks whether the *language* is known in the current document and whether the attribute(s) are known for this language.

For instance, `babel` has two variants for the Greek language: `monotoniko` (one-accent), the default, and `polotoniko` (multi-accent). To select the `polotoniko` variant, one must specify it in the document preamble, using the command `\languageattribute`. The following two examples illustrate the difference.

The Greek word for ‘Index’ \usepackage[greek,english]{babel}

9-2-7

is Εὐρετήριο.

The Greek word for ‘Index’ is \selectlanguage{greek}\indexname.

With the `polotoniko` attribute we get a different result:

The Greek word for ‘Index’ is Εὐρετήριο. 9-2-8

```
\usepackage[greek,english]{babel}
\languageattribute{greek}{polotoniko}
The Greek word for ‘Index’ is \selectlanguage{greek}\indexname.
```

## 9.3 User commands provided by language options

This section gives a general overview of the features typically offered by the various language options. It includes translations of language-dependent strings and a survey of typical shorthands intended to ease language-specific document content or to solve language-specific typesetting requirements. Some language options define additional commands to produce special date formats or numbers in a certain style. Also discussed are layout modifications as undertaken for French and Hebrew as well as the interfaces for dealing with different scripts (e.g., Latin and Cyrillic) in the same document.

### 9.3.1 Translations

As discussed earlier, `babel` provides translations for document element names that L<sup>A</sup>T<sub>E</sub>X uses in its document classes. The English versions of these strings are shown in Table 9.2 on page 547. Table 9.3 on page 551 shows the translations for a number of languages, some of them not using the normal Latin script.

Apart from the translated strings in Table 9.3, the language definition files supply alternative versions of the command `\today`, as shown in the following example.

In England the date is ‘29th February 2004’, while in Bulgaria it is ‘29 февруари 2004 г.’. In Catalonia they write ‘29 de febrer de 2004’.	<code>\usepackage[catalan,bulgarian,british]{babel}</code> <code>\raggedright</code> In England the date is ‘\today’, while in Bulgaria it is ‘{\selectlanguage{bulgarian}\today}’. In Catalonia they write ‘{\selectlanguage{catalan}\today}’.	9-3-2
---	---	-------

### 9.3.2 Available shorthands

Many of the language definition files provide shorthands. Some are meant to ease typing, whereas others provide quite extensive trickery to achieve special effects. You might not be aware of it, but L<sup>A</sup>T<sub>E</sub>X itself defines a shorthand (although it is not called by that name) that you probably use quite often: the character tilde (~), which is used to enter a “nonbreakable” space.

A number of shorthand definitions deal with “accented characters”. They were invented in the days when T<sub>E</sub>X did not yet support 8-bit input or 8-bit hyphenation

<i>Command</i>	<i>French</i>	<i>Greek</i>	<i>Polish</i>	<i>Russian</i>
\abstractname	Résumé	Περίληψη	Streszczenie	Аннотация
\alsofname	voir aussi	βλέπε επίσης	Porównaj także	см. также
\appendixname	Annexe	Παράρτημα	Dodatek	Приложение
\bibname	Bibliographie	Βιβλιογραφία	Bibliografia	Литература
\ccname	Copie à	Κοινοποίηση	Kopie:	исх.
\chaptername	Chapitre	Κεφάλαιο	Rozdział	Глава
\contentsname	Table des matières	Περιεχόμενα	Spis treści	Содержание
\enclname	P. J.	Συνημμένα	Załącznik	вкл.
\figurename	FIG.	Σχήμα	Rysunek	Рис.
\glossaryname	Glossaire	Γλωσσάριο	Glossary	Glossary
\headtoname		Πρός	Do	вх.
\indexname	Index	Ευρετήριο	Indeks	Предметный указатель
\listfigurename	Table des figures	Κατάλογος Σχημάτων	Spis rysunków	Список иллюстраций
\listtablename	Liste des tableaux	Κατάλογος Πινάκων	Spis tablic	Список таблиц
\pagename	page	Σελίδα	Strona	с.
\partname	Deuxième partie	Μέρος	Część	Часть
\prefacename	Préface	Πρόλογος	Przedmowa	Предисловие
\proofname	Démonstration	Απόδειξη	Dowód	Доказательство
\refname	Références	Αναφορές	Literatura	Список литературы
\seename	voir	βλέπε	Porównaj	см.
\tablename	TAB.	Πίνακας	Tablica	Таблица

9-3-1 In French \partname also generates the part number as a word, e.g., “Première, Deuxième, ...”

Table 9.3: Language-dependent strings in babel (French, Greek, Polish, and Russian)

patterns. When proper 8-bit hyphenation patterns are available, it is normally better to apply those and to use the `inputenc` package to select the proper input encoding (see Section 7.1.2 on page 329). However, if special processing needs to take place when an accented character appears next to a hyphenation point (as is the case for the Dutch hyphenation rules), the use of shorthands cannot be circumvented.<sup>1</sup>

### The double quote

The most popular character to be used as a shorthand character is the double quote character ("). This character is used in this way for Basque, Bulgarian, Catalan, Danish, Dutch, Estonian, Finnish, Galician, German, Icelandic, Italian, Latin, Norwegian, Polish, Portuguese, Russian, Serbian, Slovenian, Spanish, Swedish, Ukrainian, and Upper Sorbian. To describe all uses of the double quote

<sup>1</sup>This statement is true only if the underlying formatter is TeX. Omega, for example, provides additional functionality so that such cases can be handled automatically.

character as a shorthand character would go too far. Instead, it is recommended that you check the documentation that comes with the `babel` package for each language if you want to know the details. What can be said here is that its uses fall into a number of categories, each of which deserves a description and a few examples.

**Insert accented letters** For a number of languages shorthands have been created to facilitate typing accented characters. With the availability of 8-bit input and output encodings this usage might seem to have become obsolete, but this is not true for all cases. For the Dutch language, for instance, an accent needs to be removed when the hyphenation point is next to the accented letter.

Den Koning van Hispanië̄n heb ik altijd ge-eerd! Den Koning van Hispanië̄n heb ik altijd ge-eerd!

```
\usepackage[dutch]{babel}
```

```
Den Koning van Hispani"en heb ik altijd ge"eerd!
Den Koning van Hispani"en heb ik altijd ge"eerd!
```

9-3-3

**Insert special characters** In the Catalan language a special glyph, the “geminated l”, is needed for proper typesetting [167].

```
\usepackage[catalan,english]{babel}
```

The “geminated l” appears in words such as intel·ligència, illusió.

The “geminated-l” appears in words such as

```
\foreignlanguage{catalan}{inte"lig\`encia, i"lusi\`o}.
```

9-3-4

This character can also be typeset by using the commands `\lgem` and `\Lgem` or through the combinations “`\l.`” and “`\L.`” once `catalan` is selected.

**Insert special quoting characters** By default, L<sup>A</sup>T<sub>E</sub>X supports single and double quotes: ‘quoted text’ and “quoted text”. This support is not desirable in European languages. Many have their own conventions and more often than not require different characters for this purpose. For example, in Dutch traditional typesetting the opening quote should be placed on the baseline, in German typesetting the closing quote is reversed, and French typesetting requires guillemets. For Icelandic typesetting the guillemets are used as well, but the other way around—that is, pointing “inward” instead of “outward” (a convention also sometimes used in German typography).

English “quoted text” has quotes different from Dutch „quoted text” or German „quoted text“ or French « quoted text ».

```
\usepackage[dutch,ngerman,french,english]{babel}
English "quoted text" has quotes different from
\selectlanguage{dutch}Dutch "quoted text" or
\selectlanguage{ngerman}German "quoted text" or
\selectlanguage{french}French \og quoted text\fg.
```

9-3-5

The T1 font encoding provides the guillemets (see Table 7.32 on page 449), but its support for French typesetting relies on the commands `\og` and `\fg`. These commands not only produce the guillemets, but also provide proper spacing between them and the text they surround.

**Insert special hyphenation rules** A number of languages have specific rules about what happens to characters at a line break. For instance, in older German spelling ..ck.. is hyphenated as ..k-k.. and a triple f in a compound word is normally typeset as ff—except when hyphenated, in which case the third f reappears as shown in the example.

9-3-6

Brote baken	Farbstoff-fabrik
-------------	------------------

```
\usepackage[german]{babel}
\fbox{\parbox[t]{1,5cm}{Brote ba"cken}} \quad
\fbox{\parbox[t]{1,5cm}{Farbsto"ffabrik}}
```

**Insert special hyphenation indications** A number of shorthands are used to inform L<sup>A</sup>T<sub>E</sub>X about special situations with regard to hyphenation. For instance, in a number of languages it is sometimes necessary to prevent L<sup>A</sup>T<sub>E</sub>X from typesetting a ligature—for example, in a compound word. This goal can be achieved by inserting a small kern between the two letters that would normally form a ligature. The shorthand " | is available for this purpose in many language definitions.

9-3-7

Das deutsche Wort „Auflage“ sollte nicht so, sondern als »Auflage« gesetzt werden.	Das deutsche Wort "„Auflage“" sollte nicht so, sondern als ">Auf"   lage" < gesetzt werden.
--	---

Another popular shorthand is "- , which indicates a hyphenation point (like \-), but without supressing hyphenation in the remainder of the word:

9-3-8

minister- presi- dent	minister- president	ministerpresident
-----------------------------	------------------------	-------------------

```
\usepackage[dutch]{babel}
\fbox{\parbox[t]{1cm}{minister"-president}} \quad
\fbox{\parbox[t]{1cm}{minister\"-president}} \quad
\fbox{\parbox[t]{1cm}{ministerpresident}}
```

There is also "" (similar to "-", but does not print the -), "= (inserts an explicit hyphen with a breakpoint, allowing hyphenation in the combined words separately), and "~ (inserts an explicit hyphen without a breakpoint). The following example shows the effects of these shorthands, using the same word.

9-3-9

- |                               |                           |
|-------------------------------|---------------------------|
| 1. Gutenberg-Universität      | Gutenberg-Universität     |
| 2. GutenbergUniversität       | Gutenberg-Universität     |
| 3. GutenbergUniversität       | Gutenberg<br>Universität  |
| 4. Gutenberg-Universi-<br>tät | Gutenberg-<br>Universität |
| 5. Gutenberg-Universität      | Gutenberg-Universität     |

```
\usepackage[german]{babel}
\newcommand\present[1]{%
\fbox{\parbox[t]{31mm}{#1}}
\fbox{\parbox[t]{16mm}{#1}}
\par}
1. \present{Gutenberg-Universit"at}
2. \present{Gutenberg"-Universit"at}
3. \present{Gutenberg" "Universit"at}
4. \present{Gutenberg"=Universit"at}
5. \present{Gutenberg"~Universit"at}
```

### The tilde

For the languages Basque, Estonian, Galician, Greek, and Spanish, the tilde character is used for a different purpose than inserting an unbreakable space.

- For Estonian typography, the tilde-accent needs to be set somewhat lower than L<sup>A</sup>T<sub>E</sub>X's normal positioning.
- For Greek multi-accented typesetting, L<sup>A</sup>T<sub>E</sub>X needs to see the tilde as if it were a normal letter. This behavior is needed to make the ligatures in the Greek fonts work correctly.
- For Basque, Galician, and Spanish, the tilde is used in the shorthands `\~n` (`\tilde{n}`), `\~N` (`\tilde{N}`), and `\~-` (special dash). The construction `\~-` (as well as `\~-y` and `\~--`) produces a dash that disallows a linebreak after it. When the tilde is followed by any other character, it retains its original function as an “unbreakable space” (producing the overfull first line in the example). If such a space is needed before an “n”, this can be achieved by inserting an empty group (the second line in the example).

```
\usepackage[spanish,activeacute]{babel}
La eñe está presente en \alph y \Alph. La e~ne est'a presente en \verb|\alph|~y~\verb|\Alph|.
Como en castellano no se usan números Como en castellano~{}no se usan n'umeros romanos
romanos en minúscula, \roman se redefine en min'uscula, \verb|\roman| se redefine para que
para que los d'e en versalitas. los d'e en versalitas.
```

9-3-10

### The colon, semicolon, exclamation mark, and question mark

For the languages Breton, French, Russian, and Ukrainian, these four characters are used as shorthands to facilitate the use of correct typographic conventions. For Turkish typography, this ability is needed only for the colon and semicolon. The convention is that a little white space should precede these characters.

```
\usepackage[english,french]{babel}
En fran\c{c}ais on doit mettre un « petit
espace » devant la ponctuation double :
comme cela ! For English this is not
done: as shown here!
```

```
En fran\c{c}ais on doit mettre un \og petit espace\fg\
devant la ponctuation double: comme cela!
\selectlanguage{english}
For English this is not done: as shown here!
```

9-3-11

This white space is added automatically by default, but this setting can be changed in a configuration file. The use of the colon as a shorthand character can lead to problems with other packages or when including PostScript files in a document. In such cases it may be necessary to disable this shorthand (temporarily) by using `\shorthandoff`, as explained in Example 9-2-6 on page 549.

### The grave accent

The support for the languages Catalan and Hungarian makes it possible to use the grave accent (`) as a shorthand character.

- For Catalan this use of the grave accent character is not supported by default; one has to specify the option `activegrave` when loading `babel`. The purpose of this shorthand is to facilitate the entering of accented characters while retaining hyphenation. The shorthand can be used together with the letters a, e, o and A, E, O.

“Pàgina, Apèndix, Pròleg” are Catalan translations for “Page, Appendix, and Preface”.  
9-3-12

- For Hungarian this shorthand can be used with both uppercase and lowercase version of the characters c, d, g, l, n, s, t, and z. Its purpose is to insert discretionaries to invoke the correct behavior at hyphenation points.

locsan	locs- csan
eddzünk	edz- džünk
poggyász	pogy- gyász
Kodályya	Kodály- lya
mennyei	meny- nyei
vissza	visz- sza
pottyan	poty- tyan
rizzsel	rizs- zel

```
\usepackage[english,catalan,activegrave]{babel}
'‘P’agina, Ap’endix, Pr’oleg’’ \selectlanguage{english}
are Catalan translations for ‘‘Page, Appendix, and Preface’’.
```

### The acute accent

The support for the languages Catalan, Galician, and Spanish makes it possible to use the acute accent (`) as a shorthand character.

- For the support of Catalan typesetting, this shorthand can be used together with the vowels (a, e, i, o, u), both uppercase and lowercase. Its effect is to add

the accent and to retain hyphenation.

- For the support of Galician typesetting, this shorthand offers the same functionality as for Catalan with the addition that entering 'n will produce ñ.

“Páxina, Capítulo, Apéndice” are Galician translations for “Page, Chapter, and Appendix”. \usepackage[english,galician,activeacute]{babel} ‘‘P’axina, Cap’ítulo, Ap’endice’’ \selectlanguage{english} are Galician translations for ‘‘Page, Chapter, and Appendix’’.

9-3-14

- For the support of Spanish typesetting, this shorthand offers similar functionality as for Catalan and Galician.

The described functionality is made available when the `activeacute` option is used. This support is made optional because the acute accent has other uses in L<sup>A</sup>T<sub>E</sub>X, which will fail when this character is turned into a shorthand.

### The caret

The support for the languages Esperanto and Latin makes it possible to use the caret accent (^) as a shorthand character.

- For typesetting the Esperanto language, two accents are needed: the caret and the breve accent. The caret appears on the letters c, g, h, j, and s; the breve appears on the character u. Both accents can appear on lowercase and uppercase letters. The caret is defined as a shorthand that retains hyphenation and sets the caret accent somewhat lower on the character “h” (h). Used together with the letter u, this shorthand typesets the breve accent (^u results in ū); used together with the vertical bar, it inserts an explicit hyphen sign, allowing hyphenation in the rest of the word.

“Pa o, Capitro, Cita oj” are Esperanto translations for “Page, Chapter, and References”. \usepackage[english,esperanto]{babel} ‘‘Pa go, Capitro, Cita oj’’ \selectlanguage{english} are Esperanto translations for ‘‘Page, Chapter, and References’’.

9-3-15

- When a Latin text is being typeset and the attribute `withprosodicmarks` has been selected, the caret is defined to be a shorthand for adding a breve accent to the lowercase vowels (except the medieval ligatures æ and œ). This is done while retaining hyphenation points.

\usepackage[latin]{babel} \languageattribute{latin}{withprosodicmarks} \ProsodicMarksOn ^a ^e ^i ^o ^u

9-3-16

### The equals sign

The support for the languages Latin (with the attribute `withprosodicmarks` selected) and Turkish makes it possible to use the equals sign (=) as a shorthand character.

- When a Latin text is being typeset and the attribute `withprosodicmarks` has been selected, the equals sign is defined to be a shorthand for adding a macron accent to the lowercase vowels (except the medieval ligatures æ and œ). This is done while retaining hyphenation points.

```
9-3-17 \usepackage[latin]{babel} \languageattribute{latin}{withprosodicmarks}
         ä ē ī ö ū \ProsodicMarksOn =a =e =i =o =u
```

- When Turkish typesetting rules are to be followed, the equals sign needs to be preceded by a little white space. This is achieved automatically by turning the equals sign into a shorthand that replaces a preceding space character with a tiny amount of white space.

```
9-3-18 a=b \usepackage[english,turkish]{babel}
         a=b \selectlanguage{english} a=b \par \selectlanguage{turkish} a=b
```

The disadvantage of having the equals sign turn into a space character is that it may cause many other packages to fail, including the usage of PostScript files for graphics inclusions. Make sure that the shorthand is turned off with `\shorthandoff`.

### The greater than and less than signs

The support for the Spanish language makes it possible to use the greater than and less than signs (< and >) as shorthand characters for inserting a special quoting environment. This environment inserts different quoting characters when it is nested within itself. It supports a maximum of three levels of nested quotations. It also automatically inserts the closing quote signs when a new paragraph is started *within* a quote.

```
9-3-19 Some text with <<quoted text with a
         "nested quote" within it.
         »A second paragraph in the quota-
         tion.» \usepackage[spanish]{babel}
         Some text with <<quoted text with a <<nested quote>>
         within it.
         A second paragraph in the quotation.>>
```

Note that when characters are turned into shorthands, the ligature mechanism in the fonts no longer works for them. In the T1 font encoding, for instance, a ligature is defined for two consecutive “less than” signs that normally results in typesetting guillemets. In the example above, the nested quote shows clearly that this does not happen.

### The period

The support for the Spanish language also allows the use of the period (.) as a shorthand character in math mode. Its purpose is to control whether decimal numbers are written with the comma (\decimalcomma) or the period (\decimalpoint) as the decimal character.

1000,10	\usepackage[spanish]{babel}
1000.10	\decimalcomma \$1000.10\$ \par \decimalpoint \$1000.10\$

9-3-20

### 9.3.3 Language-specific commands

Apart from the translations and shorthands discussed above, some language definition files provide extra commands. Some of these are meant to facilitate the production of documents that conform to the appropriate typesetting rules. Others provide extra functionality not available by default in L<sup>A</sup>T<sub>E</sub>X. A number of these commands are described in this section.

#### Formatting dates

For some languages more than one format is used for representing dates. In these cases extra commands are provided to produce a date in different formats. In the Bulgarian tradition months are indicated using uppercase Roman numerals; for such dates the command \todayRoman is available.

29 февруари 2004 г.	\usepackage[bulgarian]{babel}
29. II. 2004 г.	\today \par \todayRoman

9-3-21

When writing in the Esperanto language two slightly different ways of representing the date are provided by the commands \hodiau and \hodiaun.

29–a de februaro, 2004	
la 29–a de februaro, 2004	\usepackage[esperanto]{babel}
la 29–an de februaro, 2004	\today \par \hodiau \par \hodiaun

9-3-22

When producing a document in the Greek language the date can also be represented with Greek numerals instead of Arabic numerals. For this purpose the command \Grtoday is made available.

29 Φεβρουαρίου 2004	\usepackage[greek]{babel}
ΚΘ' Φεβρουαρίου ΒΔ'	\today \par \Grtoday

9-3-23

The support for typesetting Hebrew texts offers the command \hebdate to translate any Gregorian date, given as “day, month, year”, into a Gregorian date in Hebrew. The command \hebday replaces L<sup>A</sup>T<sub>E</sub>X’s normal \today. When you want to produce “normal” Hebrew dates, you need to use the package hebcal, which

provides the command `\Hebrewtoday`. When it is used *outside* the Hebrew environment it produces the Hebrew date in English.

<span style="border: 1px solid black; padding: 2px;">9-3-24</span> 29 בפברואר 2004 ז' באדר, תשס"ד 29th February 2004: Adar 7, 5764 8 בנובמבר 1997	<pre>\usepackage[english,hebrew]{babel} \usepackage{hebcal} \hebdays \par \Hebrewtoday \par \selectlanguage{english} \today: \Hebrewtoday \selectlanguage{hebrew} \hebdays{8}{11}{1997}</pre>
---	---

The support for the Hungarian language provides the command `\ontoday` to produce a date format used in expressions such as “on February 10th”.

For the Upper and Lower Sorbian languages two different sets of month names are employed. By default, the support for these languages produces “new-style” dates, but “old-style” dates can be produced as well. The “old-style” date format for the Lower Sorbian language can be selected with the command `\olddatelsorbian`; `\newdatelsorbian` switches (back) to the modern form. For Upper Sorbian similar commands are available, as shown in the example.

<span style="border: 1px solid black; padding: 2px;">9-3-25</span> 29. februara 2004 29. małego rožka 2004 29. februara 2004 29. małego róžka 2004	<pre>\usepackage[usorbian,lsorbian]{babel} \newdatelsorbian \today \par \olddatelsorbian \today \par \newdateusorbian \today \par \olddateusorbian \today</pre>
--	---

In Swedish documents it is customary to represent dates with just numbers. Such dates can occur in two forms: YYYY-MM-DD and DD/MM YYYY. The command `\datesynd` changes the definition of the command `\today` to produce dates in the first numerical form; the command `\datesdmy` changes the definition of the command `\today` to produce dates in the second numerical format.

Default date format: 29 februari 2004 <code>\datesynd</code> gives: 2004-02-29 <code>\datesdmy</code> gives: 29/2 2004	<pre>\usepackage[swedish]{babel} Default date format: \today \\ \verb \datesynd  gives: \datesynd \today \\ \verb \datesdmy  gives: \datesdmy \today</pre>
--	--

### Numbering

The support for certain languages provides additional commands for representing numbers by letters. L<sup>A</sup>T<sub>E</sub>X provides the commands `\alph` and `\Alph` for this purpose. For the Esperanto language the commands `\esper` and `\Esper` are provided. The support for the Greek language changes the definition of `\alph` and `\Alph` to produce Greek letters while the support for the Bulgarian language changes them to produce Cyrillic letters. The support for the Russian and Ukrainian languages provides the commands `\asbuk` and `\Asbuk` as alternatives to the L<sup>A</sup>T<sub>E</sub>X commands.

	<i>default</i>	Esperanto	Greek	Russian	Bulgarian	Hebrew
Value	\alph\Alpha	\esper\Esper	\alph\Alpha	\asbuk\Asbuk	\alph\Alpha	\alph\Alpha\Alphfinal
1	a	A	α'	Α'	а	א
2	b	B	β'	Β'	б	ב
3	c	C	γ'	Γ'	в	ג
4	d	D	δ'	Δ'	г	ד
5	e	E	ε'	Ε'	д	ה
6	f	F	ϝ'	ϝ'	е	ו
7	g	G	ζ'	Ζ'	ж	ז
8	h	H	η'	Η'	з	ח
9	i	I	ϒ'	Θ'	и	ט
10	j	J	ι'	Ι'	κ	י
11	k	K	Ϟ'	ΙΑ'	л	א
12	l	L	ι'	ΙΒ'	м	ב
13	m	M	ιγ'	ΙΓ'	н	ג
14	n	N	ιδ'	ΙΔ'	о	ד
15	o	O	ιε'	ΙΕ'	п	ט
16	p	P	ιι'	Ιι'	р	ז
17	q	Q	ιζ'	ΙΖ'	с	י
18	r	R	ιη'	ΙΗ'	т	ח
19	s	S	ιθ'	ΙΘ'	у	ט
20	t	T	ιχ'	Κ'	ф	ט
21	u	U	ια'	ΚΑ'	х	א
22	v	V	ιβ'	ΚΒ'	ц	ב
23	w	W	ιγ'	ΚГ'	ч	ג
24	x	X	ιδ'	ΚΔ'	ш	ד
25	y	Y	ιε'	ΚΕ'	щ	ה
26	z	Z	ιئ'	Κئ'	э	ו
27	-	-	ιη'	КХ'	ю	י
28	-	-	ιθ'	КΘ'	я	ת
29	-	-	-	-	-	ט
30	-	-	-	λ'	ל	ל
40	-	-	-	μ'	מ	ס
50	-	-	-	ν'	נ	ר
100	-	-	-	ρ'	ק	ק
250	-	-	σν'	Σν'	-	רָן
500	-	-	ψ'	Φ'	-	תְּקֵן

[9-3-27]

Table 9.4: Different methods for representing numbers by letters

For Hebrew typesetting the \alph command is changed to produce Hebrew letter sequences using the “Gimatria” scheme. As there are no uppercase letters \Alpha produces the same letter sequences but adds apostrophes. In addition, an extra command, \Alphfinal, generates Hebrew letters with apostrophes and final letter forms, a variant needed for Hebrew year designators. Table 9.4 compares the various numbering schemes.

In French typesetting, numbers should be typeset following different rules than those employed in English typesetting. Namely, instead of separating thousands with a comma, a space should be used. The command `\nombre` is provided for this purpose. It can also be used *outside* the French language environment, where it will typeset numbers according to the English rules. The command `\nombre` takes an optional argument, which can be used to replace the default decimal separator (stored in `\decimalsep`). This feature can be useful in combination with the package `dcolumn` (see Section 5.7.2), in which you have to use the optional argument to achieve correct alignment.

```
\usepackage[english,french]{babel} \usepackage{dcolumn}
\newcolumntype{d}[D{,}{\decimalsep}{-1}] % align at explicit ','
                                         % but output \decimalsep
\begin{tabular}{|d|} \hline
 12,34567 \\
 12,345 67 \\
 12,345 67 \\
 9 876 543,21 \\
\end{tabular}
\par\vspace{1cm} \selectlanguage{english} % change language
\begin{tabular}{|d|} \hline
 12.345,67 \\
 9,876,543.21 \\
\end{tabular}
```

In Greece an alternative way of writing numbers exists. It is based on using letters to denote number ranges. This system was used in official publications at the end of the 19th century and the beginning of the 20th century. At present most Greeks use it for small numbers. The knowledge of how to write numbers larger than 20 or 30 is not very widespread, being primarily used by the Eastern Orthodox Church and scholars. They employ this approach to denote numbers up to 999999. This system works as follows:

- Only numbers greater than 0 can be expressed.
- For the units 1 through 9 (inclusive), the letters alpha, beta, gamma, delta, epsilon, stigma, zeta, eta, and theta are used, followed by a mark similar to the mathematical symbol “prime”, called the “numeric mark”. Because the letter stigma is not always part of the available font, it is often replaced by the first two letters of its name as an alternative. In the babel implementation the letter stigma is produced, rather than the digraph sigma tau.
- For the tens 10 through 90 (inclusive), the letters iota, kappa, lambda, mu, nu, xi, omikron, pi, and qoppa are used, again followed by the numeric mark. The qoppa that appears in Greek numerals has a distinct zig-zag form that is quite different from the normal qoppa, which resembles the Latin “q”.
- For the hundreds 100 through 900 (inclusive), the letters rho, sigma, tau, uppsilon, phi, chi, psi, omega, and sampi are used, also followed by the numeric mark.

- Using these rules any number between 1 and 999 can be expressed by a group of letters denoting the hundreds, tens, and units, followed by *one* numeric mark.
- For the number range 1000 through 999000 (inclusive), the digits denoting multiples of a thousand are expressed by the same letters as above, this time with a numeric mark in front of this letter group. This mark is rotated 180 degrees and placed *under* the baseline. As can be seen in the example below, when two letter-groups are combined, *both* numeric marks are used.

123456 in Greek notation: ρχγυν' \usepackage[english,greek]{babel}  
 987654 in Greek notation: ΗΠΖΞΝΔ' \newcommand\eng[1]{\foreignlanguage{english}{#1}}  
 123456 \eng{in Greek notation:} \greeknumeral{123456} \par  
 987654 \eng{in Greek notation:} \Greeknumeral{987654}

9-3-29

In ancient Greece yet another numbering system was used, which closely resembles the Roman one in that it employs letters to denote important numbers. Multiple occurrences of a letter denote a multiple of the “important” number; for example, the letter I denotes 1, so III denotes 3. Here are the basic digits used in the Athenian numbering system:

- I denotes the number one (1).
- II denotes the number five (5).
- Δ denotes the number ten (10).
- H denotes the number one hundred (100).
- X denotes the number one thousand (1000).
- M denotes the number ten thousand (10000).

Moreover, the letters Δ, H, X, and M, when placed under the letter Π, denote five times their original value; for example, the symbol Π denotes the number 5000, and the symbol ΠΔ denotes the number 50. Note that the numbering system does not provide negative numerals or a symbol for zero.

The Athenian numbering system, among others, is described in an article in Encyclopedia *Δομή*, Volume 2, seventh edition, page 280, Athens, October 2, 1975. This numbering system is supported by the package *athnum*, which comes with the *babel* system. It implements the command *\athnum*.

6284 in Athenian notation: ΠΧΗΗΠΔΔΔΙΙΙ \usepackage[english,greek]{babel}  
 \usepackage{athnum}\newcommand\eng[1]{\foreignlanguage{english}{#1}}  
 6284 \eng{in Athenian notation:} \\ \athnum{6284}

9-3-30

In Icelandic documents, numbers need to be typeset according to Icelandic rules. For this purpose the command *\tala* is provided. Like *\nombre* it takes an

optional argument, which can be used to replace the decimal separator used, such as for use with the `dcolumn` package.

```
\usepackage[english,icelandic]{babel}
  \usepackage{dcolumn} \newcolumntype{d}[1]{\decimalsep}{-1}}
  \tala{3141,592653} \par
  \foreignlanguage{english}{\tala{3141,592653}}\par \bigskip
  \begin{tabular}{|d|} \hline
    3,14 & \\
    123,456 7 & \\
    9 876,543 & \\
  \end{tabular} \hline
```

9-3-31

### Miscellaneous extras

In French typesetting it is customary to print family names in small capitals, *without* hyphenating a name. For this purpose the command `\bsc` (boxed small caps) ... *for French* is provided. Abbreviations of the French word “numéro” should be typeset according to specific rules; these have been implemented in the commands `\no` and `\No`. Finally, for certain enumerated lists the commands `\primo`, `\secundo`, `\tertio`, and `\quarto` are available when typesetting in French.

9-3-32	Leslie LAMPORT	Nº 9 1º 3º	<code>\usepackage[french]{babel}</code>
			<code>\Leslie~\bsc{Lamport} \quad \No9 \ \primo \ \tertio</code>

In the Italian language it is customary to write together the article and the following noun—for example, “nell’altezza”. To carry out the hyphenation of such ... *for Italian* constructs the character ‘ is made to behave as a normal letter.

In the Hungarian language the definite article can be either “a” or “az”, depending on the context. Especially with references and citations, it is not always ... *for Hungarian* known beforehand which form should be used. The support for the Hungarian language contains commands that know the rules dictating when a “z” should be added to the article. These commands all take an argument that determines which form of the definite article should be typeset together with that argument.

<code>\az{text}</code>	<code>\Az{text}</code>
------------------------	------------------------

These commands produce the article and the argument. The argument can be a star (as in `\az*`), in which case just the article will be typeset. The form `\Az` is intended for the start of a sentence.

<code>\aref{text}</code>	<code>\Aref{text}</code>	<code>\apageref{text}</code>	<code>\Apageref{text}</code>
--------------------------	--------------------------	------------------------------	------------------------------

The first two commands should be used instead of `a(z)\ref{label}`. When an equation is being referenced, the argument may be enclosed in parentheses instead of braces. For page references use `\apageref` (or `\Apageref`) to allow L<sup>A</sup>T<sub>E</sub>X to automatically produce the correct definite article.

<i>L<sup>A</sup>T<sub>E</sub>X</i>		<i>Serbian</i>		<i>Russian</i>	
\tan	tan	\tg	tg	\tg	tg
\cot	cot	\ctg	ctg	\ctg	ctg
\sinh	sinh	\sh	sh	\sh	sh
\cosh	cosh	\ch	ch	\ch	ch
\tanh	tanh	\th	th	\th	th
\coth	coth	\cth	cth	\cth	cth
\csc	csc			\cosec	cosec
\arcsin	arcsin	\arsh	arsh		
\arccos	arccos	\arch	arch		
\arctan	arctan	\arctg	arctg	\arctg	arctg
		\arcctg	arcctg	\arcctg	arcctg (extra)

Note that the redefinition of \th conflicts with its standard use as L<sup>I</sup>C<sup>R</sup> command for þ (thorn), therefore babel restricts this redefinition to math mode in cyrillic texts.

Table 9.5: Alternative mathematical operators for Eastern European languages

#### \acite{*text*} \Acite{*text*}

For citations the command \acite should be used. Its argument may be a list of citations, in which case the first element of the list determines which form of the article should be typeset.

In Eastern Europe a number of mathematical operators have a different appearance in equations than they do in “the Western world”. Table 9.5 shows the relevant commands for different languages. The Russian commands are also valid for Bulgarian and Ukrainian language support. The package grmath, which comes as part of the babel distribution, changes the definitions of these operators to produce abbreviations of their Greek names. The package can only be used in conjunction with the greek option of babel.

### 9.3.4 Layout considerations

Some of the language support files in the babel package provide commands for automatically changing the layout of the document. Some simply change the way L<sup>A</sup>T<sub>E</sub>X handles spaces after punctuation characters or ensure that the first paragraph that follows a section heading is indented. Others go much further.

*Spaces after punctuation characters* In *The T<sub>E</sub>Xbook* [82, pp.72–74], the concept of extra white space after punctuation characters is discussed. Good typesetting practice mandates that inter-sentence spaces behave a little differently than interword spaces with respect to shrinkage and expansion (during justification). However, this practice is not considered helpful in all cases, so for a number of languages (Breton, Bulgarian, Czech, Danish, Estonian, Finnish, French, German, Norwegian, Russian, Spanish, Turkish, and Ukrainian) this feature is switched off by calling the command \frenchspacing.

Another layout concept that is built into most L<sup>A</sup>T<sub>E</sub>X classes is the suppression of the paragraph indentation for the first paragraph that follows a section heading. Again, for some languages this behavior is wrong; the support for French, Serbo-Croatian, and Spanish changes it to have *all* paragraphs indented. In fact, you can request this behavior for any document by loading the package `indentfirst`.

The support for French (and Breton, for which support is derived from the support for the French language) takes this somewhat further to accomodate the typesetting rules used in France. It changes the general way lists are typeset by L<sup>A</sup>T<sub>E</sub>X by reducing the amount of vertical white space in them. For the `itemize` environment, it removes all vertical white space between the items and changes the appearance of the items by replacing “•” with “–”. *Layout of lists*

Some text with a list.

- item 1
- item 2

And some text following.

Some text with a list.

- item 1
  - item 2
- And some text following.

```
\usepackage[french,english]{babel}
\begin{minipage}[t]{4cm}
  Some text with a list.
  \begin{itemize}
    \item item 1
    \item item 2
  \end{itemize}
  And some text following.
\end{minipage}
\quad \selectlanguage{french}
\begin{minipage}[t]{4cm}
  Some text with a list.
  \begin{itemize}
    \item item 1
    \item item 2
  \end{itemize}
  And some text following.
\end{minipage}
```

`\FrenchLayout`    `\StandardLayout`

For documents that are typeset in more than one language, the support for French provides a way to ensure that lists have a uniform layout throughout the document, either the “French layout” or the “L<sup>A</sup>T<sub>E</sub>X layout”. This result can be achieved by using the command `\FrenchLayout` or `\StandardLayout` in the preamble of the document. Unfortunately, when your document is being typeset with something other than one of the document classes provided by standard L<sup>A</sup>T<sub>E</sub>X, or when you use extension packages such as `paralist`, such layout changes may have surprising and unwanted effects. In such cases it might be safest to use `\StandardLayout`.

`\AddThinSpaceBeforeFootnotes`    `\FrenchFootnotes`

In the French typesetting tradition, footnotes are handled differently than they are in the Anglo-American tradition. In the running text, a little white space should be added before the number or symbol that calls the footnote. This behavior is optional and can be selected by using the `\AddThinSpaceBeforeFootnotes` command in the preamble of your document. The text of the footnote can also be

*Paragraph  
indentation after  
heading*

*Layout of footnotes*

typeset according to French typesetting rules; this result is achieved by using the command \FrenchFootnotes.

Some text <sup>a</sup> . <small><sup>a</sup>with a footnote</small>	Some text <sup>a</sup> . <small><sup>a</sup>. with a footnote</small>	<pre>\usepackage[french,english]{babel} \AddThinSpaceBeforeFootnotes \begin{minipage}{70pt} Some text\footnote{with a footnote}. \end{minipage} \selectlanguage{french}\FrenchFootnotes \begin{minipage}{70pt} Some text\footnote{with a footnote}. \end{minipage}</pre>	<small>9-3-35</small>
--	--	--	-----------------------

*Layout of captions* The final layout change performed by the babel support for the French language is that the colon in captions for tables and figures is replaced with an en dash when one of the document classes of standard L<sup>A</sup>T<sub>E</sub>X is used.

*Internal commands redefined for magyar* The support for typesetting Hungarian documents goes even further: it redefines a number of internal L<sup>A</sup>T<sub>E</sub>X commands to produce correct captions for figures and tables. Using the same means, it changes the layout of section headings. The definition of the theorem environment is changed as well. As explained above, such changes may lead to unexpected and even unwanted behavior, so be careful.

*Right to left typesetting* To support typesetting Hebrew documents, even more drastic changes are needed because the Hebrew language has to be typeset from right to left. This requires the usage of a T<sub>E</sub>X extension (i.e., eT<sub>E</sub>X with a L<sup>A</sup>T<sub>E</sub>X format) to correctly typeset a Hebrew document.

### 9.3.5 Languages and font encoding

As shown in some of the earlier examples, some languages cannot be supported by, for instance, simply translating some texts and providing extra support for special hyphenation needs. Many languages require characters that are not present in L<sup>A</sup>T<sub>E</sub>X's T1 encoding. For some, just a few characters are missing and can be constructed from the available glyphs; other languages are not normally written using the Latin script. Some of these are supported by the babel system.

#### Extensions to the OT1 and T1 encodings

For some languages just a few characters are missing in the OT1 encoding and sometimes even in the T1 encoding. When the missing characters can be constructed from the available glyphs, it is relatively easy to rectify this situation. Such is the case for the Old Icelandic language. It needs a number of characters that can be represented by adding the "ogonek" to available glyphs. To access these you should use the shorthands in the next example. Note that each of these shorthands is composed of " and an 8-bit character, so use of the inputenc package is required.

```
9-3-36 \usepackage[icelandic]{babel}
           \usepackage[T1]{fontenc} \usepackage[latin1]{inputenc}
o Ó ö Ó ö É é É but: "é "É
"o "Ó "ö "ó "e "E "é "É but: "\é "\É
```

Old Icelandic may not be a language in daily use, but the Polish language certainly is. For this language the OT1 encoding is missing a few characters (note that they are all included in T1). Again the missing characters *can* be constructed, and their entry is supported with shorthands. The support for entering the letters “pointed z” and “accented z” comes in two forms, as illustrated below. The reason for this duality is historical.

```
9-3-37 \usepackage[polish]{babel}
a Á á Ā ā É ē Ł ł Ñ ñ ó ó Š ſ Ÿ Ÿ
ż Ż ż Ź Ÿ "X "X
"r "R ż Ż ż Ÿ
\usepackage[polish]{babel}
'a "A "c "C "e "E "l "n "N "o "O "s "S \par
\polishrz "r "R "z "Z "x "X \par
\polishzx "r "R "z "Z "x ."X \par
```

All such shorthands were devised when 7-bit font encodings were the norm and producing a glyph such as “A” required some internal macro processing (if it was possible at all). With today’s 8-bit fonts there is no requirement to use the shorthands. For example, with T1-encoded fonts, standard input methods may be used instead.

```
9-3-38 \usepackage[T1]{fontenc}
a Á á Ā ā É ē Ł ł Ñ ñ ó ó Š ſ Ÿ Ÿ
ż Ż ż Ź Ÿ "X "X
\k a \k Á \'c \'C \k e \k E \l{} \n \'N \'o \'O \'s \'S\par
\.z \.Z \'z \'Z
```

### Basic support for switching font encodings

In the situation where simply constructing a few extra characters to support the correct typesetting of a language does not offer a sufficient solution, switching from one font encoding to another becomes necessary. This section describes the commands provided by `babel` and its language support files for this task. Note that these commands are normally “hidden” by `babel`’s user interface.

```
\latinencoding \cyrillicencoding \hebrewencoding
```

The `babel` package uses `\latinencoding` to record the Latin encoding (OT1 or T1) used in the document. To determine which encoding is used, `babel` tests whether the encoding current at `\begin{document}` is T1; if it is not, it (perhaps wrongly) assumes OT1.

The languages that are typeset using the Cyrillic alphabet define the command `\cyrillicencoding` to store the name for the Cyrillic encoding. The command `\hebrewencoding` serves the same purpose for the Hebrew font encoding. At the time of writing no `\greekencoding` command was available, because `babel` supported only a single encoding (LGR) for Greek.

```
\textlatin{text}
```

This command typesets its argument in a font with the Latin encoding, independent of the encoding of the surrounding text.

```
\textcyrillic{text}
```

This command is (only) defined when one of the options `bulgarian`, `russian`, or `ukrainian` is used. It typesets its argument using a font in the Cyrillic encoding stored in `\cyrillicencoding`.

```
\textgreek{text} \texttol{text}
```

These commands are defined by the `greek` language option. Both typeset their arguments in a font with the Greek encoding; the command `\texttol` uses an outline font.

Declarative forms for these `\text...` commands are also available; they are called `\latintext`, `\greektext`, `\outlfamily`, and `\cyrillictext`.

### Basic support for switching typesetting directions

To support the typesetting of Hebrew texts, the direction of typesetting also needs to be changed. Several commands with different names have been defined for this purpose.

```
\sethebrew \unsethebrew
```

The command `\sethebrew` switches the typesetting direction to “right to left”, switches the font encoding to a Hebrew encoding, *and* shifts the “point of typesetting” to start from the right margin. The command `\unsethebrew` switches the typesetting direction to “left to right”, switches the font encoding to the one in use when `\sethebrew` was called, *and* shifts the “point of typesetting” to start from the left margin.

```
\R{text} \L{text}
```

The commands `\R` and `\L` should be used when a small piece of Hebrew text needs to appear in the same location relative to the surrounding text. The use of these commands is illustrated in the following example. Note the location of the second text typeset with Hebrew characters.

```
\usepackage[X2,T1]{fontenc} \usepackage[greek,russian,hebrew,english]{babel}
Some English text, \R{hebrew text}, \textgreek{Greek text},
\textcyrillic{Cyrillic text}
\sethebrew more Hebrew text\unsethebrew{}, more English text.
```

9-3-39

Some English text, פּוֹרָטְוּןְגַּעֲמָנְתָּ, Γρεεκ τεξτ, Τικτίμ-  
מִיְּהָ תְּפֵשְׁתָּ, more English text. פּוֹרָטְוּןְגַּעֲמָנְתָּ

## 9.4 Support for non-Latin alphabets

The babel distribution contains support for three non-Latin alphabets: the Cyrillic alphabet, the Greek alphabet, and the Hebrew alphabet. They are discussed in the following sections.

### 9.4.1 The Cyrillic alphabet

The Cyrillic alphabet is used by several of the Slavic languages in Eastern Europe, as well as for writing tens of languages used in the territory encompassed by the former Soviet Union. Vladimir Volovich and Werner Lemberg, together with the L<sup>A</sup>T<sub>E</sub>X team, have integrated basic support for the Cyrillic language into L<sup>A</sup>T<sub>E</sub>X. This section addresses the issues of Cyrillic fonts, the encoding interface, and their integration with babel.

Historically, support for Russian in T<sub>E</sub>X has been available from the American Mathematical Society [14]. The AMS system uses the wncyr fonts and is based on a transliteration table originally designed for Russian journal names and article titles in the journal *Mathematical Reviews*. In this journal the AMS prefers that the same character sequence in the electronic files produce either the Russian text with Russian characters or its transliteration with English characters, without any ambiguities.

However, with the spread of T<sub>E</sub>X in Russia, proper support for typesetting Russian (and later other languages written in the Cyrillic alphabet) became necessary. Over the years several 7- and 8-bit input encodings were developed, as well as many font encodings. The Cyrillic system is designed to work for any 8-bit input encoding and is able to map all of them onto a few Cyrillic font encodings, each supporting a number of languages.

#### Fonts and font encodings

For compatibility reasons, only the upper 128 characters in an 8-bit T<sub>E</sub>X font are available for new glyphs. As the number of glyphs in use in Cyrillic-based languages during the 20th century far exceeds 128, four “Cyrillic font encodings” have been defined [17]. Three of them—T2A, T2B, and T2C—satisfy the basic structural requirements of L<sup>A</sup>T<sub>E</sub>X’s T\* encodings and, therefore, can be used in multilingual documents with other languages being based on standard font encodings.<sup>1</sup>

The work on the T2\* encodings was performed by Alexander Berdnikov in collaboration with Mikhail Kolodin and Andrew Janishevsky. Vladimir Volovich provided the integration with L<sup>A</sup>T<sub>E</sub>X.

<sup>1</sup>The fourth Cyrillic encoding, X2, contains Cyrillic glyphs spread over the 256 character positions, and is thus suitable only for specific, Cyrillic-only applications. It is not discussed here.

Two other L<sup>A</sup>T<sub>E</sub>X Cyrillic font encodings exist: the 7-bit OT2 encoding developed by the American Mathematical Society, which is useful for short texts in Cyrillic, and the 8-bit LCY encoding, which is incompatible with the L<sup>A</sup>T<sub>E</sub>X's T\* encodings and, therefore, unsuitable for typesetting multilingual documents. The OT2 encoding was designed in such a way that the same source could be used to produce text either in the Cyrillic alphabet or in a transliteration.

### Cyrillic Computer Modern fonts

The default font family with L<sup>A</sup>T<sub>E</sub>X is Knuth's Computer Modern, in its 7-bit (OT1-encoded CM fonts) or 8-bit (T1-encoded EC fonts) incarnation. Olga Lapko and Andrey Khodulev developed the LH fonts, which provide glyph designs compatible with the Computer Modern font family and covering all Cyrillic font encodings. They provide the same font shapes and sizes as those available for its Latin equivalent, the EC family. These fonts are found on CTAN in the directory `fonts/cyrillic/lh`. Installation instructions appear in the file `INSTALL` in that distribution.<sup>1</sup>

A collection of hyphenation patterns for the Russian language that support the T2\* encodings, as well as other popular font encodings used for Russian typesetting (including the Omega internal encoding), are available in the `ruhyphen` distribution on CTAN (`language/hyphenation/ruhyphen`). The patterns for other Cyrillic languages should be adapted to work with the T2\* encodings.

### Using Cyrillic in your documents

Support for Cyrillic in L<sup>A</sup>T<sub>E</sub>X is based on the standard `fontenc` and `inputenc` packages, as well as on the `babel` package. For instance, one can write the following in the preamble of the document:

```
\usepackage[T2A]{fontenc}      \usepackage[koi8-r]{inputenc}
\usepackage[russian]{babel}
```

The input encoding `koi8-r` (KOI8 optimized for Russian) can be replaced by any of the following Cyrillic input encodings:

`cp855` Standard MS-DOS Cyrillic code page.

`cp866` Standard MS-DOS Russian code page. Several variants, distinguished by differences in the code positions 242–254, exist: `cp866av` (Cyrillic Alternative), `cp866mav` (Modified Alternative Variant), `cp866nav` (New Alternative Variant), and `cp866tat` (for Tatar).

`cp1251` Standard MS Windows Cyrillic code page.

<sup>1</sup>Other fonts, including Type 1 fonts, can also be used, provided that their T<sub>E</sub>X font encoding is compatible with the T2\* encodings. In particular, the CM-Super fonts cover the whole range of Cyrillic encodings; see Section 7.5.1 on page 353 for details.

`koi8-r` Standard Cyrillic code page that is widely used on UN\*X-like systems for Russian language support. Variants for Ukrainian are `koi8-u` and `koi8-ru`. An ECMA variant (ISO-IR 111 ECMA) is `isoir111`.

`iso88595` ISO standard ISO 8859-5 (also called ISO-IR 144).

`maccyr` Apple Macintosh Cyrillic code page (also known as Microsoft `cp10007`) and `macukr`, the Apple Macintosh Ukrainian code page.

`ctt, dbk, mnk, mos, ncc` Mongolian code pages.

Not all of these code pages are part of the standard `inputenc` distribution, so some may have to be obtained separately.

When more than one input encoding is used within a document, you can use the `\inputencoding` command to switch between them. To define the case of text, two standard L<sup>A</sup>T<sub>E</sub>X commands, `\MakeUppercase` and `\MakeLowercase`, can produce uppercase or lowercase, respectively. The low-level T<sub>E</sub>X `\uppercase` and `\lowercase` should never be used in L<sup>A</sup>T<sub>E</sub>X and will not work for Cyrillic.

In the previous example of a preamble, the font encoding to be used was explicitly declared. For multilingual documents *all* encodings needed should be enumerated via the `\usepackage[...]{fontenc}` command. Changing from one font encoding to another can be accomplished by using the `\fontencoding` command, but it is advisable that such changes be performed by a higher-level interface such as the `\selectlanguage` command. In particular, when using `babel`, you can write

```
\usepackage[koi8-r]{inputenc} \usepackage[russian]{babel}
```

where `babel` will automatically choose the default font encoding for Russian, which is T2A, when it is available. Table 9.6 on the following page shows the layout of the T2A encoding.

### Font encodings for Cyrillic languages

The Cyrillic font encodings support the languages listed below. Note that some languages, such as Bulgarian and Russian, can be properly typeset with more than one encoding.

T2A: Abaza, Avar, Agul, Adyghei, Azerbaijani, Altai, Balkar, Bashkir, Bulgarian, Buryat, Byelorussian, Gagauz, Dargin, Dungan, Ingush, Kabardino-Cherkess, Kazakh, Kalmyk, Karakalpak, Karachaevskii, Karelian, Kirghiz, Komi-Zyrian, Komi-Permyak, Kumyk, Lak, Lezghin, Macedonian, Mari-Mountain, Mari-Valley, Moldavian, Mongolian, Mordvin-Moksha, Mordvin-Erzya, Nogai, Oroch, Osetin, Russian, Rutul, Serbian, Tabasaran, Tadzhik, Tatar, Tati, Teleut, Tofalar, Tuva, Turkmen, Udmurt, Uzbek, Ukrainian, Hanty-Obskii, Hanty-Surgut, Gipsi, Chechen, Chuvash, Crimean-Tatar

	‘0	‘1	‘2	‘3	‘4	‘5	‘6	‘7	
‘00x	‘	‘‘	‘^	‘~	‘“	‘”	‘°	‘`	‘0x
‘01x	‘`	‘-	‘.	‘,	‘I	‘⟨	‘⟩		‘1x
‘02x	“	”	^	“	”	-	-		‘2x
‘03x	‘o	‘i	‘j	‘ff	‘fi	‘fl	‘ffi	‘fli	‘3x
‘04x	‘`	‘!	‘"	‘#	‘\$	‘%	‘&		‘4x
‘05x	‘l	‘)	‘^		‘.	‘-	‘.		‘5x
‘06x	‘0	‘1	‘2	‘3	‘4	‘5	‘6	‘7	‘6x
‘07x	‘8	‘9	‘:	‘:	‘<		‘>	‘?	‘7x
‘10x	‘o	‘A	‘B	‘C	‘D	‘E	‘F	‘G	‘8x
‘11x	‘H	‘I	‘J	‘K	‘L	‘M	‘N	‘O	‘9x
‘12x	‘P	‘Q	‘R	‘S	‘T	‘U	‘V	‘W	‘Ax
‘13x	‘X	‘Y	‘Z	‘	‘\	‘	‘^	‘—	‘Bx
‘14x	‘`	‘a	‘b	‘c	‘d	‘e	‘f	‘g	‘Cx
‘15x	‘h	‘i	‘j	‘k	‘l	‘m	‘n	‘o	‘Dx
‘16x	‘p	‘q	‘r	‘s	‘t	‘u	‘v	‘w	‘Ex
‘17x	‘x	‘y	‘z	‘{	‘	‘}	‘~	‘-	‘Fx
‘20x	‘Г	‘F	‘Ђ	‘Ћ	‘Ћ	‘Ж	‘З	‘Љ	
‘21x	‘Ї	‘К	‘К	‘К	‘Æ	‘Н	‘Н	‘S	
‘22x	‘Θ	‘Ҫ	‘Ӧ	‘Ӧ	‘Ӧ	‘Ӧ	‘Ӧ	‘Ӧ	
‘23x	‘҃	‘Ҽ	‘Ӫ	‘ӫ	‘ӫ	‘ӫ	‘ӫ	‘ӫ	
‘24x	‘г	‘ғ	‘ҕ	‘ҕ	‘h	‘ж	‘ڙ	‘ڃ	
‘25x	‘ї	‘Ҝ	‘ҝ	‘ҝ	‘æ	‘Ҥ	‘Ҥ	‘S	
‘26x	‘ө	‘ӫ	‘Ӧ	‘Ӧ	‘Ӧ	‘Ӧ	‘Ӧ	‘Ӧ	
‘27x	‘ҹ	‘Ҽ	‘ӫ	‘ӫ	‘ӫ	‘“	‘“	‘”	
‘30x	‘А	‘Б	‘В	‘Г	‘Д	‘Е	‘Ж	‘З	
‘31x	‘И	‘Ӣ	‘К	‘Л	‘М	‘Н	‘О	‘П	
‘32x	‘Р	‘С	‘Т	‘Ү	‘Ф	‘Х	‘Ц	‘Ч	
‘33x	‘Ш	‘Ӯ	‘Ђ	‘Ы	‘Ӯ	‘Э	‘Ю	‘Я	
‘34x	‘а	‘б	‘в	‘г	‘д	‘е	‘ж	‘з	
‘35x	‘и	‘Ӣ	‘к	‘л	‘м	‘н	‘о	‘п	
‘36x	‘р	‘с	‘т	‘ү	‘ф	‘х	‘ц	‘ч	
‘37x	‘ш	‘Ӯ	‘Ђ	‘ы	‘Ӯ	‘э	‘ю	‘я	
	“8	“9	“A	“B	“C	“D	“E	“F	

Characters marked in blue need to be present (in their specified positions) in every text encoding, as they are transparently passed through T<sub>E</sub>X.

Table 9.6: Glyph chart for a T2A-encoded font (larm1000)

- T2B: Abaza, Avar, Agul, Adyghei, Aleut, Altai, Balkar, Byelorussian, Bulgarian, Buryat, Gagauz, Dargin, Dolgan, Dungan, Ingush, Itelmen, Kabardino-Cherkess, Kalmyk, Karakalpak, Karachaevkii, Karelian, Ketskii, Kirghiz, Komi-Zyrian, Komi-Permyak, Koryak, Kumyk, Kurdian, Lak, Lezghin, Mansi, Mari-Valley, Moldavian, Mongolian, Mordvin-Moksha, Mordvin-Erzya, Nanai, Nganasan, Negidal, Nenets, Nivh, Nogai, Oroch, Russian, Rutul, Selkup, Tabasaran, Tadzhik, Tatar, Tati, Teleut, Tofalar, Tuva, Turkmen, Udyghei, Uigur, Ulch, Khakass, Hanty-Vahovskii, Hanty-Kazymskii, Hanty-Obskii, Hanty-Surgut, Hanty-Shurysharskii, Gipsi, Chechen, Chukcha, Shor, Evenk, Even, Enets, Eskimo, Yukagir, Crimean-Tatar, Yakut
- T2C: Abkhazian, Bulgarian, Gagauz, Karelian, Komi-Zyrian, Komi-Permyak, Kumyk, Mansi, Moldavian, Mordvin-Moksha, Mordvin-Erzya, Nanai, Orok (Ulta), Negidal, Nogai, Oroch, Russian, Saam, Old-Bulgarian, Old-Russian, Tati, Teleut, Hanty-Obskii, Hanty-Surgut, Evenk, Crimean-Tatar

The basic *L<sup>A</sup>T<sub>E</sub>X* distribution comes with all the encoding and font definition files for handling Cyrillic. The *babel* package includes support for Bulgarian, Russian, and Ukrainian. Together with the font files (to be installed separately), *L<sup>A</sup>T<sub>E</sub>X* can use this package to provide complete support for typesetting languages based on the Cyrillic alphabet.

#### **Running *MakeIndex* and *B<sub>B</sub>T<sub>E</sub>X***

Recognizing that standard *MakeIndex* and *B<sub>B</sub>T<sub>E</sub>X* programs cannot handle 8-bit input encodings natively, the T2 bundle comes with utilities to allow Cyrillic 8-bit input to be handled correctly by those programs.

For indexes, *rumakeindex* is a wrapper for *MakeIndex* that creates a properly sorted index when Cyrillic letters are used in the entries. Use of the *rumakeindex* utility also requires the *sed* program.<sup>1</sup> The utility should be run instead of standard *MakeIndex* when you are creating an index containing Cyrillic characters. Note that the *rumakeindex* script on UN\*X uses the koi8-r encoding, whereas the corresponding batch file on MS-DOS, *rumkidxd.bat*, uses the cp866 encoding, and the batch file on MS Windows, *rumkidxw.bat*, uses the cp1251 encoding. If a different encoding is needed, changes have to be introduced in the relevant files. Alternatively, you might consider using *xindy*, a newer index preparation program, which is described in Section 11.3.

For bibliographic references, *rubibtex* is a wrapper for *B<sub>B</sub>T<sub>E</sub>X* that produces Cyrillic letters in item names, which correspond to the reference keys when a *B<sub>B</sub>T<sub>E</sub>X* bibliographic database is used. You should also install the *citehack* package from the T2 bundle in that case. Moreover, the installed version of the *B<sub>B</sub>T<sub>E</sub>X* program should be able to handle 8-bit input (e.g., the *B<sub>B</sub>T<sub>E</sub>X8* program described

<sup>1</sup> Available on any UN\*X and for Microsoft operating systems on PC distributed by GNU (e.g., at <http://www.simtel.net>).

in Section 13.1.1). As in the case of *MakeIndex* described above, the **rubibtex** script and batch files also require the **sed** program.

Note that the **rubibtex** script on UN\*X uses the **koi8-r** encoding, whereas the corresponding batch file on MS-DOS, **rubibtex.bat**, uses the **cp866** encoding. When another encoding is needed, changes should be introduced in the relevant files.

#### 9.4.2 The Greek alphabet

Greek support in **babel** comes in two variants: the one-accent **monotoniko** (the default), which is used in most cases in everyday communications in Greece today, and the multi-accent **polotoniko**, which has to be specified as an attribute, as explained in Section 9.2.3.

The first family of Greek fonts for T<sub>E</sub>X was created during the mid-1980s by Silvio Levy [114]. Other developers improved or extended these fonts, or developed their own Greek fonts.

In **babel** the Greek language support is based on the work of Claudio Beccari in collaboration with Apostolos Syropoulos, who developed the Greek **cb** font family [12]. In their paper these authors discuss in some detail previous efforts to support the Greek language with T<sub>E</sub>X. The sources of the **cb** fonts are available on CTAN in the directory **languages/greek/cb** or on the T<sub>E</sub>X Live CD in the directory **texmf/fonts/source/public/cbgreek**. Hyphenation patterns corresponding to this font family are found in the file **grhyph.tex** or **grphyph.tex** in the same directory on CTAN and in **texmf/tex/generic/hyphen** on T<sub>E</sub>X Live.

The **cb** fonts use the LGR font encoding. At the time of this book's writing, work was under way to design a font encoding that is compatible with L<sup>A</sup>T<sub>E</sub>X's standards. When it is ready, it will become the T7 encoding. Table 9.7 on the next page shows the layout of the complete LGR encoding.

It is possible to use Latin alphabetic characters for inputting Greek according to the transliteration scheme shown in Table 9.8 on page 576. This table shows that the Latin "v" character has no direct equivalent in the Greek transcription. In fact, it is used to indicate that one *does not* want a final sigma. For example, "sv" generates a median form sigma although it occurs in a final position.

By default, the **greek** option of **babel** will use **monotoniko** Greek. Multi-accented mode is requested by specifying the **language** attribute **polotoniko** for the **greek** option:

```
\usepackage[greek]{babel}
\languageattribute{greek}{polotoniko}
```

For both modes, some seldom-used characters have been defined to behave like letters (\catcode 11). For **monotoniko** Greek, this is the case for the characters ' and ". In the **polotoniko** variant, the characters <, >, ~, ', and | also behave like letters. The reason for this behavior is that the LGR encoding contains many ligatures with these characters to produce the right glyphs; see Table 9.9 on page 576. Table 9.10 shows the available composite accent and spiritus combinations.

	'0	'1	'2	'3	'4	'5	'6	'7	
'00x	-	~	¤	¤	¤	¤	¤	¤	"0x
'01x	ı	A <sub>I</sub>	H <sub>I</sub>	Ω <sub>I</sub>	A	Ŷ	α	ü	"1x
'02x	,	„	„	„	„	„	„	„	"2x
'03x	€	%o	ə	˜	‘	‘	‘	‘	"3x
'04x	„	!	,	*	*	%	.	,	"4x
'05x	(	)	*	+	.	-	.	/	"5x
'06x	0	1	2	3	4	5	6	7	"6x
'07x	8	9	:	:	<	=	>	:	"7x
'10x	~	A	B	~	Δ	E	Φ	Γ	"8x
'11x	H	I	Θ	K	Λ	M	N	O	"9x
'12x	Π	X	P	Σ	T	Υ	„	Ω	"Ax
'13x	Ξ	Ψ	Z	[	„	]	„	„	"Bx
'14x	‘	α	β	ς	δ	ε	ϙ	ϙ	"Cx
'15x	ı	ı	ı	ı	ı	ı	ı	ı	"Dx
'16x	π	χ	ρ	ς	τ	υ	ω	ω	"Ex
'17x	ξ	ψ	ζ	«	,	»	~	—	"Fx
'20x	à	à	à	à	à	à	à	à	"8x
'21x	á	á	á	á	á	á	á	á	"9x
'22x	ã	ã	ã	F	ã	ã	ã	ã	"Ax
'23x	ä	ä	ä	ä	ä	ä	ä	ä	"Bx
'24x	ë	ë	ë	ë	ë	ë	ë	ë	"Cx
'25x	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	"Dx
'26x	ô	ô	ô	ô	ô	ô	ô	ô	"Ex
'27x	ó	ó	ó	ó	ó	ó	ó	ó	"Fx
'30x	ö	ö	ö	F	ö	ö	ö	ö	"8x
'31x	ı	ı	ı	ı	ı	ı	ı	ı	"9x
'32x	ı	ı	ı	ı	ı	ı	ı	ı	"Ax
'33x	ı	ı	ı	ı	ı	ı	ı	ı	"Bx
'34x	ë	ë	ë	ë	ë	ë	ë	ë	"Cx
'35x	é	é	é	é	ó	ó	ó	ó	"Dx
'36x	í	í	í	í	ü	ü	ü	ü	"Ex
'37x	ó	ó	ó	ó	ó	ó	ó	ó	"Fx
	"8	"9	"A	"B	"C	"D	"E	"F	

Characters marked in blue should be ASCII characters in every *L<sup>A</sup>T<sub>E</sub>X* text encoding (compare Table 9.6 on page 572), as they are transparently passed through *T<sub>E</sub>X*. In LGR this is not the case for A-Z and a-z, which can produce problems in multilingual documents.

Table 9.7: Glyph chart for an LGR-encoded font (grmn1000)

a b c d e f g h i j k l m n o p q r s t u v w x y z	α β θ ε φ γ η ι θ κ λ μ ν ο π χ ρ ζ τ υ ω ξ ψ ζ	
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z	Α Β Κ Δ Ε Φ Γ Η Ι Θ Κ Λ Μ Ν Ο Π Χ Ρ Σ Τ Υ Ζ	[9-4-1]
A B " Δ E Φ Γ H I Θ K Λ M N O Π X P Σ T Y Ζ	Α Β " Δ Ε Φ Γ Η Ι Θ Κ Λ Μ Ν Ο Π Χ Ρ Σ Τ Υ Ζ	

Table 9.8: Greek transliteration with Latin letters for the LGR encoding

	<i>Input</i>	<i>Result</i>	<i>Example</i>	
Acute	'a 'e 'h 'i 'o 'u 'w	άέήόύώ	g'ata	γάτα
Diaeresis	"i "u "I "U	ιύΙΥ	qa"ide'uh c	χαδεύης
Rough breathing	<a <e <h <i <o <r <u <w	άέήόύώ	<'otan	δταν
Smooth breathing	>a >e >h >i >o >r >u >w	άέήόύώ	>'aneu	·άνευ
Grave	'a 'e 'h 'i 'o 'u 'w	άέήόύώ	dad'i	δαδί
Circumflex	~a ~h ~i ~u ~w	άέήόύώ	ful~hc	φυλής
Diacritic below	a  h  w	άηώ		
	'w  ~w  >'w  >~w  <'w  <~w	ɸɸɸɸɸɸɸ		

Table 9.9: LGR ligatures producing single-accented glyphs

<i>Input</i>	<i>Result</i>	<i>Input</i>	<i>Result</i>	
''i ''i ''u ''u ̄̄̄̄	ἴ̄̄̄̄			
>'a >'e >'h >'i >'o >'u >'w	ἄέήόύώ	>'A >'E >'H >'I >'O >'U >'W	ἌΈΉΊΏ	
>'a >'e >'h >'i >'o >'u >'w	ἄέήόύώ	>'A >'E >'H >'I >'O >'U >'W	ἌΈΉΊΏ	
<'a <'e <'h <'i <'o <'u <'w	άέήόύώ	<'A <'E <'H <'I <'O <'U <'W	ΆΈΉΊΏ	
<'a <'e <'h <'i <'o <'u <'w	άέήόύώ	<'A <'E <'H <'I <'O <'U <'W	ΆΈΉΊΏ	
>~a >~h >~i >~u >~w	άήήύώ	>~A >~H >~I >~U >~W	Άήήύώ	
<~a <~h <~i <~u <~w	άήήύώ	<~A <~H <~I <~U <~W	Άήήύώ	[9-4-2]

Table 9.10: Available composite spiritus and accent combinations

### 9.4.3 The Hebrew alphabet

The first support for Hebrew that became part of the babel distribution was developed by Boris Lavva and Alon Ziv, based on earlier work that offered support for typesetting Hebrew texts with L<sup>A</sup>T<sub>E</sub>X 2.09 and T<sub>E</sub>X-X<sub>E</sub>T. This support was developed further by these two authors and Rama Porrat. At the time of writing Tzafrir Cohen has started a sourceforge project called “ivritex” (<http://ivritex.sf.net>) to extend the work even more.

# כותרת של מאמר שנכתב באמצעות $\text{\LaTeX}$ בעברית

רמה פורט

2 בפברואר 2002

## 1 זהות התחלה

אפשר להתחיל בעברית ולאחר מכן לעבור ללווזיות  
אפשר גם לכלול נוסחאות  $102 = a^2 + 83$  ולהזור בעברית.

### 1.1 תחת סעיף בעבודה

- \* המצב לא טוב אבל יש תקווה לעתיד.

\* יש לפניך שולחן עם המון מטבחו. לכל מטבח צד לבן הצד השחור. עיין  
 קשרוות. אתה יודע ש 10 מטבחות הן עם הצד הלבן למטה, והשאר עם  
 הצד השחור למעלה. עליך לחלק את המטבחות לשתי קבוצות, כך שבכל  
 קבוצה יהיה אותו מספר של לבנים (כלומר צד לבן למטה).

לא מצאת את הפתרון? - [rama@huji.ac.il](mailto:rama@huji.ac.il)

## Last section 2

9-4-4

אפשר להשתמש בכל האפשרויות של התוכנה בגירסתה הלווזית.

Figure 9.1: A Hebrew document

The current support for typesetting Hebrew is based on fonts from the Hebrew University of Jerusalem. These fonts have a particular 7-bit encoding for which the Local Hebrew encoding (LHE) has been developed. Figure 9.1 used the Jerusalem font; in Table 9.11 on the following page the encoding of these fonts is shown. The support in babel uses the Jerusalem font as the regular font, Old Jaffa for a font with an italic shape, and the Dead Sea font for typesetting bold letters. When a sans serif font is needed, the Tel Aviv font is used; it is also deployed as a replacement for a typewriter font.

As an alternative to these fonts, two other (copyrighted, but freely available on CTAN) fonts are supported: Hclassic is a “modernized Classical Hebrew” font; Hcaption is a slanted version of it. Furthermore, three shalom fonts are available: ShalomScript10 contains handwritten Hebrew letters; ShalomStick10 contains sans serif letters; and ShalomOldStyle10 contains old-style letters. Yet an-

	'0	'1	'2	'3	'4	'5	'6	'7	
'02x	א	א	װ	ײ	,	"	"	װ	
'03x	װ	-	-	-	-				"1x
'04x		!	׮						
'05x	(	)			.	.	.	/	"2x
'06x			.		.	.	.	.	
'07x	.	.	:	:	׮		׮	?	"3x
'10x	.	.	.	.	.	.	.	.	"4x
'11x	.	.	.	.	.	.	.	.	
'12x	.	.	.	.	.	.	.	.	"5x
'13x	.	.	.	.	\	.	.	.	
'14x	א	ב	ג	ד	ה	ו	׮	׮	"6x
'15x	ט	,	׮	כ	ל	ם	ס	׮	
'16x	נ	ס	׮	׮	׮	׮	׮	׮	"7x
'17x	׮	װ	׮						
	"8	"9	"A	"B	"C	"D	"E	"F	

Table 9.11: Glyph chart for an LHE-encoded font (shold10)

other available family of fonts are the Frank Ruehl fonts, which come in regular, bold extended, and slanted shapes. The Carmel font family offers regular and slanted shapes and was designed for headers and emphasized text. The Redis family comes with regular, slanted, and bold extended shapes. For all supported font families, the package `hebfont` defines commands to select them. These commands are shown in Table 9.12 on the next page.

A few input encodings are available as part of the support for Hebrew. They are not automatically provided with the `inputenc` distribution.

`si960` This 7-bit Hebrew encoding uses ASCII character positions 32–127. Also known as “oldcode”, it is defined by Israeli standard SI-960.

`8859-8` This 8-bit mixed Hebrew and Latin encoding is also known as “newcode”. It is defined by the standard ISO 8859-8.

`cp862` This IBM code page is commonly used by MS-DOS on IBM-compatible personal computers. It is also known as “pccode”.

`cp1255` The MS Windows 1255 (Hebrew) code page resembles ISO 8859-8. In addition to Hebrew letters, this encoding contains vowels and dots (nikud).

<i>Command</i>	<i>Corresponds to Declaration</i>	<i>Font Family</i>	<i>Example</i>
\textjm	\rmfamily	Jerusalem font	מִפְוָמָפָךְ
\textds	\bfseries	Dead Sea font	זִמְפּוֹמָפָךְ
\textoj	\itshape	Old Jaffa font	לִמְפּוֹמָפָךְ
\textta	\sffamily	Tel Aviv font	תֵּלְאַבִּיבְמִפְוָמָפָךְ
	\ttfamily		
\textcrml	\fontfamily{crml}	Carmel fonts	כַּרְמֵלְמִפְוָמָפָךְ
\textfr	\fontfamily{fr}}	Frank Ruehl fonts	פְּרָנָקְרַיְהָמִפְוָמָפָךְ
\textredis	\fontfamily{redis}	Redis fonts	רֵדִיסְמִפְוָמָפָךְ
\textclas	\fontfamily{clas}	Classic fonts	כְּלִסְטִיקְמִפְוָמָפָךְ
\textshold	\fontfamily{shold}	Shalom Old Style font	שָׁלוֹםְאָוֶןְמִפְוָמָפָךְ
\textshscr	\fontfamily{shscr}	Shalom Script font	שָׁלוֹםְסְקְרִיטְמִפְוָמָפָךְ
9-4-5	\textshstk	Shalom Stick font	שָׁלוֹםְשִׁקְקָחָמִפְוָמָפָךְ

Table 9.12: Hebrew font-changing commands

## 9.5 Tailoring babel

This section explains some of the commands that are made available by the core `babel` package to construct language definition files (which are usually loaded when a language option is requested). Section 9.5.3 then looks in some detail at the template file `language.skeleton`, which can be used as a basis to provide support for additional languages.

Language definition files (file extension `.ldf`) have to conform to a number of conventions, since they complement the common shared code of `babel` provided in the file `babel.def` for producing language-dependent text strings. Similarly, to allow for language switching like the capability built into `babel`, certain rules apply. The basic working assumptions follow.

- Each language definition file `<lang>.ldf` must define five macros, which are subsequently used to activate and deactivate the language-specific definitions. These macros are `\<language>hyphenmins`, `\captions<language>`, `\date<language>`, `\extras<language>`, and `\noextras<language>`, where `<language>` is either the name of the language definition file or the name of a `babel` package option. These macros and their functions are discussed below.
- When a language definition file is loaded, it can define `\l@<language>` to be a variant (*dialect*) of `\language0` when `\l@<language>` is undefined.
- The language definition files must be written in a way that they can be read not just in the preamble of the document, but also in the middle of document processing.

### 9.5.1 Hyphenating in several languages

Since T<sub>E</sub>X version 3.0, hyphenation patterns for multiple languages can be used together. These patterns have to be administered somehow. In particular, the plainT<sub>E</sub>X user has to know for which languages patterns have been loaded, and to what values of the command sequence `\language` they correspond. The babel package abstracts from this low-level interface and manages this information by using an external file, `language.dat`, in which one records which languages have hyphenation patterns *and* in which files these patterns are stored. This configuration file is then processed<sup>1</sup> when INIT<sub>E</sub>X is run to generate a new L<sup>A</sup>T<sub>E</sub>X format. An example of this file is shown here:

```
%%% Filename : language.dat
%%% Description : Instruct iniTeX which pattern files to load.

english      ushyph.tex          % American English
=USenglish
=american

russian      ruhyph.tex         % Russian
french       frhyph.tex   frhyphx.tex % French
=patois
=francais

UKenglish    gbhyph.tex         % UK English
=british

german       dehypt.tex          % Traditional German
%ngerman     dehyphn.tex         % New German (not loaded)
%dutch       nehyph96.tex        % Dutch (not loaded)

dumylang     dumyhyph.tex        % For testing new language
nohyphenation zerohyph.tex      % Language with no patterns
```

This configuration file `language.dat` can contain empty lines and comments, as well as lines that start with an equals (=) sign. Such a line will instruct L<sup>A</sup>T<sub>E</sub>X that the hyphenation patterns just processed will be known under an alternative name. The first element on each line specifies the name of the language; it is followed by the name of the file containing the hyphenation patterns. An optional third entry can specify a hyphenation exception file in case the exceptions are stored in a separate file (e.g., `frhyphx.tex` in the previous example).

For each language in `language.dat`, the command `\l@<language>` is defined in the L<sup>A</sup>T<sub>E</sub>X format (i.e., `\l@english` and so on). When the document is processed with such a format, babel checks for each language whether the command `\l@<language>` is defined and, if so, it loads the corresponding hyphenation pat-

<sup>1</sup>Make sure that you do not have several such files in your T<sub>E</sub>X installation, because it is not always clear which of them will be examined during the format generation. The authors nearly got bitten during the book production when INIT<sub>E</sub>X picked up the system configuration file and not the specially prepared one containing all the patterns for the examples.

terns; otherwise, it loads the patterns for the default language 0 (the one loaded first by INITEX); for compatibility reasons this language should contain US-English hyphenation patterns.

```
initex latex.ltx
This is TeX, Version 3.14159 (Web2C 7.3.3.1) (INITEX)
(/tex/texmf/tex/latex/base/latex.ltx
...
24 hyphenation exceptions
Hyphenation trie of length 33878 has 835 ops out of 1501
  2 for language 5
  207 for language 4
  224 for language 3
  86 for language 2
  135 for language 1
  181 for language 0
No pages of output.
```

Seven “languages” are loaded into the format, as defined in the `language.dat` file: `english` (0), `russian` (1), `french` (2), `UKenglish` (3), `german` (4), `dumylang` (5), and `nohyphenation` (6; implicitly defined with no hyphenation tries). Babel uses these text strings (or their equivalents, specified preceded by an = sign in `language.dat`) to identify a language.

If `language.dat` cannot be opened for reading during the INITEX run, babel will attempt to use the default hyphenation file `hyphen.tex` instead. It informs the user in this event.

### 9.5.2 The package file

To help make use of the features of L<sup>A</sup>T<sub>E</sub>X, the `babel` package contains a package file called `babel.sty`. This file is loaded by the `\usepackage` command and defines all the language options supported by `babel` (see Table 9.1 on page 543). It also takes care of a number of compatibility issues with other packages. Local customization for `babel` can be entered in the configuration file `bbllopts.cfg`, which is read at the end of `babel.sty`.

Apart from the language options listed in Table 9.1 on page 543, `babel` pre-declares a few options that can influence the behavior of language definition files. For instance, `activeacute` and `activegrave` by default do nothing, but they are used with, for instance, Catalan (`catalan.ldf`) to activate the acute and grave accents when the relevant options are specified.

A third option, `KeepShorthandsActive`, instructs `babel` to keep shorthand characters active when processing of the package file ends. Note that this is *not* the default as it can cause problems with other packages. Nevertheless, in some cases, such as when you need to use shorthand characters in the preamble of a document, this option can be useful.

### 9.5.3 The structure of the babel language definition file

The babel distribution comes with the file `language.skeleton`, which provides a convenient skeleton for developing one's own language file to support a new language. It serves as a convenient model to understand how the babel core commands are used. The file is shown here, and the commands used in it are described as they occur.

Throughout `language.skeleton`, you will find the string “`(language)`”; it should be replaced by the name of the language for which you are providing support. If this language is known to have a dialect that needs a slightly different support, you can arrange for this support as well. In such a case, the strings “`(dialect)`” should be replaced by the name of the dialect. If your language does not need support for a dialect, you should remove the corresponding lines of code.

*Copyright and introduction* The file starts with copyright and license information.

```

1  % \iffalse meta-comment
2  %
3  % Copyright 1989-2003 Johannes L. Braams and any individual authors
4  % listed elsewhere in this file. All rights reserved.
5  %
6  % This file is part of the Babel system release 3.7.
7  % -----
8  %
9  % This work may be distributed and/or modified under the
10 % conditions of the LaTeX Project Public License, either version 1.3
11 % of this license or (at your option) any later version.
12 % The latest version of this license is in
13 % http://www.latex-project.org/lppl.txt
14 % and version 1.3 or later is part of all distributions of LaTeX
15 % version 2003/12/01 or later.
16 %
17 % This work has the LPPL maintenance status "maintained".
18 %
19 % This Current Maintainer of this work is Johannes Braams.
20 %
21 % \fi
22 % \CheckSum{0}
23 %% docstring      = " This file can act as a template for
24 %%                   people who want to provide extra
25 %%                   languages to be included in the babel
26 %%                   distribution.
27 %

```

*Identification of the language* This is followed by information identifying the file and language.

```

28  %<*dtx>
29  % \iffalse
30  %   Tell the \LaTeX\ system who we are and write an entry in the
31  %   transcript file.
32  %\ProvidesFile{<language>.dtx}
33  %</dtx>
34  %<code>\ProvidesLanguage{<language>}
35  %\fi
36  %\ProvidesFile{<language>.dtx}
37      [2003/03/18 v1.5 <Language> support from the babel system]

```

```
\ProvidesLanguage{name} [release-information]
```

The command `\ProvidesLanguage` (line 34) identifies the language definition file. It uses the same syntax as L<sup>A</sup>T<sub>E</sub>X's `\ProvidesPackage`. For instance, the file `welsh.ldf` contains the following declaration:

```
\ProvidesLanguage{welsh}
```

The *release-information* can be used to indicate that at least this version of `babel` is required.

The next section then sets up a documentation driver to allow for typesetting the file itself using the `doc` package. See Chapter 14 for details.

*A documentation driver*

```

38  \%iffalse
39  %% Babel package for LaTeX version 2e
40  %% Copyright (C) 1989 -- 2003
41  %%           by Johannes Braams, TeXniek
42  %
43  %% Please report errors to: J.L. Braams
44  %%                         babel@braams.cistron.nl
45  %
46  %     This file is part of the babel system, it provides the source code for
47  %     the <language> language definition file.
48  %<*filedriver>
49  \documentclass{ltxdoc}
50  \newcommand*{\TeXhax}{\TeX hax}
51  \newcommand*{\babel}{\textsf{babel}}
52  \newcommand*{\langvar}{$\langle\!\!\langle \mathit{lang} \rangle\!\!\rangle$}
53  \newcommand*{\note}[1]{}
54  \newcommand*{\Lopt}[1]{\textsf{#1}}
55  \newcommand*{\file}[1]{\texttt{#1}}
56  \begin{document}
57  \DocInput{<language>.dtx}
58  \end{document}
59  %</filedriver>
60  %
61  %% \GetFileInfo{<language>.dtx}
62  %
```

The following part starts with the documentation of the features provided by the language definition file. Use the methods described in Chapter 14 for documenting code and providing a short user manual.

```

63  % \changes{v1.1}{1994/02/27}{Rearranged the file a little}
64  % \changes{v1.2}{1994/06/04}{Update for \LaTeXe}
65  % \changes{v1.3}{1995/05/13}{Update for \babel\ release 3.5}
66  % \changes{v1.4}{1996/10/30}{Update for \babel\ release 3.6}
67  % \changes{v1.5}{1997/03/18}{Update for \babel\ release 3.7}
68  %
69  % \section{The <language> language}
70  %
71  %   The file \file{\filename}\footnote{The file described in this
72  %   section has version number \fileversion\ and was last revised on
73  %   \filedate.} defines all the language definition macros for the
74  %   <language> language.
75  %
76  % \StopEventually{}%
```

```

77  %
78  %     The macro |\LdfInit| takes care of preventing that this file is
79  %     loaded more than once, checking the category code of the
80  %     \texttt{@} sign, etc.
81  %     \begin{macrocode}
82  %<*code>
83  \LdfInit{<language>}{captions<language>}
84  %     \end{macrocode}
85  %

```

### \LdfInit

The macro `\LdfInit` (line 83) performs a couple of standard checks that have to be made at the beginning of a language definition file, such as checking the category code of the `@` sign and preventing the `.ldf` file from being processed twice.

```

Defining language
and dialects   86  % When this file is read as an option, i.e. by the |\usepackage|
87  % command, \texttt{@<language>} could be an ‘unknown’ language in
88  % which case we have to make it known. So we check for the
89  % existence of |\l@<language>| to see whether we have to do
90  % something here.
91 %
92 %     \begin{macrocode}
93 \ifx\undefined\l@<language>
94   \nopatterns{<language>}
95   \adddialect{\l@<language>0}\fi
96 %     \end{macrocode}
97 %     For the <Dialect> version of these definitions we just add a
98 %     ‘‘dialect’’. Also, the macros |\captions{dialect}| and
99 %     |\extras{dialect}| are |\let| to their \texttt{@<language>} counterparts
100 %     when these parts are defined.
101 %     \begin{macrocode}
102 \adddialect{\l@<dialect>}\l@<language>
103 %     \end{macrocode}
104 %     The next step consists of defining commands to switch to (and
105 %     from) the <Language> language.
106 %

```

### \adddialect{\l@variant}{\l@lang}

The command `\adddialect` adds the name of a variant (dialect) language `\l@variant`, for which already defined hyphenation patterns can be used (the ones for language `lang`).<sup>1</sup> If a language has more than one variant, you can repeat this section as often as necessary.

“Dialect” is somewhat of a historical misnomer, as `lang` and `variant` are at the same level as far as `babel` is concerned, without co-notation indicating whether one or the other is the main language. The “dialect” paradigm comes in handy if you want to share hyphenation patterns between various languages. Moreover, if no hyphenation patterns are preloaded in the format for the language `lang`, `babel`’s default behavior is to define this language as a “dialect” of the default language (`\language0`).

<sup>1</sup>When loading hyphenation patterns with L<sup>A</sup>T<sub>E</sub>X `babel` uses the `\addlanguage` command to declare the various languages specified in `language.dat`; see Section 9.5.1.

For instance, the first line below indicates that for Austrian one can use the hyphenation patterns for German (defined in `german.ldf`). The second line tells us that Nynorsk shares the hyphenation patterns of Norsk (in `norsk.ldf`).

```
\adddialect{\l@austrian}{\l@german}
\adddialect{\l@nynorsk}{\l@norsk}
```

The following example shows how language variants can be obtained using the dialect mechanism, where there can be differences in the names of sectioning elements or for the date.

Dialectical variants: Norsk: Bibliografi Nynorsk: Litteratur Dutch: 29 februari 2004 Afrikaans: 29 Februarie 2004	\usepackage[dutch,afrikaans,norsk,nyrnorsk,english]{babel} Dialectical variants: \par \selectlanguage{norsk} Norsk: \bibname \par \selectlanguage{nyrnorsk} Nynorsk: \bibname \par \selectlanguage{dutch} Dutch: \today \par \selectlanguage{afrikaans} Afrikaans: \today
---	--

The next part deals with the set-up for language attributes, if necessary.

*Defining language attributes*

```
107 % Now we declare the |<attrib>| language attribute.
108 % \begin{macrocode}
109 \bbl@declare@ttribute{<language>}{{<attrib>}}{%
110 % \end{macrocode}
111 % This code adds the expansion of |\extras<attrib><language>| to
112 % |\extras<language>|.
113 % \begin{macrocode}
114 \expandafter\addto\expandafter\extras<language>
115 \expandafter\extras{<attrib><language>}{%
116 \let\captions<language>\captions<attrib><language>
117 }
118 % \end{macrocode}
119 %
```

```
\bbl@declare@ttribute{lang}{attr}{exec}
```

This command (used on line 109) declares that for the attribute *attr* in the language *lang*, the code *exec* should be executed. For instance, the file `greek.ldf` defines an attribute `polutoniko` for the Greek language:

```
\bbl@declare@ttribute{greek}{polutoniko}{...}
```

When you load the Greek language with the `polutonikogreek` option (which is equivalent to setting the attribute `polutoniko`), Greek will then be typeset with multiple accents (according to the code specified in the third argument).

If you want to define more than one attribute for the current language, repeat this section as often as necessary.

*Adjusting hyphenation patterns* Now we deal with the minimum number of characters required to the left and right of hyphenation points.

```

120  % \begin{macro}{\<language>hyphenmins}
121  % This macro is used to store the correct values of the hyphenation
122  % parameters \lefthyphenmin and \righthyphenmin.
123  % \begin{macrocode}
124  \providehyphenmins{\<language>}{{\tw@}{\thr@}}
125  % \end{macrocode}
126  % \end{macro}
127  %

```

`\providehyphenmins{lang}{hyphenmins} \<language>hyphenmins`

The command `\providehyphenmins` (line 124) provides a *default* setting for the hyphenation parameters `\lefthyphenmin` (minimum number of characters on the left before the first hyphen point) and `\righthyphenmin` (minimum numbers on the right) for the language *lang*, by defining `\<language>hyphenmins` unless it is already defined for some reason. The `babel` package detects whether the hyphenation file explicitly sets `\lefthyphenmin` and `\righthyphenmin` and automatically defines `\<language>hyphenmins`, in which case the `\providehyphenmins` declaration has no effect.

The syntax inside `babel` is storage optimized, dating back to the days when every token counted. Thus, the argument *hyphenmins* contains the values for both parameters simply as two digits, making the assumption that you will never want a minimum larger than 9. If this assumption is wrong, you must surround the values with braces within *hyphenmins*. For example,

`\providehyphenmins{german}{{10}{5}}`

would request to leave at least 10 characters before a hyphen and at least 5 characters after it (thus essentially never hyphenate).

If you want to explicitly overwrite the settings regardless of any existing specification, you can do so by providing a value for `\<language>hyphenmins` yourself. For instance,

`\def\germanhyphenmins{43}`

never considers hyphenation points with less than four letters before and three letters after the hyphen. Thus, it will never hyphenate a word with less than seven characters.

Hyphenation patterns are built with a certain setting of these parameters in mind. Setting their values lower than the values used in the pattern generation will merely result in incorrect hyphenation. It is possible, however, to use higher values in which case the potential hyphenation points are simply reduced.

*Translations for language-dependent strings* The translations for language-dependent strings are set up next.

```

language-dependent 128  % \begin{macro}{\captions<language>}
strings 129  % The macro |\captions<language>| defines all strings used in the
130  % four standard documentclasses provided with LATEX.
131  % \begin{macrocode}

```

```

132  \def\captions<language>{}
133  %   \end{macrocode}
134  % \end{macro}
135  %
136  % \begin{macro}{\captions<dialect>}
137  %   \begin{macrocode}
138  \let\captions<dialect>\captions<language>
139  %   \end{macrocode}
140  % \end{macro}
141  %

```

*\captions<language>{replacement text definitions}*

The macro `\captions<language>` (line 132) defines the macros that hold the translations for the language-dependent strings used in L<sup>A</sup>T<sub>E</sub>X for the language `<language>`. It must also be provided for each dialect being set up. If the dialect uses the same translation, `\let` can be used (as shown in line 138). Otherwise, you have to provide a full definition.

```

142  % \begin{macro}{\date<language>}
143  %   The macro |\date<language>| redefines the command |\today| to
144  %   produce <Language> dates.
145  %   \begin{macrocode}
146  \def\date<language>{%
147  }
148  %   \end{macrocode}
149  % \end{macro}
150  %
151  % \begin{macro}{\date<dialect>}
152  %   The macro |\date<dialect>| redefines the command |\today| to
153  %   produce <Dialect> dates.
154  %   \begin{macrocode}
155  \def\date<dialect>{%
156  }
157  %   \end{macrocode}
158  % \end{macro}
159  %

```

*\date<language>{definition of date}*

The macro `\date<language>` (line 146) defines the text string for the `\today` command for the language `<language>` being defined in a `.ldf` file.

For some languages (or dialects), extra definitions have to be provided. This is *Providing extra features* done in the next section.

```

160  % \begin{macro}{\extras<language>}
161  % \begin{macro}{\noextras<language>}
162  %   The macro |\extras<language>| will perform all the extra
163  %   definitions needed for the <Language> language. The macro
164  %   |\noextras<language>| is used to cancel the actions of
165  %   |\extras<language>|. For the moment these macros are empty but
166  %   they are defined for compatibility with the other
167  %   language definition files.
168  %
169  %   \begin{macrocode}
170  \addto\extras<language>{%
171  \addto\noextras<language>{%
172  %   \end{macrocode}

```

```

173  % \end{macro}
174  % \end{macro}
175  %
176  % \begin{macro}{\extras<dialect>}
177  % \begin{macro}{\noextras<dialect>}
178  %   Also for the '<dialect>' variant no extra definitions are
179  %   needed at the moment.
180  %   \begin{macrocode}
181  \let\extras<dialect>\extras<language>
182  \let\noextras<dialect>\noextras<language>
183  % \end{macrocode}
184  % \end{macro}
185  % \end{macro}
186  %

```

### `\extras<language>{extra definitions}`

The macro `\extras<language>` (line 170) contains all extra definitions needed for the language `<language>` being defined in a `.ldf` file. Such extras can be commands to turn shorthands on or off, to make certain characters active, to initiate French spacing, to position umlauts, and so on.

### `\noextras<language>{reverse extra definitions}`

To allow switching between any two languages, it is necessary to return to a known state for the T<sub>E</sub>X engine—in particular, with respect to the definitions initiated by the command `\extras<language>`. The macro `\noextras<language>` (line 171) must contain code to revert all such definitions so as to bring T<sub>E</sub>X back to a known state.

*Clean up and finish* The file finishes with the following lines of code.

```

187  % The macro |\ldf@finish| takes care of looking for a
188  % configuration file, setting the main language to be switched on
189  % at |\begin{document}| and resetting the category code of
190  % \texttt{@} to its original value.
191  % \begin{macrocode}
192  \ldf@finish{<language>}
193  %</code>
194  % \end{macrocode}
195  %
196  % \Finale
197  %\endinput

```

### `\ldf@finish{lang}`

The macro `\ldf@finish` (line 192) performs a couple of tasks that are necessary at the end of each `.ldf` file. The argument `lang` is the name of the language as it is defined in the language definition file. The macro starts by verifying whether the system contains a file `lang.cfg`—that is, a file with the same name as the language definition file, but with the extension `.cfg`. This file can be used to add site-specific actions to a language definition file, such as adding strings to `\captions<language>` to support local document classes, or activating or deactivating shorthands for acute or grave accents. In particular, the `babel` distribution

for French written by Daniel Flipo comes with a file `frenchb.cfg` that contains a few (commented-out) supplementary definitions for typesetting French that can be activated (uncommented) by the user if they appear to be useful. Other tasks performed by the macro include resetting the category code of the @ sign, and preparing the language to be activated at the beginning of the document.

### Adding definitions to babel's data structures

On various lines (114, 170, 171), the command `\addto` was used to extend one of the `babel` data structures holding translations or code for a certain language.

```
\addto\csname{code}
```

This command extends the definition of the control sequence `\csname` with the TeX code specified in `code`. The control sequence `\csname` does not have to have been defined previously. As an example, the following lines are taken from the file `russianb.ldf`, where `code` is added to the commands `\captionsrussian`, `\extrasrussian`, and `\noextrasrussian`.

```
\addto\captionsrussian{%
  \def\prefacename{%
    {\cyr\CYRP\cyrr\cyre\cyrd\cyri\crys\cyrl\cyro\cyrv\cyri\cyre}}%
  ...
}
\addto\extrasrussian{\cyrillictext}
\addto\noextrasrussian{\latintext}
\initiate@active@char{"
\addto\extrasrussian{\languageshorthands{russian}}
\addto\extrasrussian{\bblob@activate{"}}
\addto\noextrasrussian{\bblob@deactivate{"}}
}
```

### Language-level commands for shorthands

Shorthands on the language or system level are set up in the language definition files. An incomplete example of this process was given in the previous section. In this section we describe all commands and declarations that can be used for this purpose.

```
\initiate@active@char{char}
```

This macro can be used in language definition files to turn the character `char` into a “shorthand character”. When the character is already defined to be a shorthand character, this macro does nothing. Otherwise, it defines the control sequence `\normal@char{char}` to expand to the character `char` in its “normal state” and it

defines the active character to expand to `\normal@char{char}` by default. Subsequently, its definition can be changed to expand to `\active@char{char}` by calling `\bbl@activate{char}`. When a character has been made active, it will remain active until deactivated or until the end of the document is reached. Its definition can be changed at any time during the typesetting stage of the document.

For example, several language definition files make the double quote character active with the following statement:

```
\initiate@active@char{"}
```

For French the configuration file `frenchb.cfg` defines two-character shorthands:

```
\initiate@active@char{<<}    \initiate@active@char{>>}
```

```
\bbl@activate{char}    \bbl@deactivate{char}
```

The command `\bbl@activate` “switches on” the active behavior of the character *char* by changing its definition to expand to `\active@char{char}` (instead of `\normal@char{char}`). Conversely, the command `\bbl@deactivate` lets the active character *char* expand to `\normal@char{char}`. This command does not change the `\catcode` of the character, which stays active.

```
\textormath{text-code}{math-code}
```

Recognizing that some shorthands declared in the language definition files have to be usable in both text and math modes, this macro allows you to specify the code to execute when in text mode (*text-code*) or when in math mode (*math-code*). As explained on page 446, providing commands for use in text and math can have unwanted side effects, so this macro should be used with great care.

```
\allowhyphens    \bbl@allowhyphens
```

When L<sup>A</sup>T<sub>E</sub>X cannot hyphenate a word properly by itself—for instance, because it is a compound word or because the word contains accented letters constructed using the `\accent` primitive—it needs a little help. This help involves making L<sup>A</sup>T<sub>E</sub>X think it is dealing with two words, which appear as one word on the page. For this purpose `babel` provides the command `\allowhyphens`, which inserts an invisible horizontal skip, unless the current font encoding is T1.<sup>1</sup> In some cases one wants to insert this “help” unconditionally; for these cases `\bbl@allowhyphens` is available. This invisible skip has the effect of making L<sup>A</sup>T<sub>E</sub>X think it is dealing with two words that can be hyphenated separately.

<sup>1</sup>In contrast to the OT1 encoding, the T1 encoding contains most accented characters as real glyphs so that the `\accent` primitive is almost never used.

```
\declare@shorthand{name}{charseq}{exec}
```

The macro `\declare@shorthand` defines shorthands to facilitate entering text in the given language. The first argument, `name`, specifies the name of the collection of shorthands to which the definition belongs. The second argument, `charseq`, consists of one or more characters that correspond to the shorthand being defined. The third argument, `exec`, contains the code to be executed when the shorthand is encountered in the document. A few examples from various language definition files follow.

```
\declare@shorthand{dutch}{y}{\textormath{\ij{}}{\ddot y}}
\declare@shorthand{german}{a}{\textormath{"{a}\allowhyphens}{\ddot a}}
\declare@shorthand{french}{;}{...}
\declare@shorthand{system}{;}{\string;}
```

The latter two instructions are found in the file `frenchb.1df`, where the first handles the case where the ; character is active and the third argument provides code for ensuring that a thin space is inserted before “high” punctuation (;, :, !, and ?). The last command deals with the case where these French punctuation rules are inactivated (note that these four punctuation characters are made active in `frenchb.1df`).

## 9.6 Other approaches

In general, the `babel` package does a good job of translating document element names and making text input somewhat more convenient. However, for several languages, individuals or local user groups have developed packages and versions of `TeX` that cope with a given language on a deeper level—in particular, by better integrating the typographic traditions of the target language.

An example of such a package is `french` [51, 66], which was developed by Bernard Gaulle. Special customized versions of (I)A`TeX` exist (e.g., Polish and Czech, distributed by the `TeX` user groups GUST and `CsTUG`, respectively).

### 9.6.1 More complex languages

In the world of non-Latin alphabets, one more level of complexity is added when one wants to treat the Arabic or Hebrew [140] languages. Not only are they typeset from right to left, but, in the case of Arabic, the letter shapes change according to their positions in a word.

Several systems to handle Hebrew are available on CTAN (`language/hebrew`). In particular, `babel` offers an interface for Hebrew written by Boris Lavva. For

Arabic there is the ArabT<sub>E</sub>X system [102], developed by Klaus Lagally. This package extends the capabilities of (I<sup>A</sup>)T<sub>E</sub>X to generate Arabic writing using an ASCII transliteration (CTAN *nonfree/language/arabtex*).

Serguei Dachian, Arnak Dalalyan, and Vardan Hakobian provide Armenian support (CTAN *language/armtex*).

For the languages of the Indian subcontinent, most of the support is based on the work of Frans Velthuis. In particular, recently Anshuman Pandey developed packages for Bengali (*bengali* package and associated fonts on CTAN *language/bengali/pandey*), Sanskrit (Anshuman Pandey's *devnag* package on CTAN *language/devanagari/velthuis*), and Gurmukhi (CTAN *language/gurmukhi/pandey*).

Oliver Corff and Dorjpalam Dorj's *manjutex* package can be used for typesetting languages using the Manju (Mongolian) scripts (CTAN *language/manju/manjutex*).

Ehitopian language support, compatible with *babel*, is available through Berhanu Beyene, Manfred Kudlek, Olaf Kummer, and Jochen Metzinger's *ethiop* package and fonts (CTAN *language/ethiopia/ethiop*).

For Chinese, Japanese, and Korean (the so-called CJK scripts), one can use Werner Lemberg's *cjk* package [113], which contains fonts and utilities (CTAN *language/chinese/CJK*).

### 9.6.2 Omega

No discussion of multilingual typesetting would be complete without mentioning Omega [137], an extension of T<sub>E</sub>X developed by Yannis Haralambous and John Plaice. Omega's declared aim is to improve on T<sub>E</sub>X's multilingual typesetting abilities by making significant changes to the executable *T<sub>E</sub>X, the Program*. It potentially provides far simpler solutions in many of the areas addressed by *babel* by offering the following features:

- Omega can be used to read text files in any encoding (8-bit, 16-bit, or more).
- Omega handles shorthands internally by applying specified transformations to recognized sequences of input characters.
- Omega has an internal structure that is far more flexible than that of T<sub>E</sub>X for handling large sets of characters and large fonts.
- Omega supports many different types of script and all writing directions used for present-day scripts.

These enhancements to the T<sub>E</sub>X typesetting paradigm will make it easier to typeset a range of languages: Arabic, Bantu, Basque, Georgian, Hindi, Khmer, Chinese, Cree, or Mongolian—and all within the same document! It is also hoped (at end 2003) that enhancements to L<sup>A</sup>T<sub>E</sub>X will soon appear to support these new facilities, thus providing a fully multilingual L<sup>A</sup>T<sub>E</sub>X system.