

The datatool Bundle: Databases and Data Manipulation

Nicola L.C. Talbot

Dickimaw Books
dickimaw-books.com

version 3.4.5 2026-08-23

This document is also available as HTML (`datatool-user.html`).
The datatool bundle includes the following documentation:

User Manual for datatool (`datatool-user.pdf`)

This document is the main user guide for the datatool package.

Documented Code for datatool (`datatool-code.pdf`)

Advanced users wishing to know more about the inner workings of all the packages provided in the datatool bundle should read “Documented Code for datatool v3.4.5”.

CHANGES

Change log.

README.md

Package summary.

DEPENDS.txt

List of all packages unconditionally required by datatool (hard dependencies). Other unlisted packages may be required under certain circumstances. For help on installing packages see, for example, How do I update my T_EX distribution?¹ or (for Linux users) Updating T_EX on Linux.²

Related resources:

¹tex.stackexchange.com/questions/55437

²tex.stackexchange.com/questions/14925

- [datatool FAQ](#)³
- [Bug tracker](#)⁴
- [performance gallery](#)⁵
- [datatool-regions](#)⁶
- [datatool-english](#)⁷
- [datatooltk](#)⁸



The datatool bundle is provided to help perform repetitive commands, such as mail merging, but since $\text{\TeX}$ is designed as a typesetting language, don't expect this bundle to perform as efficiently as custom database systems or a dedicated mathematical or scripting language. **If the provided packages take a frustratingly long time to compile your document, use another language to perform your calculations or data manipulation and save the results in a file that can be input into your document.** For large amounts of data that need to be sorted or filtered or joined, consider storing your data in an SQL database and use `datatooltk` to import the data, using SQL syntax to filter, sort and otherwise manipulate the values.

³dickimaw-books.com/faq.php?category=datatool

⁴dickimaw-books.com/bugtracker.php?category=datatool

⁵dickimaw-books.com/gallery/#performance

⁶ctan.org/pkg/datatool-regions

⁷ctan.org/pkg/datatool-english

⁸ctan.org/pkg/datatooltk

Contents

List of Examples

ix

I. User Guide

1

1. Introduction

2

1.1. Rollback	4
1.2. L ^A T _E X3	4
1.2.1. Regular Expressions	4
1.2.2. Comma-Separated Lists	5
1.2.3. Calculations	6

2. Base Commands (datatool–base package)

8

2.1. datatool–base Options	8
2.2. Data Types	11
2.2.1. Numeric Options	13
2.2.2. Parsing Locale-Formatted Numbers and Currency Values	15
2.2.3. Datum Commands	16
2.2.4. Datum Items (Advanced)	20
2.3. Localisation	27
2.3.1. Encoding	31
2.3.2. Numerical	42
2.3.3. Lexicographical	46
2.3.4. Adding New Region Support	53
2.3.5. Adding New Language Support	54
2.4. Conditionals	62
2.4.1. If-Else or Case Conditionals	62
2.4.2. ifthen conditionals	89
2.5. Decimal Functions	97
2.5.1. Plain Numbers	98
2.5.2. Formatted Numbers	108
2.6. Currency	114
2.7. Dates and Times	131
2.8. Strings	138
2.8.1. Substitution and String Splitting	138
2.8.2. Initial Letters	140
2.8.3. Advanced Utility Commands	146

2.9. Comma-Separated Lists	149
2.9.1. List Settings	150
2.9.2. Formatting Lists	151
2.9.3. List Elements	153
2.9.4. Adding to Lists	155
2.9.5. Sorting Lists	156
3. Databases (datatool package)	176
3.1. Options	177
3.2. Example Databases	182
3.2.1. Student Marks (CSV)	182
3.2.2. Student Scores	184
3.2.3. Customers	187
3.2.4. Product List	191
3.2.5. Price List	194
3.2.6. Balance Sheet (CSV)	195
3.2.7. Fruit (CSV)	196
3.2.8. Profits (CSV)	197
3.2.9. Time to Growth (CSV)	199
3.2.10. Time to Growth (TSV)	201
3.2.11. Generic X/Y Data (CSV)	203
3.3. Action Command	205
3.3.1. Defined Actions	207
3.3.2. Action Settings	224
3.4. Creating a New Database	231
3.5. Deleting or Clearing a Database	235
3.6. Database Conditionals and Metadata	236
3.7. Displaying the Contents of a Database	241
3.7.1. Display Options	242
3.7.2. Associated Commands	252
3.7.3. Examples	260
3.8. Iterating Through a Database	285
3.8.1. Iterating Over Rows with \DTLmapdata	285
3.8.2. Iterating Over Rows with \DTLforeach	293
3.9. Loops and Conditionals with tabular-like Environments	303
3.10. Null Values	310
3.10.1. Examples	311
3.10.2. Advanced Commands	315
3.11. Special Values	317
3.12. Editing Database Rows	318
3.13. Arithmetical Computations on Database Entries	321
3.14. Sorting a Database	325
3.14.1. Sorting with \DTLsortdata	325
3.14.2. Sorting with \dtlsort	334

3.15. Database Files (I/O)	336
3.15.1. File Formats	336
3.15.2. I/O Settings	346
3.15.3. Loading a Database from an External File	360
3.15.4. Saving a Database to an External File	364
3.15.5. I/O Examples	365
3.16. Advanced Database Commands	369
3.16.1. Operating on Current Row	371
3.16.2. Advanced Iteration	374
4. Pie Charts (datapie package)	377
4.1. Package Options	380
4.2. Settings	380
4.2.1. Pie Chart Data	381
4.2.2. Pie Chart Style	381
4.2.3. Pie Chart Labels	383
4.3. Pie Chart Examples	385
4.3.1. Pie Chart Filtering Examples	386
4.3.2. Pie Chart Styles Examples	387
4.3.3. Pie Chart Labels Examples	389
4.3.4. Pie Chart Placeholder Example	391
4.3.5. Pie Chart Label Formatting Example	391
4.3.6. Pie Chart Colour Example	393
4.4. Pie Chart Variables	394
4.5. Pie Chart Label Formatting	395
4.6. Pie Chart Colours	396
4.7. Pie Chart Hooks	397
5. Bar Charts (databar package)	399
5.1. Package Options	406
5.2. Settings	406
5.2.1. Bar Chart Data	408
5.2.2. Bar Chart Style	409
5.2.3. Bar Chart Labels	414
5.2.4. Bar Chart Axes	416
5.3. Bar Chart Examples	420
5.3.1. Labels	420
5.3.2. Filtering	423
5.3.3. Lower Label Alignment	424
5.3.4. Upper Label Alignment	427
5.3.5. Multi Bar Chart Labels	427
5.3.6. Axes	429
5.3.7. Bar Colours	431
5.3.8. Hooks	436

5.4.	Bar Chart Associated Commands	441
5.4.1.	Axes	441
5.4.2.	Textual	442
5.4.3.	Bar Colours	446
5.4.4.	Hooks	448
6.	Scatter and Line Plots (dataplot package)	451
6.1.	Plot Settings	453
6.1.1.	Database Content	454
6.1.2.	Plot Size	455
6.1.3.	Marker and Line Styles	457
6.1.4.	Axes	461
6.2.	Plot Examples	472
6.2.1.	Basic Examples	473
6.2.2.	Legend Examples	480
6.2.3.	Plot Style Examples	490
6.2.4.	Plot Axes Examples	495
6.2.5.	Begin and End Hooks	509
6.3.	Supplementary Plot Commands	510
6.3.1.	General Functions and Variables	511
6.3.2.	Stream Functions and Variables	512
6.3.3.	Post-Stream Hooks	514
6.4.	Conditionals	518
6.4.1.	Lengths	520
6.4.2.	Counters	521
6.4.3.	Macros	521
6.5.	Adding to a Plot Stream at the Start or End	524
7.	Converting a BibT_EX database into a datatool database (databib package)	526
7.1.	Package Options	527
7.2.	BibT _E X: An Overview	527
7.2.1.	BibT _E X database	528
7.2.2.	Column Keys (Fields)	531
7.3.	Loading a databib database	536
7.4.	The databib Database Construction Commands	537
7.5.	Sorting a databib database	538
7.6.	Displaying a databib database	540
7.7.	Changing the bibliography style	542
7.7.1.	Modifying an existing style	543
7.8.	Iterating through a databib database	549
7.9.	Multiple Bibliographies	552
7.10.	Examples	553
7.10.1.	Sort by Author	559
7.10.2.	Tabulate Bib Data	560

7.10.3. Publications Since a Given Year	562
7.10.4. Five Most Recent Publications	564
7.10.5. Compact Bibliography	566
7.10.6. Highlight a Given Author	567
7.10.7. Separate Bib Types	569
7.10.8. Multiple Bibliographies	572
7.11. Localisation	574
8. Creating an index, glossary or list of abbreviations (datagidx package)	577
8.1. Options	578
8.1.1. Package Options	578
8.1.2. Post-Package Options	581
8.2. Defining Index/Glossary Databases	587
8.3. Loading Data Created by <code>datatooltk</code>	588
8.4. Defining Terms	588
8.4.1. Markup Commands for Terms	591
8.4.2. Commands to Assist Label Creation	596
8.4.3. Commands to Assist Sorting	598
8.5. Referencing Terms	599
8.5.1. Shortcut Commands	602
8.5.2. Locations	603
8.6. Adding Extra Fields	604
8.7. Abbreviations	606
8.7.1. Using Abbreviations	607
8.7.2. Unsetting and Resetting Abbreviations	609
8.8. Displaying the Index or Glossary	609
8.8.1. Hooks and Associated Commands	611
8.8.2. Index or Glossary Styles	615
8.8.3. Sorting the Index or Glossary Database	616
8.9. Supplementary Commands	618
8.9.1. Conditionals and Loops	618
8.9.2. New Terms	620
8.9.3. Styles	621
8.10. Database Structures	622
8.10.1. The Catalogue Database	622
8.10.2. The Term Databases	623
8.11. Examples	624
9. Referencing People (person package)	628
9.1. Package Options	629
9.2. Other Options	629
9.3. Defining and Undefining People	630
9.4. Genders	633

Contents

9.5. Displaying Information	636
9.5.1. Accessing Individual Information	643
9.5.2. List People	645
9.6. Examples	646
9.6.1. Mail Merging	646
9.6.2. Order of Service	648
9.7. Advanced Commands	653
9.7.1. Conditionals	654
9.7.2. Iterating Through Defined People	656
9.7.3. Localisation	657
9.7.4. Hooks	660
10. Acknowledgements	662
 II. Summaries and Index	 663
Symbols	664
Glossary	665
Command Summary	667
Command Summary: A	667
Command Summary: C	668
Command Summary: D	669
Command Summary: E	776
Command Summary: F	777
Command Summary: G	778
Command Summary: H	781
Command Summary: I	782
Command Summary: L	785
Command Summary: M	800
Command Summary: N	801
Command Summary: P	802
Command Summary: R	817
Command Summary: S	817
Command Summary: T	818
Command Summary: U	820
Command Summary: V	821
Command Summary: X	821
Command Summary: Y	822
 Environment Summary	 823

Contents

Package Option Summary	824
Index	828

List of Tables

- 3.1. Mappings Used with `csv-content=literal` Before Re-Scanning . . . 351
- 9.1. Synonyms provided by the `shortcuts` package option 637

List of Examples

If an example shows the icon  then the source code is embedded in the PDF as an attachment. If your PDF viewer supports attachments, you can extract the self-contained example file to try it out for yourself. Alternatively, you can click on the download icon  which will try downloading the example source code from your closest CTAN mirror, but make sure that this user manual matches the version on CTAN first. You can also try using:

```
texdoc -l datatool-user-example<nnn>
```

where *<nnn>* is the example number zero-padded to three digits to find out if the example files are installed on your device.

1.	Regular Expressions with $\text{\LaTeX}$ 3	5
2.	Comma-Separated Lists with $\text{\LaTeX}$ 3	6
3.	Performing Calculations with $\text{\LaTeX}$ 3	7
4.	Performing Calculations with LuaLaTeX	7
5.	Datum Control Sequences	20
6.	Datum Tests for Equality	25
7.	Datum Control Sequences to Floating Point Variables	26
8.	Localisation Support (en-CA)	29
9.	Localisation Support (fr-CA)	30
10.	Icelandic Alphabetic	49
11.	IJ-Initial Support	51
12.	Icelandic Sorting and Letter Groups	52
13.	Test for Integer Value	65
14.	Test for Real Value	66
15.	Test for Currency	67
16.	Test for Numerical	68
17.	Test for Strings	68
18.	Test for Data Type	69
19.	Testing if an Element is in a Comma-Separated List	72
20.	String Equality Tests	74
21.	String Less Than	74
22.	String Greater Than	75
23.	String Between Tests	75
24.	Substring Tests	77
25.	Prefix Tests	78

List of Examples

26.	Suffix Tests	80
27.	All Upper/Lower Case Tests	81
28.	Numerical Comparisons (Parsed)	82
29.	Conditionals (l3fp)	85
30.	Conditionals (lua)	86
31.	Conditionals (fp)	86
32.	Conditionals (pgfmath)	87
33.	Numerical/String Comparisons	89
34.	Data Type Conditionals for use with ifthen	95
35.	Order Conditionals for use with ifthen	96
36.	Substring Conditionals for use with ifthen	97
37.	Decimal Functions (l3fp)	105
38.	Decimal Functions (lua)	106
39.	Decimal Functions (fp)	107
40.	Decimal Functions (pgfmath)	109
41.	Formatting and Parsing Currency (No Region)	117
42.	Currency Formats (GB Region)	120
43.	Currency Formats (GB and IE Regions)	123
44.	Defining a Currency	131
45.	Parsing Dates and Times	136
46.	Parsing Dates and Times and Reformatting	137
47.	String Substitution and Splitting	140
48.	Name or Phrase Initials	144
49.	Word Initial Letter with UTF-8	145
50.	Word Initial Commands	146
51.	CSV List Argument Expansion	150
52.	Formatting CSV Lists	153
53.	Elements of a CSV List	155
54.	Appending, Prepending and Inserting List Elements	156
55.	Sorting Lists with \dtlsortlist (Case vs No Case)	169
56.	Sorting Lists with \dtlsortlist (Letter vs Word)	169
57.	Sorting Lists with \dtlsortlist (comma and parenthetical markers)	171
58.	Sorting Lists with \DTLsortwordlist (comma and parenthetical markers)	172
59.	Sorting Lists with \dtlsortlist and UTF-8	173
60.	Sorting Lists with \DTLsortwordlist and UTF-8 and No Localisation Support	174
61.	Sorting Lists with \DTLsortwordlist and UTF-8 and Localisation Support	174
62.	Sort Word Hook (Roman Numerals)	175
63.	New Value Expansion	179
64.	Trimming New Values	181
65.	Creating and Displaying a Database with \DTLaction	205
66.	Adding New Columns Using Actions	211
67.	Select row action	216
68.	Row aggregate actions	222

List of Examples

69.	Automatically Formatting Values Calculated by Actions	231
70.	Creating a New Database with a Label	235
71.	Column with No Values	241
72.	Display Data with Custom Alignment	261
73.	Display Data in a Table Omitting Columns	262
74.	Display Data in a Table with Named Columns	263
75.	Display Data in a Table with Filtered Rows	264
76.	Referencing Rows from Displayed Data	267
77.	Inserting a Column at the Start of Displayed Data	268
78.	Display Data in a Table with an Extra Column	270
79.	Adjusting the Item Hook to Calculate Totals and Show Negative Numbers in Red	273
80.	Display Two Database Rows Per Tabular Row	274
81.	Display Two Database Rows Per Tabular Row (Top to Bottom)	275
82.	Display Data in a Stripy Table	277
83.	Display Stripy Two Database Rows Per Tabular Row	278
84.	Display Two Fields in One Column	281
85.	Displaying Data with Calculations, Filtering and Row Highlighting	284
86.	Iterating Over Rows with \DTLmapdata and DTLenvmapdata	287
87.	Iterating Over Rows with \DTLmapdata to Append a Column	293
88.	Display Data in a Table with \DTLforeach	299
89.	Using \DTLforeach to Display a Stripy Table	300
90.	Displaying Data with Row Numbers Using \DTLforeach	301
91.	Using \DTLforeach to Display Data in a Table with a Running Total Column	302
92.	Editing a Database with \DTLforeach	303
93.	Loops and Alignment	310
94.	CSV Data Containing Empty Cells and Missing Final Cells	312
95.	Constructed Data With Missing (Null) Values	313
96.	Display Data With Missing (Null) Values Shown as a Dash	314
97.	Iterating Through Data with Empty or Missing Values	316
98.	Editing a Row of Data	321
99.	Sorting CSV Data Using \DTLsortdata by Organisation, Surname and Forename With No Replacements	329
100.	Sorting CSV Data Using \DTLsortdata by Organisation, Surname and Forename With Replacements	330
101.	Sorting Data Using \DTLsortdata With Replacements (Null vs Empty)	331
102.	Sorting CSV Data Using \DTLsortdata With Language Support	332
103.	Sorting Data Using \DTLsortdata on Age then Surname (Empty or Null Values)	332
104.	Sorting Data Using \DTLsortdata on Age then Surname (No Empty Sort Values)	333
105.	Sorting Data Using \DTLsortdata by Descending Numeric and Ascending String Values	334
106.	Sorting CSV Data Using \dtlsort by Organisation, Surname and Forename With Replacements	336

List of Examples

107. Loading Data With No Parsing	361
108. Loading Data With No Parsing and Columns Identified as Decimal	362
109. Loading Data With No Parsing and Columns Identified as Decimal and Currency	362
110. Loading Data With No Parsing and Columns Identified as Decimal and Currency with Reformatting	363
111. Loading and Saving Data (Be Careful of Category Codes)	367
112. Loading a TSV File	368
113. Automatically Reformatting Data While Loading a CSV file	369
114. Pie Chart	379
115. Pie Chart (Action ‘pie chart’)	379
116. Pie Chart (Filtering)	386
117. Separating Segments from a Pie Chart	388
118. Separating a Range of Segments from a Pie Chart	389
119. Separating Individual Consecutive Segments from a Pie Chart	390
120. Pie Chart (Inner and Outer Labels)	391
121. Pie Chart (Labels Rotated)	392
122. Pie Chart (Percentage Rounding)	392
123. Pie Chart (Changing the Label Format)	393
124. Pie Chart (Changing and Referencing the Segment Colours)	395
125. Vertical Bar Chart	402
126. Vertical Bar Chart (Action ‘bar chart’)	403
127. Horizontal Bar Chart	404
128. Multi Bar Chart	404
129. Multi Bar Chart (Action ‘multibar chart’)	405
130. Bar Chart With Labels	421
131. Bar Chart With Labels (Action ‘bar chart’)	422
132. Bar Chart (Filtering)	423
133. Horizontal Bar Chart with Labels (Default Alignment)	425
134. Horizontal Bar Chart with Labels (lower-label-style=same)	425
135. Horizontal Bar Chart with Labels (lower-label-style=below)	426
136. Horizontal Bar Chart with Labels (lower-label-style=above)	426
137. Horizontal Bar Chart with Upper Labels Over the Bars (negative upper- label-offset)	428
138. Multi Bar Chart With Group Labels	429
139. Bar Chart With Axes	430
140. Bar Chart With Rotated Tick Labels	432
141. Bar Chart With a Limited Set of Custom Colours	433
142. Bar Chart Cycling through the Colour Set	434
143. Single Colours for Positive and Negative Bars	435
144. Shaded Bar	436
145. Hook at Every Bar	437
146. Every Bar Hook (Filtering)	438
147. Bar Chart With a Legend	440
148. Multi Bar Chart With a Legend	441

List of Examples

149. Scatter Plot (One Database)	474
150. Scatter Plot (Two Databases)	474
151. Scatter Plot (Action)	476
152. Scatter Plot (One Database, Two Sets of Data)	477
153. Scatter Plot (Two Databases, Two Sets of Data)	478
154. Scatter Plot (Two Databases, Multiple Sets of Data)	479
155. Scatter Plot With Mismatched X and Y Columns	480
156. Scatter Plot with Custom Legend Labels (One Database, Two Sets of Data)	481
157. Scatter Plot with Custom and Default Legend Labels (One Database, Two Sets of Data)	482
158. Scatter Plot with an Omitted Legend Label (One Database, Two Sets of Data)	483
159. Scatter Plot (Two Databases with Name Map)	485
160. Scatter Plot with Legend Label Mappings (Two Databases, Multiple Sets of Data)	486
161. Scatter Plot with Legend Label Mappings and Custom formatting (Two Databases, Multiple Sets of Data)	486
162. Scatter Plot with Custom Legend Labels (Two Databases, Multiple Sets of Data)	488
163. Scatter Plot with Shifted Legend (Two Databases, Multiple Sets of Data)	489
164. Scatter Plot with Custom Legend (Two Databases, Multiple Sets of Data)	490
165. Line and Scatter Plot (Two Databases)	491
166. Scatter Plot with Custom Colours and Styles (Two Databases, Multiple Sets of Data)	492
167. Scatter Plot with the Same Line Colour for Each Stream in a Given Database (Two Databases, Multiple Sets of Data)	494
168. Scatter Plot with Plot Marks Reset (Two Databases, Multiple Sets of Data)	495
169. Setting the Plot Bounds	496
170. Rounding the Tick Labels	497
171. Changing the Axis Style	498
172. Grid	499
173. Custom Grid Lines	500
174. Plot Encapsulated in a Box	501
175. Plot Encapsulated in a Box Without Ticks	502
176. Positive and Negative Axes	503
177. Extending the Axes	504
178. Changing the Tick Label Node Style	505
179. Side Axes	506
180. Side-Axes, Extended Axes and Boxed	507
181. No Side-Axes, Extended Axes and Boxed	508
182. Redefining the Start and End Hooks	510
183. Bibliography Sorted by Author	561
184. Tabulate Bib Data	563
185. List of Publications Since a Given Year	564
186. Five Most Recent Publications	565
187. Compact Bibliography	568
188. Highlighting a given author	570

List of Examples

189. Separate List of Journals and Conference Papers	572
190. Multiple Bibliographies	575
191. Creating an Index	626
192. Creating a List of Abbreviations	627
193. Mail Merging	648
194. Memorial Order of Service	649
195. Memorial Order of Service (Shortcuts)	650
196. Baptism Order of Service	651
197. Baptism Order of Service (Shortcuts and Localisation)	653

Part I.

User Guide

1. Introduction

The following packages are provided by the datatool bundle:

- **datatool-base** This is the underlying package automatically loaded by all the other listed packages, but may be loaded without the other packages if only the base functions are required. The datatool-base package may be used to:
 - Determine whether an argument is an integer, a real number, currency or a string. Locale dependent number settings are supported (such as a comma as a decimal character and a full stop as a number group character). As from version 3.0, scientific notation is also supported.
 - Convert locale dependent numbers or currency to plain number format, enabling arithmetic to be performed on elements of the database.
 - Names can be converted to initials.
 - Determine if strings are all upper or lower case.
 - Perform string comparisons (both case sensitive and case insensitive).

See §2.

- **datatool**
Main package providing database support. Automatically loads datatool-base. This package can be used to:
 - Create or load databases.
 - Sort rows of a database (either numerically or alphabetically, ascending or descending).
 - Perform repetitive operations on each row of a database (e.g. mail merging). Conditions may be imposed to exclude rows.

Database commands are described in §3.

- **datagidx**
The datagidx package (see §8) can be used to generate indexes or glossaries as an alternative to packages such as glossaries. Note that datagidx is far more limited than glossaries and doesn't provide any localisation support. See §8.

- **datapie**

The datapie package can be used to convert a database into a pie chart:

- Segments can be separated from the rest of the chart to make them stand out.

1. Introduction

- Colour/grey scale options.
- Predefined segment colours can be changed.
- Hooks provided to add extra information to the chart

See §4.

- **dataplot**

The dataplot package can be used to convert a database into a two dimensional plot using markers and/or lines. Three dimensional plots are currently not supported. See §6.

- **databar**

The databar package can be used to convert a database into a bar chart:

- Colour/grey scale options.
- Predefined bar colours can be changed.
- Hooks provided to add extra information to the chart

See §5.

- **databib**

The databib package can be used to convert a B_BT_EX database into a datatool database. See §7.

- **person**

The person package can be used to reference people by the appropriate gender pronouns. Automatically loads datatool. See §9.



The datapie and databar packages do not support the creation of 3D charts, and I have no plans to implement them at any later date. The use of 3D charts should be discouraged. They may look pretty, but the purpose of a chart is to be informative. Three dimensional graphics cause distortion, which can result in misleading impressions. The pgf manual provides a more in-depth discussion on the matter.

The code providing the mathematical functions have some limitations. These limitations will therefore also be present in the various packages provided with datatool, according to the underlying package (fp or pgfmath) or L^AT_EX3 kernel commands or Lua code used. As from version 3.0, the new default is lua, if `\directlua` is defined, or `l3fp` otherwise. To avoid repeated parsing, some functions, such as the aggregate functions (§3.13) or charts (§§4, 6 & 5), will use L^AT_EX3 commands regardless of the `math` option.

1.1. Rollback

Version 3.0 is a major new version where many commands have been rewritten to use L^AT_EX3 macros. Additionally, some packages, such as xkeyval and substr are no longer loaded. If you experience any backward-compatibility problems with the new version, you can rollback to the previous version (2.32):

```
\usepackage{datatool}[=v2.32]
```

Rollback provides a useful way of reverting back to an earlier release if there's a problem with a new version. However, the further away the rollback date is from the current L^AT_EX kernel, the more likely that incompatibilities will occur. If you have historic documents that you need to compile, consider using the historic T_EX Live Docker images. (See, for example, Legacy Documents and T_EX Live Docker Images.^a)

^adickimaw-books.com/blog/legacy-documents-and-tex-live-docker-images

1.2. L^AT_EX3

The L^AT_EX kernel has changed significantly since datatool was first released in 2007. There is now improved support for UTF-8 and many of the commands provided by datatool now have much better L^AT_EX3 alternatives. You may find some tasks more efficient if you use L^AT_EX3 commands directly. However, L^AT_EX3 commands are intended for internal use within the definitions of document commands rather than explicit use in the document.

L^AT_EX3 syntax must first be switched on (`\ExplSyntaxOn`) before defining commands that use them and then switched off (`\ExplSyntaxOff`) afterwards. Spaces are ignored, so you need to use `~` if an actual space is required. Further information can be found in the `interface3.pdf` document:

```
texdoc interface3
```

1.2.1. Regular Expressions

L^AT_EX3 provides commands for regular expressions. A simple example is shown below that replaces `\emph{boo}` with `\textbf{BOO}`. More generally, the custom command searches for any instance of `\emph{<word>}`, where `<word>` consists of one or more word characters (`\w+`), and replaces it with `\textbf{<WORD>}`, where the argument is the original `<word>` converted to uppercase using `\text_uppercase:n`.

1


```

\documentclass{article}
\ExplSyntaxOn
\NewDocumentCommand{\testreplace} { m }
{
  \regex_replace_all:nnN
    { \c{emph} \cB\{ (\w+) \cE\} }
    { \c{textbf} { \c{text_uppercase:n}{ \1 } } }
    #1
}
\ExplSyntaxOff
\begin{document}
\newcommand{\teststring}{The duck said \emph{boo}
to the goose.}
Original: \teststring

\testreplace{\teststring}
Replaced: \teststring
\end{document}

```

↑ Example 1: Regular Expressions with L^AT_EX3



Original: The duck said *boo* to the goose.

Replaced: The duck said **BOO** to the goose.

1.2.2. Comma-Separated Lists

L^AT_EX3 provides commands for dealing with CSV lists. You may prefer to use those instead of the commands provided by `datatool-base` described in §2.9. 

```

\documentclass{article}
\ExplSyntaxOn
\clist_new:N \l_my_clist
\NewDocumentCommand \createmylist { m }
{
  \clist_set:Nn \l_my_clist { #1 }
}
\NewDocumentCommand \mylistelement { m }
{

```

```

\clist_item:Nn \l_my_clist { #1 }
}
\NewDocumentCommand \reversemylist { }
{
  \clist_reverse:N \l_my_clist
}
\NewDocumentCommand \displaymylist { }
{
  \clist_use:Nnnn \l_my_clist {~and~ } { ,~ }
{ ,~and~}
}
\ExplSyntaxOff
\begin{document}
\createmylist{ant,duck,goose,zebra}
\displaymylist

Second element: \mylistelement{2}.

\reversemylist
\displaymylist

Second element: \mylistelement{2}.
\end{document}

```

↑ Example 2: Comma-Separated Lists with L^AT_EX3



```

ant, duck, goose, and zebra
Second element: duck.
zebra, goose, duck, and ant
Second element: goose.

```

1.2.3. Calculations

If you have complex calculations, you may prefer to use L^AT_EX3 commands directly instead of using the `datatool`-base commands described in §2.5.1.



```

\documentclass{article}
\ExplSyntaxOn
\newcommand{\myfunc} [3]
{

```



```

\fp_to_decimal:n{ #1 + 0.5 * sqrt(#2) / (#3) }
}
\ExplSyntaxOff
\newcommand{\numA}{1023.5}
\newcommand{\numB}{54.75000}
\newcommand{\numC}{-20648.68}
\begin{document}
$ \numA+\frac{\sqrt{\numB}}{2\times\numC} =
\myfunc{\numA}{\numB}{\numC} $
\end{document}

```



↑ Example 3: Performing Calculations with L^AT_EX3



$$1023.5 + \frac{\sqrt{54.75000}}{2 \times -20648.68} = 1023.499820828152$$

If you plan on re-parsing commands such as the example `\numA`, `\numB` and `\numC` commands, then it would be better to convert them to L^AT_EX3 floating point variables or constants. See the L^AT_EX3 Interfaces document for further details.

Example 4 performs the same calculation but uses `\directlua`, which requires LuaL^AT_EX:

4

```

\newcommand{\myfunc}[3]{%
\directlua{tex.print(#1+0.5*math.sqrt(#2)/(#3))}%
}

```



↑ Example 4: Performing Calculations with LuaL^AT_EX



$$1023.5 + \frac{\sqrt{54.75000}}{2 \times -20648.68} = 1023.4998208282$$

2. Base Commands (datatool–base package)

```
\usepackage[<options>]{datatool–base}
```

The datatool–base package may be loaded on its own, without the datatool package, if no database commands (see §3) are required. Available package options for datatool–base are listed below.

2.1. datatool–base Options

Options can be passed through the package option list in the usual way. Some options may also be later set with:

```
\DTLsetup{<key=value list>}
```

(Options specific to locale files should be set with `\DTLsetLocaleOptions`, see §2.3.)

```
math=<processor>
```

initial: varies

This setting may only be used as a package option, not in `\DTLsetup`, and identifies the required maths processor. This determines how the floating point commands described in §2.5 are defined. The value may be one of the following.

```
math=13fp
```

This setting defines the datatool–base floating point commands (such as `\dtladd`) to use $\text{\LaTeX 3}$ commands. This is the default setting unless $\text{\LaTeX}$ is used.

```
math=lua
```

This setting defines the datatool–base floating point commands (such as `\dtladd`) to use `\directlua` to perform the mathematical calculations. This is the default setting if $\text{\LaTeX}$ is used.

2. Base Commands (*datatool*–base package)

math=fp

This setting defines the *datatool*–base floating point commands (such as `\dtladd`) to use the *fp* package commands to perform the mathematical calculations. (Automatically loads the *fp* package.) Note that the *fp* package can be less precise than *L^AT_EX*3 or Lua. (See examples 37, 39 & 38.)

math=pgfmath

This setting defines the *datatool*–base floating point commands (such as `\dtladd`) to use the *pgfmath* package commands to perform the mathematical calculations. (Automatically loads the *pgfmath* package.) Note that the *pgfmath* package has limitations and may produce the error:

! Dimension too large

This option is maintained for backward-compatibility but, in general, the new default `l3fp` or `lua` options are better.

As from version 3.0, some functions, such as the aggregate functions (§3.13) or charts (§§4, 6 & 5), will use *L^AT_EX*3 commands regardless of the **math** option to avoid repeated parsing.

verbose=*<boolean>*

default: **true**; initial: **false**

If true, this option will write extra informational messages to the transcript.

lang-warn=*{ <boolean> }*

default: **true**; initial: **true**

This setting may only be used as a package option, not in `\DTLsetup`. If false, this setting switches off localisation warnings. Note that this will also switch off *tracklang* warnings. If true, this setting will switch on *datatool*–base localisation warnings without altering *tracklang* warnings. If you need *tracklang* warnings to be switched back on again for the next package that requires it, use `\TrackLangShowWarningstrue`.

nolocale

This setting may only be used as a package option, not in `\DTLsetup`, and has no value. If used it will prevent any localisation files from being loaded, regardless of the document language settings. This option will override *locales* (and *lang*). See §2.3.

locales= $\{\langle locale\ list\rangle\}$

This setting may only be used as a package option, not in `\DTLsetup`. It counteracts the effect of `no locale` and tracks each listed language tag using tracklang's `\TrackLanguageTag`. Note that localisation support must be installed separately. See §2.3.

This option will have an effect on packages that are subsequently loaded that also use tracklang. Note that multiple instances of this option override each other.

lang= $\{\langle locale\ list\rangle\}$

alias: **locales**

A synonym of `locales`.

initial-purify= $\langle value\rangle$

initial: **early**

This boolean setting indicates whether or not to purify the $\langle text\rangle$ argument of `\DTLGetInitialLetter` before parsing.

auto-reformat-types= $\langle list\rangle$ initial: **integer, decimal, si, currency, datetime, date, time**

This option takes a comma-separated list, where the items in the list may be any of the following keywords: `integer`, `decimal`, `si`, `currency`, `datetime`, `date`, `time`. This identifies which data types should be automatically reformatted if the corresponding `auto-reformat numeric` option or `auto-reformat datetime` option is on.

The `auto-reformat-types` does not switch on the corresponding `auto-reformat numeric` option or `auto-reformat datetime` option. It simply establishes which data types should be affected when the applicable option is on.

For example:

```
\DTLsetup{
  auto-reformat-types={decimal, si, datetime},
  numeric={auto-reformat},
  datetime={parse=auto-reformat}
}
```

In the above, if `\DTLparse` identifies a decimal or SI notation (but not an integer) or a datetime (but not a date or a time) then the string value will be automatically reformatted.



If `auto-reformat-types` is missing all numeric types, then the `auto-reformat` numeric option will have no effect. Similarly, if `auto-reformat-types` is missing all temporal types, then the `auto-reformat` datetime option will have no effect.



lists=*<key=value list>*

default: **true**

This setting may be used to adjust the behaviour of commands that deal with lists. The value should be a *<key>=<value>* list of options, which are described in §2.9.1.



compare=*<key=value list>*

default: **true**

This setting may be used to adjust the behaviour of commands that deal with comparisons. The value should be a *<key>=<value>* list of options, which are described in §2.9.5.1.



numeric=*<key=value list>*

default: **true**

This setting may be used to adjust the behaviour of commands that deal with numeric (but not temporal) values. The value should be a *<key>=<value>* list of options, which are described in §2.2.1.



datetime=*<key=value list>*

default: **true**

This determines whether or not commands such as `\DTLparse` should also try parsing for timestamps (date and time), dates (no time) or times (no date). The temporal data types were only added to *datatool-base* version 3.0 and are still experimental so this feature is off by default. The value should be a *<key>=<value>* list of options, which are described in §2.7.

2.2. Data Types

The *datatool-base* package recognises the following data types:

Integers

An integer is a sequence of digits, optionally groups of three digits may be separated by the number group character. The default number group character is a comma (,) but may be changed using `\DTLsetnumberchars`. Examples: 1,234 (which has the default number group character) and 1234 (which is also a plain number) but not 1234.0 (which is a decimal). A double sign (such as ++1 234 or --1 234) isn't permitted and will be treated as a string.

2. Base Commands (*datatool-base package*)

If a whole number is represented in scientific notation (for example, $1\text{e}+4$ instead of 1000) then it will be identified as a decimal not an integer. Otherwise, a large integer will be considered a string (otherwise it will trip $\text{T}_{\text{E}}\text{X}$'s integer limit).

Real Numbers (Decimals)

A real number is a sequence of digits as per integers followed by the decimal character followed by one or more digits. The number group character is only recognised before the decimal character. The decimal character is a full stop “decimal point” by default. The number group and decimal characters may be changed using `\DTLsetnumberchars`. Examples: 1,234.0 (which has the default number group character and decimal character), 1234.0 (which is also a plain number) but not 1234 (which is an integer). A double sign (such as $++1234.0$ or $-+1234.0$) isn't permitted and will be treated as a string.

As from version 3.0, scientific notation, such as $2.5\text{e}+10$ or $1\text{E}-5$ is supported. Note that the locale symbols aren't supported when parsing for scientific notation. The format should be $\langle mantissa \rangle \text{E} \langle exponent \rangle$ or $\langle mantissa \rangle \text{e} \langle exponent \rangle$. A space may occur between the mantissa and the E/e. The exponent must be an integer. The mantissa may include a decimal point. If the `auto-reformat` setting is on, parsed scientific notation will have the value encapsulated with `\DTLscinum`.

Currency

The parser recognises currency values if provided in one of the following forms: `\DTLcurrency{<num>}`, `\DTLfmtcurrency{<sym>}{<num>}` `\DTLfmtcurr{<currency-code>}{<num>}` or `<sym><num>` where `<sym>` is a recognised currency symbol (identified with `\DTLnewcurrencysymbol`) and `<num>` is an integer or decimal using the current number group character and decimal character (not scientific notation). The sign may occur before the currency symbol. Some regional localisation files will also recognise currency where the symbol is prefixed with the region's code.

Examples: $\$1,234.56$ and `\pounds1234` (which both have a recognised currency symbol) and `\DTLfmtcurrency{£}1,234` or `\DTLcurrency{1,234.00}` (which both use known currency formatting commands) but not “1,234 USD” (which doesn't fit the recognised format). Both `-\pounds1234` and `\pounds-1234` are recognised as a currency with a negative numeric value.

Additionally, `\DTLfmtcurr{GBP}1,234` will also be recognised as currency (although it requires the `datatool-GB.ldf` region file to be loaded in order to correctly format the value). If `datatool-GB.ldf` has been loaded, then $\text{GB£}1,234$ will also be recognised. However, if, say, `datatool-IE.ldf` has been loaded, then $\text{IE€}1,234$ *won't* be recognised as that region doesn't support a currency prefix. See §2.6.

Temporal Values (Dates and Times) New to version 3.0 and still experimental. ISO dates and times can be parsed (if enabled with `parse`) and converted into a numerical form so that they can be treated as numbers.



If temporal parsing is off or the format is unsupported, dates and times will be treated as strings. Regional formats can only be supported if they have been defined in a loaded region file. See §2.3.

There are three temporal types:

1. Dates are in the form $\langle YYYY \rangle - \langle MM \rangle - \langle DD \rangle$ where $\langle YYYY \rangle$ is the year, $\langle MM \rangle$ is the two digit month and $\langle DD \rangle$ is the two digit day. The numeric value is the integer JDN.
2. Times are in the form $\langle hh \rangle : \langle mm \rangle : \langle ss \rangle$ or $\langle hh \rangle : \langle mm \rangle$ where $\langle hh \rangle$ is the two digit 24 hour, $\langle mm \rangle$ is the two digit minute, and $\langle ss \rangle$ is the two digit second (“00” if omitted). The numeric value is the JF.
3. Timestamps include both a date and time. If the time zone is missing, UTC+0 is assumed. Recognised formats:

$$\begin{aligned} &\langle YYYY \rangle - \langle MM \rangle - \langle DD \rangle \text{T} \langle hh \rangle : \langle mm \rangle : \langle ss \rangle \langle TZh \rangle : \langle TZm \rangle \\ &\langle YYYY \rangle - \langle MM \rangle - \langle DD \rangle \text{T} \langle hh \rangle : \langle mm \rangle : \langle ss \rangle \text{Z} \\ &\langle YYYY \rangle - \langle MM \rangle - \langle DD \rangle \text{T} \langle hh \rangle : \langle mm \rangle : \langle ss \rangle \end{aligned}$$

Where $\langle TZh \rangle$ is the time zone hour and $\langle TZm \rangle$ is the time zone minute. A space may also be used instead of “T” as the separator between the date and time. The corresponding numeric value is the JD, which is the integer JDN plus the fractional JF.

Strings

Any non-blank content that doesn’t belong to the above types is considered to be a string. See §2.8.

Unknown

Blank values are classified as an unknown type. This may be the result of an empty element in a CSV list or file.

Null

Values that are missing (not simply empty) are considered null values. This is similar in concept to `\c_novalue_tl` but uses a different internal marker. See §3.10 for further details.

2.2.1. Numeric Options

The options listed here govern parsing and formatting of localised integers, decimals and currency. The options may be passed in the value of the `numeric` option. For example:

```
\DTLsetup{
  numeric={
    auto-reformat,
    region-currency-prefix=smallcaps
  }
}
```

`auto-reformat`=*<boolean>*

default: **true**; initial: **false**

Determines whether or not commands like `\DTLparse` should reformat the string part for integers, decimals and currency. (According to the `auto-reformat-types` setting.)

This option has no effect with `csv-content=no-parse` as the values aren't parsed. Use `convert-numbers` instead.

If `auto-reformat-types` includes the keyword `integer`, then any integers will be reformatted according to the current localisation settings. If `auto-reformat-types` includes the keyword `decimal`, then any decimals not in scientific notation will be reformatted according to the current localisation settings.

If `auto-reformat-types` includes the keyword `si`, then any scientific notation, will be have the string part set to

`\DTLscinum{<value>}`

If `siunitx` is loaded, this will be defined to use `\num` otherwise it will simply expand to its argument.

If `auto-reformat-types` includes the keyword `currency`, then currency will be reformatted to use `\DTLfmtcurr`, if the associated currency code can be determined, or to `\DTLfmtcurrency` otherwise.

`region-currency`=*<boolean>*

default: **true**; initial: **true**

Determines whether or not the region hook should change the default currency. The region files should provide a command called `\datatool<Region>SetCurrency` which checks this boolean value before setting the default currency.

Note that if the region hook has already set the default currency, this option won't undo that. It can only prevent the change the next time the hook is used (for example, when the

document language changes).

`currency-symbol-style=<value>`

initial: **symbol**

This option simply redefines `\DTLcurrCodeOrSymOrChar` to expand to its first argument (`iso`) or second argument (`symbol`) or third argument (`string`).

`set-currency=<currency-code>`

Essentially this is like doing:

```
\DTLsetdefaultcurrency{<currency-code>}
\DTLsetup{region-currency=false}
```

However, unlike `\DTLsetdefaultcurrency` the value `<currency-code>` must be a defined currency code.

`region-currency-prefix=<value>`

initial: **normal**

Redefines `\datatoolcurrencysymbolprefixfmt`. Allows values are: `normal` (redefines to expand to its argument), `smallcaps` (redefines to expand to use `\textsc` with the argument converted to lowercase), or `smaller` (redefines to use `\textsmaller`, which will require the `relsize` package).

2.2.2. Parsing Locale-Formatted Numbers and Currency Values

Formatted numbers can be parsed provided the appropriate number group character and decimal character have been set with `\DTLsetnumberchars` and the currency symbol has been declared with `\DTLdefcurrency` (typically by loading a region file via `locales` or the document language support). If you want to format a plain number, you can use `\DTLdecimaltolocale` or `\DTLdecimaltocurrency`, described in §2.3, or use `siunitx`.

`\DTLconverttodecimal{<num>}{<cs>}`

Converts a formatted number `<num>` to a plain number and stores the result in `<cs>`. The `<num>` argument may be a command whose definition is a formatted number. A full expansion is not used on `<num>` to allow for non-robust currency symbols.



`\DTLconverttodecimal` is internally used by commands like `\DTLadd` to obtain the numerical value. The result is then converted back to a formatted number using either `\DTLdecimaltolocale` or `\DTLdecimaltocurrency`, depending on the data type of the supplied arguments. The result is a datum control sequence to reduce the need for re-parsing.

A warning is issued if the data type is a string rather than a numeric value and the value will be treated as zero. An empty $\langle num \rangle$ is also treated as zero. No trimming is performed on $\langle num \rangle$.

For example:



```
\DTLconverttodecimal{\$1,234.50}{\myNum}
```

This will define `\myName` to expand to `1234.50` (assuming the default number group character and decimal character). Again, the result is a datum control sequence to reduce the need for re-parsing.

2.2.3. Datum Commands

Instead of repeatedly parsing the same content, you may prefer to parse it once and store the information for later use. This can be done with the following command:



```
\DTLparse{<cs>}{<content>}
```

This parses $\langle content \rangle$ (without expansion) to determine its data type and (if numerical) its value.



```
\DTLxparse{<cs>}{<content>}
```

As `\DTLparse` but fully expands $\langle content \rangle$ before parsing.

In both cases, the parsed data is stored in the control sequence $\langle cs \rangle$ (a datum control sequence) in a form that includes the original value (or expanded value in the case of `\DTLxparse`), the data type, the numerical value (if one of the numerical types), and the currency symbol (if applicable).

The “string value”, which is the content that $\langle cs \rangle$ will expand to, may be automatically reformatted if an applicable setting is in effect (such as `numeric={auto-reformat}`).



This means that the numerical value is still available even if the number group character and decimal character are later changed. The important thing is to ensure that they are correct before parsing the data.

2. Base Commands (*datatool-base package*)

The datum item format is particularly useful with databases (see §3) that have numeric data which needs to be converted into plain numbers for arithmetic computations (such as aggregates) or plotting. If `store-datum` is enabled before creating the database, each value will be stored as a datum item. If you then assign a placeholder command to the value, for example with `\DTLmapgetvalues`, then that command will be a datum control sequence in the same format as that obtained with `\DTLparse`.

The component parts can then be extracted using the following expandable commands, where `<cs>` is the datum control sequence.

`\DTLusedatum{<cs>}`

Expands to the original value `<content>` that was parsed (or the expanded value in the case of `\DTLxparse`, or the reformatted string value, if the applicable option was in effect). You can also simply use the datum control sequence. The difference is that `\DTLusedatum` can fully expand the datum value whereas using the datum control sequence directly won't. If `<cs>` is `\dtlnovalue`, then `\DTLusedatum{<cs>}` will expand to `\dtlnovalue`.

`\DTLdatumvalue{<cs>}`

Expands to the numeric value (as a plain number) if the parsed value was numerical, otherwise expands to empty. If `<cs>` is `\dtlnovalue`, then `\DTLdatumvalue{<cs>}` will expand to `\DTLnumbernull`.

`\DTLdatumcurrency{<cs>}`

Expands to the currency symbol if the parsed value was a currency, otherwise expands to empty. If `<cs>` is `\dtlnovalue`, then `\DTLdatumcurrency{<cs>}` will expand to `\dtlnovalue`.

`\DTLdatumtype{<cs>}`

Expands to an integer representing the data type: 0 (string), 1 (integer), 2 (decimal), 3 (currency), 4 (timestamp), 5 (date), 6 (time) or -1 (unknown). If `<cs>` is `\dtlnovalue`, then `\DTLdatumtype{<cs>}` will expand to the unknown data type value.

For example:

```
\DTLparse\mydatum{1,234.0}  
Data type: \DTLdatumtype{\mydatum}.
```

Note that the data type is actually stored as a $\LaTeX$ 3 integer constant, but `\DTLdatumtype` will convert the constant value to an integer denotation. If you want the actual constant, use:

```
\exp_args:NV \datatool_datum_type:Nnnnn <cs>
```

but there's no check for `\dtlnovalue` in this case.

For debugging purposes, you may find it easier to have a textual representation of the data type so that you don't have to lookup what the numeric value represents. You can do this with:

```
\DTLgetDataTypeName{<number>}
```

This will expand to one of: `\DTLdatatypeunsetname`, `\DTLdatatypestringname`, `\DTLdatatypeintegername`, `\DTLdatatypedecimalname`, `\DTLdatatypecurrencyname`, `\DTLdatatypedatetimenam`, `\DTLdatatype-datetime`, `\DTLdatatypetimenam`, or `\DTLdatatypeinvalidname`.

You may also “show” the component parts in the console and transcript:

```
\datatool_datum_show:N{<cs>}
```

Instead of parsing an existing value, you can define a new datum control sequence using one of the commands below. Only `\DTLsetfpdatum` performs any parsing.

```
\DTLsetintegerdatum{<cs>}{<formatted value>}{<value>}
```

Defines the control sequence `<cs>` as an integer datum, where `<formatted value>` is the formatted integer and `<value>` is the integer value as a plain number.

```
\DTLxsetintegerdatum{<cs>}{<formatted value>}{<value>}
```

As `\DTLsetintegerdatum` but expands `<formatted value>` and `<value>`.

```
\DTLsetdecimaldatum{<cs>}{<formatted value>}{<value>}
```

Defines the control sequence `<cs>` as a decimal datum, where `<formatted value>` is the formatted decimal and `<value>` is the decimal value as a plain number.

```
\DTLxsetdecimaldatum{<cs>}{<formatted value>}{<value>}
```

As `\DTLsetdecimaldatum` but expands `<formatted value>` and `<value>`.



```
\DTLsetfpdatum{<cs>}{<formatted value>}{<value>}
```

Similar to `\DTLsetdecimaldatum` but this will expand and parse `<value>` and store it with the `\datatool_datum_fp:nnn` markup.



```
\DTLsetcurrencydatum{<cs>}{<formatted value>}{<value>}{<currency symbol>}
```

Defines the control sequence `<cs>` as a currency datum, where `<formatted value>` is the formatted currency and `<value>` is the currency value as a plain number. This has an extra argument which is the currency symbol.



```
\DTLxsetcurrencydatum{<cs>}{<formatted value>}{<value>}{<currency symbol>}
```

As `\DTLsetcurrencydatum` but expands `<formatted value>`, `<value>` and `<currency symbol>`.



```
\DTLsetstringdatum{<cs>}{<string>}
```

Defines the control sequence `<cs>` as a string datum.



```
\DTLxsetstringdatum{<cs>}{<string>}
```

As `\DTLsetstringdatum` but expands `<string>`.

Datum control sequences may be used in commands that expect a formatted number, such as `\DTLadd`, as demonstrated in Example 5, which is produced with the code below.



```
\usepackage{datatool-base}
\usepackage{siunitx}
\DTLparse{\numA}{23,452}
\DTLparse{\numB}{45.0}
\DTLparse{\numC}{\pounds 24.50}
\DTLsetfpdatum{\numD}{\num{1.5e-4}}{1.5e-4}
\begin{document}
Original value: \DTLusedatum{\numC} or \numC.
Numeric value: \DTLdatumvalue{\numC}.
Currency: \DTLdatumcurrency{\numC}.
Data type: \number\DTLdatumtype{\numC}.
```

 5

```

\DTLadd{\result}{\numA}{\numB}
$\numA + \numB = \result$

\DTLaddall{\result}{\numA,\numB,\numC}
$\numA + \numB + \numC = \result$

\dtladd{\result}{\DTLdatumvalue{\numA}}{\DTLdatum-
value{\numB}}
$\DTLdatumvalue{\numA} + \DTLdatumvalue{\numB}
= \result$

\dtladdall{\result}
{\DTLdatumvalue{\numA},\DTLdatumvalue{\numB},\DTL-
datumvalue{\numC}}
$\DTLdatumvalue{\numA} + \DTLdatumvalue{\numB}
+ \DTLdatumvalue{\numC} = \result$

\DTLxsetdecimaldatum{\total}{\num{\result}}{\result}
Total: \total.

\dtlml{\result}{20}{\DTLdatumvalue{\numD}}
$20 \times \numD = \result$

\end{document}

```

Example 5: Datum Control Sequences



Original value: £24.50 or £24.50. Numeric value: 24.50. Currency: £.

Data type: 3.

$$23,452 + 45.0 = 23,497$$

$$23,452 + 45.0 + £24.50 = £23,521.50$$

$$23452 + 45.0 = 23497$$

$$23452 + 45.0 + 24.50 = 23521.5$$

Total: 23 521.5.

$$20 \times 1.5 \times 10^{-4} = 0.003$$

2.2.4. Datum Items (Advanced)

If you have the expansion text from a datum control sequence (a datum item), that text will be in the form:


```
<marker-cs> { <string> } { <value> } { <currency> } { <type> }
```

Decimals may have the `<value>` part stored as:

```
\datatool_datum_fp:nnn { <fp-value> } { <fp-var-content> } { <decimal> }
```

With `math=fp` this expands to `<decimal>` (since the `fp` package can't parse scientific notation) otherwise this expands to `<fp-value>` (the original value if supplied in scientific notation or the plain number obtained from parsing a locale decimal). The `<fp-var-content>` argument allows an `l3fp` variable to be reconstructed (with `\datatool_set_fp:Nn`) without having to reparse the value.

Temporal data types may have the `<value>` part stored as:

```
\DTLtemporalvalue { <number> } { <ISO> }
```

This allows the date/time stamp to be retained. This simply expands to the first argument by default, which is the numeric value associated with the data. In the case of date (without time), the value is an integer Julian day; in the case of a timestamp (date and time), the value is a decimal Julian date; in the case of time (without a date), the value is the fractional part of the Julian date.

The date/time stamp can be extracted with:

```
\datatool_extract_timestamp:NN <datum-cs> <result-tl>
```

where `<result-tl-var>` is the token list variable in which to store the date/time stamp and `<datum-cs>` is the datum control sequence. This works by locally redefining `\DTLtemporalvalue` and then expanding `\DTLtemporalvalue { <value> }`. If the `<value>` part in `<datum-cs>` is just a number and not encapsulated within `\DTLtemporalvalue` then this trick won't work and the number will need to be converted back. The result will be empty if there is no date/time information.

To allow for new data types introduced in a later version, you can check for the current maximum allowed value with:

```
\datatool_max_known_type:
```

This will expand to the appropriate constant.

```
\datatool_if_valid_datum_type:nTF { <n> } { <true> } { <false> }
\datatool_if_valid_datum_type_p:n { <n> }
```

Tests if the argument `<n>` represents a valid data type (including unknown).



```
\datatool_if_numeric_datum_type:nTF {<n>} {<true>}
{<false>}
\datatool_if_numeric_datum_type_p:n {<n>}
```

Tests if the argument $\langle n \rangle$ represents a numeric data type. Note that temporal data types are considered numeric.



```
\datatool_if_temporal_datum_type:nTF {<n>} {<true>}
{<false>}
\datatool_if_temporal_datum_type_p:n {<n>}
```

Tests if the argument $\langle n \rangle$ represents a temporal data type.



```
\datatool_if_number_only_datum_type:nTF {<n>} {<true>}
{<false>}
\datatool_if_number_only_datum_type_p:n {<n>}
```

Tests if the argument $\langle n \rangle$ represents an integer or decimal data type (not currency or temporal).



```
\datatool_if_any_int_datum_type:nTF {<n>} {<true>}
{<false>}
\datatool_if_any_int_datum_type_p:n {<n>}
```

Tests if the argument $\langle n \rangle$ represents data type that has an integer value (integer or date, but not decimal or currency or timestamps or times).

2.2.4.1. Datum Components

It's possible to pick out the desired component using an $\langle n \rangle$ of $\langle m \rangle$ style of command. However, the following commands are provided as it's more obvious from the command name which element is required. Note that these require L^AT_EX3 syntax enabled:



```
\datatool_datum_string:Nnnnn <marker-cs> {<string>} {<value>}
{<currency>} {<type>}
```

Expands to $\langle string \rangle$.



```
\datatool_datum_value:Nnnnn <marker-cs> {<string>} {<value>}
{<currency>} {<type>}
```

Expands to $\langle value \rangle$.

```
\datatool_datum_currency:Nnnnn  $\langle marker-cs \rangle$  { $\langle string \rangle$ } { $\langle value \rangle$ }
{ $\langle currency \rangle$ } { $\langle type \rangle$ }
```

Expands to $\langle currency \rangle$.

```
\datatool_datum_type:Nnnnn  $\langle marker-cs \rangle$  { $\langle string \rangle$ } { $\langle value \rangle$ }
{ $\langle currency \rangle$ } { $\langle type \rangle$ }
```

Expands to $\langle type \rangle$.

2.2.4.2. Datum Tests for Equality

If you want to test if a datum control sequence is equal to a string, then you can't simply use $\backslash tl\_if\_eq:NnTF$ or $\backslash ifdefstring$ as the datum markup will prevent a match. Commands such as $\backslash DTLifstringeq$ expand the arguments which will remove the datum markup, but the following commands take the data type into account.

If both arguments have a numeric type then they will be compared numerically and by the currency symbol. If both are ordinary token lists without the datum markup then they will be compared using a normal token list comparison. If one has the datum format and the other doesn't, then a string comparison is used.

```
\datatool_if_value_eq:NN $\underline{TF}$   $\langle tl var1 \rangle$   $\langle tl var2 \rangle$  { $\langle true \rangle$ } { $\langle false \rangle$ }
```

Compares two variables where one or other may be a datum control sequence or simply a token list variable.

```
\datatool_if_value_eq:Nn $\underline{TF}$   $\langle tl var \rangle$  { $\langle tl \rangle$ } { $\langle true \rangle$ } { $\langle false \rangle$ }
```

Test for equality where the variable $\langle tl var \rangle$ may be a datum control sequence and the token list $\langle tl \rangle$ may be a datum item.

```
\datatool_if_value_eq:nN $\underline{TF}$  { $\langle tl \rangle$ }  $\langle tl var \rangle$  { $\langle true \rangle$ } { $\langle false \rangle$ }
```

Test for equality where the token list $\langle tl \rangle$ may be a datum item and the variable $\langle tl var \rangle$ may be a datum control sequence.

```
\datatool_if_value_eq:nn $\underline{TF}$  { $\langle tl1 \rangle$ } { $\langle tl2 \rangle$ } { $\langle true \rangle$ } { $\langle false \rangle$ }
```

Compares two token lists where one or other may be a datum item.

Example 6 demonstrates the above commands. First some datum control sequences are defined:

6

2. Base Commands (*datatool-base package*)

```
\DTLparse{\Fruit}{Pear}  
\DTLparse{\Price}{\$1.50}  
\DTLparse{\Quantity}{10}
```

The following `\OtherPrice` is numerically equivalent to `\Price` and has the same currency symbol but the string representation is different:

```
\DTLsetcurrencydatum{\OtherPrice}  
{1 dollar 50\textcent}{1.5}{\$}
```

Similarly, the following `\OtherQuantity` has the same numerical value as `\Quantity` but it's a decimal instead of an integer:

```
\DTLsetdecimaldatum{\OtherQuantity}{10.00}{10.0}
```

For convenience a command is provided for the tests:

```
\newcommand{\test}[3]{#1=#2 (\texttt{\string#3}) ?  
#3{#1}{#2}{true}{false}.\par}
```

The actual tests need to have $\text{\LaTeX}3$ syntax enabled:

```
\ExplSyntaxOn
```

First are the string tests:

```
\test \Fruit {Pear} \tl_if_eq:NnTF  
\test \Fruit {Pear} \tl_if_eq:enTF  
\test \Fruit {Pear} \datatool_if_value_eq:NnTF
```

The next set may appear to be numeric tests but they are still string tests because they are being compared with a non-datum token list.

```
\test \Price {\$1.50} \tl_if_eq:NnTF  
\test \Price {\$1.50} \tl_if_eq:enTF  
\test \Price {\$1.50} \datatool_if_value_eq:NnTF
```

2. Base Commands (*datatool-base* package)

```
\test \Price {\$1.5} \datatool_if_value_eq:NnTF
```

For an actual numeric test, both arguments must use the datum format. Note that `\Price` and `\OtherPrice` are numerically equivalent but when viewed as token list variables, they don't have the same content.

```
\test \Price \OtherPrice \tl_if_eq:NNTF
\test \Price \OtherPrice \datatool_if_value_eq:NNTF
```

There are similar tests for the quantity:

```
\test \Quantity {10} \tl_if_eq:NnTF
\test \Quantity {10} \tl_if_eq:enTF
\test \Quantity {10} \datatool_if_value_eq:NnTF
\test \Quantity {10.00} \datatool_if_value_eq:NnTF
\test \Quantity \OtherQuantity \tl_if_eq:NNTF
\test \Quantity \OtherQuantity \datatool_if_value_eq:NNTF
```

↑ Example 6: Datum Tests for Equality

```
Pear=Pear (\tl_if_eq:NnTF) ? false.
Pear=Pear (\tl_if_eq:enTF) ? true.
Pear=Pear (\datatool_if_value_eq:NnTF) ? true.
$1.50=$1.50 (\tl_if_eq:NnTF) ? false.
$1.50=$1.50 (\tl_if_eq:enTF) ? true.
$1.50=$1.50 (\datatool_if_value_eq:NnTF) ? true.
$1.50=$1.5 (\datatool_if_value_eq:NnTF) ? false.
$1.50=1 dollar 50¢ (\tl_if_eq:NNTF) ? false.
$1.50=1 dollar 50¢ (\datatool_if_value_eq:NNTF) ? true.
10=10 (\tl_if_eq:NnTF) ? false.
10=10 (\tl_if_eq:enTF) ? true.
10=10 (\datatool_if_value_eq:NnTF) ? true.
10=10.00 (\datatool_if_value_eq:NnTF) ? false.
10=10.00 (\tl_if_eq:NNTF) ? false.
10=10.00 (\datatool_if_value_eq:NNTF) ? true.
```

2.2.4.3. Conversion to Floating Point

If you need to set an `l3fp` variable to a value that may be a datum control sequence or datum item or may not yet be parsed, you can use:

`\datatool_set_fp:Nn <fp-var> {<value>}`

This sets the floating point variable `<fp-var>` to the floating point number obtained from the given `<value>`. If the `<value>` is either a datum control sequence or datum item then no parsing is required. If not, the `<value>` will be expanded and then parsed to obtain its numeric value before setting the variable. (Be aware that this may cause non-robust currency symbols to expand so that they are no longer recognised as a currency symbol.) If `<value>` is determined to have a string or unknown data type the variable will be set to zero.

Example 7 performs floating point calculations on a formatted number (which needs to be parsed according to the current settings) and a value provided in scientific notation (with a formatted representation using `siunitx`).

```
\usepackage{datatool-base}
\usepackage{siunitx}
\DTLparse{\numA}{1,500.0}
\DTLsetfpdatum{\numB}{\num{1.5e-4}}{1.5e-4}
\begin{document}
A = \numA \space (value: \DTLdatumvalue\numA) .
B = \numB \space (value: \DTLdatumvalue\numB) .

\ExplSyntaxOn
\datatool_set_fp:Nn \l_tmpa_fp { \numA }
\datatool_set_fp:Nn \l_tmpb_fp { \numB }
\fp_to_tl:N \l_tmpa_fp \c_space_tl
\texttimes \c_space_tl
\fp_to_tl:N \l_tmpb_fp \c_space_tl = \c_space_tl
\fp_eval:n { \l_tmpa_fp * \l_tmpb_fp }
\ExplSyntaxOff
\end{document}
```

↑ Example 7: Datum Control Sequences to Floating Point Variables

$A = 1,500.0$ (value: 1500.0). $B = 1.5 \times 10^{-4}$ (value: 1.5e-4).
 $1500 \times 1.5e-4 = 0.225$

2.3. Localisation

The *datatool-base* package (v3.0+) loads the *tracklang* package, which attempts to determine the document localisation settings. No actual localisation is provided by *tracklang*, but it enables support to be easily added and maintained independently from a package (that uses the *tracklang* interface) with *ldf* files that have a particular naming scheme.

This means that by adding a file called `datatool-⟨locale⟩.ldf` to $\text{\TeX}$'s path, the file can automatically be loaded by *datatool-base* without any adjustments to the *datatool-base* code. There is a search order for `⟨locale⟩` to allow for fine grained support. See the *tracklang* documentation for further details or the “Locale Sensitive Files” section of *Using tracklang in Packages with Localisation Features*.¹

The *tracklang* package has limitations, but you may be able to supply the language identifier as a document class option, for example:

```
\documentclass[british]{article}
```

or load *babel*/*polyglossia* and setup language support before the first package to load *tracklang*, for example:

```
\usepackage[british]{babel}
\usepackage{datatool-base}
```

or use *datatool-base*'s `locales` (or `lang`) option, for example:

```
\usepackage[locales=en-GB]{datatool-base}
```

If you use `\babelprovide`, ensure that you have at least version 1.6.4 of *tracklang* and load *tracklang* after all instances of `\babelprovide`. There's no support for “lazy loading” in the document environment.

Note that this option will have an effect on packages that are subsequently loaded that also use *tracklang*. Likewise, if you have already loaded a package that uses *tracklang* (such as *datetime2*) then the tracked locales from that will be picked up. For example:

```
\usepackage[en-GB]{datetime2}
\usepackage{datatool-base}
```

See the *tracklang* documentation or *Localisation with tracklang.tex*² for further details.

¹dickimaw-books.com/latex/tracklang/otherpkg.shtml

²dickimaw-books.com/latex/tracklang



If tracklang doesn't recognise the language identifier, the root language will be “undetermined” (with code “und”) and so the file `datatool-undetermined.ldf` (provided with `datatool`) will be loaded.

For some packages (such as `databib` and `person`), the localisation support just relates to translating fixed text and the corresponding filename may simply have *<locale>* as the tracklang root language label. So regardless of whether you have used `locales=en-GB` or `locales=en-US`, the `person` package will require the file `person-english.ldf` (provided with `datatool-english`).

However, settings such as the currency symbol are specific to a region not a language. So `locales=en-GB` would need the default currency switched to GBP whereas `locales=en-IE` would need the default currency switched to EUR and `locales=en-ZA` would need the default currency switched to ZAR.

Therefore, localisation support for `datatool-base` (and its supplementary packages) is split into two parts: the language file `datatool-<language>.ldf` (for example, `datatool-english.ldf`) which deals with the orthography, translations of fixed text, and other language-specific code, and the region file `datatool-<region>.ldf` (for example, `datatool-GB.ldf`) which deals with language-independent region code. You will need both files for full support but partial support can be obtained if one is missing.

The region files are fairly straightforward (albeit time-consuming) to create. They are therefore all bundled together in a single distribution `datatool-regions` which needs to be installed in addition to installing `datatool`. See §2.3.4 for further details.

Locale-sensitive commands that relate to regions may all be reset back to their original definitions with:



```
\DTLresetRegion
```

Note that this will clear `\l_datatool_current_region_tl` and reset the current number group character and decimal character and currency in addition to redefining commands such as `\DTLCurrentLocaleCurrencyDP`.

The language files are more complicated and require knowledge of someone familiar with the language. Each language bundle should therefore be developed independently by a maintainer fluent in the language and it will need to be installed in addition to installing `datatool`. At the time of writing, only `datatool-english` is available, but you can copy and adapt it as appropriate. (Don't add me as author or maintainer of your contribution.) The `datatool-english` bundle includes limited support for Old English (Anglo-Saxon) for Latin and Runic scripts, which may be used as examples for extended Latin or non-Latin languages. See §2.3.5 for further details.

Locale-sensitive commands that relate to language may all be reset back to their original definitions with:



```
\DTLresetLanguage
```


2. Base Commands (*datatool-base* package)

Note that this clears `\l_datatool_current_language_tl` in addition to redefining commands such as `\DTLlandname`, but only for the *datatool-base* set of commands. Additional commands provided for the supplementary packages are not affected.

Example 8 assumes that *datatool-regions* and *datatool-english* are both installed.

8

```
\usepackage[locales=en-CA]{datatool-base}
\begin{document}
Default currency: \DTLCurrencyCode.

\newcommand{\mylist}{elk,élite,elephant}
\DTLsortwordlist{\mylist}{\DTLsortletterhandler}
Sorted list: \DTLformatlist{\mylist}.
\end{document}
```

↑ Example 8: Localisation Support (en-CA)



Default currency: CAD.
Sorted list: elephant, élite and elk.

The above example shows the default currency code “CAD”, which has been set by *datatool-CA.ldf*. The sorted list has “élite” between “elephant” and “elk” because *datatool-english.ldf* has enabled support for common UTF-8 characters so that “é” is treated as “e” for sorting purposes.

Suppose now that you have *datatool-regions* installed but no French support. However your document language is French Canadian (fr-CA):

9

```
\usepackage{babel}
\babelprovide{canadianfrench}
\usepackage{datatool-base}
\begin{document}
Default currency: \DTLCurrencyCode.

\newcommand{\mylist}{elk,élite,elephant}
\DTLsortwordlist{\mylist}{\DTLsortletterhandler}
Sorted list: \DTLformatlist{\mylist}.
\end{document}
```

In this case, the *datatool-CA.ldf* file is found, so the default currency code is still CAD but no file is found to provide support for the sorting handler so the extended Latin character “é” is placed after the Basic Latin characters.

↑ Example 9: Localisation Support (fr-CA)

Default currency: CAD.
Sorted list: elephant, elk & élite.

Localisation files may provide options. These are defined with: `\datatoollocaledefinekeys:nn`. This is simply a shortcut that uses `\keys_define:nn`. The `<module>` should be the applicable language code (for example, “en”) or region code (for example, “GB”) or tag (for example, “en-CA” or “fr-CA”), depending on what kind of support the file provides. Sub-modules may also be specified.

These options can be set in the document with:

```
\DTLsetLocaleOptions[<parent module(s)>]{<module(s)>}{<key=value
list>}{<modifier>}
```

If the optional argument is provided, this iterates over each locale parent module and sets the given options for each sub-module identified by `<parent>/<module>`. If the optional argument is omitted or empty, this iterates over each locale module and sets the given options.

For example, with `datatool-GB.ldf` the parent module is “GB” and there are no sub-modules. To switch number style:

```
\DTLsetLocaleOptions{GB}{number-style=education}
```

Another example, both `datatool-GB.ldf` and `datatool-CA.ldf` support a currency symbol prefix so the setting can be switched on for both at the same time:

```
\DTLsetLocaleOptions{CA,GB}{currency-symbol-prefix}
```

The `databib-english.ldf` has parent module “en” and sub-module “databib”. To switch the way month names are abbreviated for the `abbrv` style:

```
\DTLsetLocaleOptions[en]{databib}{short-month-
style=dotless}
```

Or:

```
\DTLsetLocaleOptions{en/databib}{short-month-  
style=dotless}
```

The unstarred form uses: `\datatoolsetlocaleoptions:nn` This iterates over each module and sets the provided options using `\keys_set:nn`, which will trigger an error for unknown options.

The starred form uses: `\datatoolsetlocaleoptions:nn` This iterates over each module and sets the provided options using `\keys_set_known:nn`, which won't trigger an error for unknown options.

If you want to directly use the `l3keys` functions, the module path should be prefixed with “`datatool / locale /`”.

2.3.1. Encoding

In recent years, the $\text{\LaTeX}$ kernel has provided significant improvements to UTF-8 support for pdf $\text{\LaTeX}$. (The newer engines, Xe $\text{\LaTeX}$ and Lua $\text{\LaTeX}$ are natively UTF-8.) In particular, even if you don't load `inputenc`, the document is now assumed to be UTF-8 (whereas in the past the default encoding was ASCII).

If `inputenc` is required, it should be loaded before `datatool-base` (and `tracklang`). Non-UTF-8 documents may not be supported by the localisation files. For example, the `\string` argument of `\DTLdefcurrency` may not be correct.

To assist localisation files, the `datatool-base` package provides both a string (detokenized) variable and corresponding token list variables that expand to common symbols (mostly currency) that are included in Unicode and may be of use with localisation. These variables are first defined to expand to an approximate ASCII representation, but then will be redefined if the relevant `datatool-⟨encoding⟩.ldf` file is found. This means that unsupported encodings will fallback on ASCII values. There is limited support for ISO-8859-1 (cent, pound, currency and yen).

For example, `datatool-GB.ldf` defines the GBP currency as follows:

```
\datatool_def_currency:nnnV  
{ \datatoolGBcurrencyfmt }  
{ GBP }  
{ \pounds }  
\l_datatool_pound_tl
```

This means that the region `ldf` file doesn't need to keep track of the encoding. (The language `ldf` typically does.)

`\l_datatool_cent_str`

Expands to the string representation of the cent sign “ ¢ ”, if supported by the current encoding, or “ c ” otherwise..

`\l_datatool_cent_tl`

Expands to the symbol representation of the cent sign “ ¢ ”, if supported by the current encoding, or “ c ” otherwise..

`\l_datatool_pound_str`

Expands to the string representation of the pound sign “ £ ”, if supported by the current encoding, or “ L ” otherwise..

`\l_datatool_pound_tl`

Expands to the symbol representation of the pound sign “ £ ”, if supported by the current encoding, or “ L ” otherwise..

`\l_datatool_currency_str`

Expands to the string representation of the currency sign “ \$ ”, if supported by the current encoding, or “ # ” otherwise..

`\l_datatool_currency_tl`

Expands to the symbol representation of the currency sign “ \$ ”, if supported by the current encoding, or “ # ” otherwise..

`\l_datatool_yen_str`

Expands to the string representation of the yen sign “ ¥ ”, if supported by the current encoding, or “ Y ” otherwise..

`\l_datatool_yen_tl`

Expands to the symbol representation of the yen sign “ ¥ ”, if supported by the current encoding, or “ Y ” otherwise..

2. Base Commands (*datatool-base* package)

`\l_datatool_middot_str`

Expands to the string representation of the middle dot (raised decimal point) “ ”, if supported by the current encoding, or “ . ” otherwise..

`\l_datatool_middot_tl`

Expands to the symbol representation of the middle dot (raised decimal point) “ ”, if supported by the current encoding, or “ . ” otherwise..

`\l_datatool_florin_str`

Expands to the string representation of the florin sign “ ”, if supported by the current encoding, or “ f ” otherwise..

`\l_datatool_florin_tl`

Expands to the symbol representation of the florin sign “ ”, if supported by the current encoding, or “ f ” otherwise..

`\l_datatool_baht_str`

Expands to the string representation of the baht sign “ ”, if supported by the current encoding, or “ B ” otherwise..

`\l_datatool_baht_tl`

Expands to the symbol representation of the baht sign “ ”, if supported by the current encoding, or “ B ” otherwise..

`\l_datatool_ecu_str`

Expands to the string representation of the ecu sign “ ”, if supported by the current encoding, or “ CE ” otherwise..

`\l_datatool_ecu_tl`

Expands to the symbol representation of the ecu sign “ ”, if supported by the current encoding, or “ CE ” otherwise..

`\l_datatool_colonsign_str`

Expands to the string representation of the colon sign “ ”, if supported by the current encoding, or “C” otherwise..

`\l_datatool_colonsign_tl`

Expands to the symbol representation of the colon sign “ ”, if supported by the current encoding, or “C” otherwise..

`\l_datatool_cruzerio_str`

Expands to the string representation of the cruzerio sign “ ”, if supported by the current encoding, or “Cr” otherwise..

`\l_datatool_cruzerio_tl`

Expands to the symbol representation of the cruzerio sign “ ”, if supported by the current encoding, or “Cr” otherwise..

`\l_datatool_frenchfranc_str`

Expands to the string representation of the French franc sign “ ”, if supported by the current encoding, or “F” otherwise..

`\l_datatool_frenchfranc_tl`

Expands to the symbol representation of the French franc sign “ ”, if supported by the current encoding, or “F” otherwise..

`\l_datatool_lira_str`

Expands to the string representation of the lira sign “ ”, if supported by the current encoding, or “L” otherwise..

`\l_datatool_lira_tl`

Expands to the symbol representation of the lira sign “ ”, if supported by the current encoding, or “L” otherwise..

`\l_datatool_mill_str`



Expands to the string representation of the mill sign “ ¢ ”, if supported by the current encoding, or “m” otherwise..

`\l_datatool_mill_tl`



Expands to the symbol representation of the mill sign “ ¢ ”, if supported by the current encoding, or “m” otherwise..

`\l_datatool_naira_str`



Expands to the string representation of the naira sign “ ₦ ”, if supported by the current encoding, or “N” otherwise..

`\l_datatool_naira_tl`



Expands to the symbol representation of the naira sign “ ₦ ”, if supported by the current encoding, or “N” otherwise..

`\l_datatool_peseta_str`



Expands to the string representation of the peseta sign “ ¢ ”, if supported by the current encoding, or “Pts” otherwise..

`\l_datatool_peseta_tl`



Expands to the symbol representation of the peseta sign “ ¢ ”, if supported by the current encoding, or “Pts” otherwise..

`\l_datatool_rupee_str`



Expands to the string representation of the rupee sign “ ₹ ”, if supported by the current encoding, or “Rs” otherwise..

`\l_datatool_rupee_tl`



Expands to the symbol representation of the rupee sign “ ₹ ”, if supported by the current encoding, or “Rs” otherwise..

`\l_datatool_won_str`

Expands to the string representation of the won sign “ ₩ ”, if supported by the current encoding, or “W” otherwise..

`\l_datatool_won_tl`

Expands to the symbol representation of the won sign “ ₩ ”, if supported by the current encoding, or “W” otherwise..

`\l_datatool_shekel_str`

Expands to the string representation of the shekel sign “ ₪ ”, if supported by the current encoding, or “S” otherwise..

`\l_datatool_shekel_tl`

Expands to the symbol representation of the shekel sign “ ₪ ”, if supported by the current encoding, or “S” otherwise..

`\l_datatool_dong_str`

Expands to the string representation of the dong sign “ ₩ ”, if supported by the current encoding, or “d” otherwise..

`\l_datatool_dong_tl`

Expands to the symbol representation of the dong sign “ ₩ ”, if supported by the current encoding, or “d” otherwise..

`\l_datatool_euro_str`

Expands to the string representation of the euro sign “ € ”, if supported by the current encoding, or “E” otherwise..

`\l_datatool_euro_tl`

Expands to the symbol representation of the euro sign “ € ”, if supported by the current encoding, or “E” otherwise..

`\l_datatool_kip_str`

Expands to the string representation of the kip sign “ ”, if supported by the current encoding, or “K” otherwise..

`\l_datatool_kip_tl`

Expands to the symbol representation of the kip sign “ ”, if supported by the current encoding, or “K” otherwise..

`\l_datatool_tugrik_str`

Expands to the string representation of the tugrik sign “ ”, if supported by the current encoding, or “T” otherwise..

`\l_datatool_tugrik_tl`

Expands to the symbol representation of the tugrik sign “ ”, if supported by the current encoding, or “T” otherwise..

`\l_datatool_drachma_str`

Expands to the string representation of the drachma sign “ ”, if supported by the current encoding, or “Dr” otherwise..

`\l_datatool_drachma_tl`

Expands to the symbol representation of the drachma sign “ ”, if supported by the current encoding, or “Dr” otherwise..

`\l_datatool_germanpenny_str`

Expands to the string representation of the Germany penny sign “ ”, if supported by the current encoding, or “p” otherwise..

`\l_datatool_germanpenny_tl`

Expands to the symbol representation of the Germany penny sign “ ”, if supported by the current encoding, or “p” otherwise..

`\l_datatool_peso_str`

Expands to the string representation of the peso sign “ \$ ”, if supported by the current encoding, or “ P ” otherwise..

`\l_datatool_peso_tl`

Expands to the symbol representation of the peso sign “ \$ ”, if supported by the current encoding, or “ P ” otherwise..

`\l_datatool_guarani_str`

Expands to the string representation of the guarani sign “ ₧ ”, if supported by the current encoding, or “ G . ” otherwise..

`\l_datatool_guarani_tl`

Expands to the symbol representation of the guarani sign “ ₧ ”, if supported by the current encoding, or “ G . ” otherwise..

`\l_datatool_austral_str`

Expands to the string representation of the austral sign “ \$ ”, if supported by the current encoding, or “ A ” otherwise..

`\l_datatool_austral_tl`

Expands to the symbol representation of the austral sign “ \$ ”, if supported by the current encoding, or “ A ” otherwise..

`\l_datatool_hryvnia_str`

Expands to the string representation of the hryvnia sign “ ₴ ”, if supported by the current encoding, or “ S ” otherwise..

`\l_datatool_hryvnia_tl`

Expands to the symbol representation of the hryvnia sign “ ₴ ”, if supported by the current encoding, or “ S ” otherwise..

`\l_datatool_cedi_str`

Expands to the string representation of the cedi sign “ ”, if supported by the current encoding, or “S” otherwise..

`\l_datatool_cedi_tl`

Expands to the symbol representation of the cedi sign “ ”, if supported by the current encoding, or “S” otherwise..

`\l_datatool_livretournois_str`

Expands to the string representation of the livre tournois sign “ ”, if supported by the current encoding, or “lt” otherwise..

`\l_datatool_livretournois_tl`

Expands to the symbol representation of the livre tournois sign “ ”, if supported by the current encoding, or “lt” otherwise..

`\l_datatool_spesmilo_str`

Expands to the string representation of the spesmilo sign “ ”, if supported by the current encoding, or “Sm” otherwise..

`\l_datatool_spesmilo_tl`

Expands to the symbol representation of the spesmilo sign “ ”, if supported by the current encoding, or “Sm” otherwise..

`\l_datatool_tenge_str`

Expands to the string representation of the tenge sign “ ”, if supported by the current encoding, or “T” otherwise..

`\l_datatool_tenge_tl`

Expands to the symbol representation of the tenge sign “ ”, if supported by the current encoding, or “T” otherwise..

`\l_datatool_indianrupee_str`

Expands to the string representation of the Indian rupee sign “₹”, if supported by the current encoding, or “R” otherwise..

`\l_datatool_indianrupee_tl`

Expands to the symbol representation of the Indian rupee sign “₹”, if supported by the current encoding, or “R” otherwise..

`\l_datatool_turkishlira_str`

Expands to the string representation of the Turkish lira sign “₺”, if supported by the current encoding, or “L” otherwise..

`\l_datatool_turkishlira_tl`

Expands to the symbol representation of the Turkish lira sign “₺”, if supported by the current encoding, or “L” otherwise..

`\l_datatool_nordicmark_str`

Expands to the string representation of the Nordic mark sign “₰”, if supported by the current encoding, or “M” otherwise..

`\l_datatool_nordicmark_tl`

Expands to the symbol representation of the Nordic mark sign “₰”, if supported by the current encoding, or “M” otherwise..

`\l_datatool_manat_str`

Expands to the string representation of the manat sign “₼”, if supported by the current encoding, or “M” otherwise..

`\l_datatool_manat_tl`

Expands to the symbol representation of the manat sign “₼”, if supported by the current encoding, or “M” otherwise..

`\l_datatool_ruble_str`



Expands to the string representation of the ruble sign “₽”, if supported by the current encoding, or “R” otherwise..

`\l_datatool_ruble_tl`



Expands to the symbol representation of the ruble sign “₽”, if supported by the current encoding, or “R” otherwise..

`\l_datatool_lari_str`



Expands to the string representation of the lari sign “₾”, if supported by the current encoding, or “L” otherwise..

`\l_datatool_lari_tl`



Expands to the symbol representation of the lari sign “₾”, if supported by the current encoding, or “L” otherwise..

`\l_datatool_bitcoin_str`



Expands to the string representation of the bitcoin sign “₿”, if supported by the current encoding, or “L” otherwise..

`\l_datatool_bitcoin_tl`



Expands to the symbol representation of the bitcoin sign “₿”, if supported by the current encoding, or “L” otherwise..

`\l_datatool_som_str`



Expands to the string representation of the som sign “₮”, if supported by the current encoding, or “c” otherwise..

`\l_datatool_som_tl`



Expands to the symbol representation of the som sign “₮”, if supported by the current encoding, or “c” otherwise..

If any of the currency symbols are available in the current encoding, they will be added to the currency signs regular expression variable:



```
\l_datatool_currencysigns_regex
```

This may be used within the locale handler to match for supported currency symbols.

2.3.2. Numerical

Non locale-sensitive numeric commands (such as `\dtladd`) require plain numbers with a period/full stop decimal point (.) and no number group character or currency symbol.

Numeric commands for formatted numbers (such as `\DTLadd`) parse their values for the currency symbol, decimal character and number group character. The number group character is only used in integers and before the decimal character in decimal and currency values. The decimal character is only relevant to decimal numbers and currency values.



```
\DTLsetnumberchars{<number group char>}{<decimal char>}
```

Sets the current number group character and decimal character. The default values are “,” (comma) and “decimal point” (full stop/period), although localisation support may change this.

With $\LaTeX$ 3 syntax enabled, the following may be used instead.



```
\datatool_set_numberchars:nn{<number group char>}{<decimal char>}  
variants: nV Vn VV
```

As from version 3.0, `\DTLsetnumberchars` simply uses this function to set the current number group character and decimal character.



```
\datatool_set_numberchars:nnnn{<format number group  
char>}{<format decimal char>}{<parse number group char>}{<parse decimal char>}  
variants: VVVV eeee
```

Allows alternative content to be used when formatting, but be aware that repeated parsing and formatting will fail if the parsing and formatting characters are different.

For more complex parsing requirements, regular expressions can be provided to match the number group character and decimal character sub-groups:



```
\datatool_set_numberchars_regex:nnnn{<format number group  
char>}{<format decimal char>}{<parse number group regex>}{<parse decimal regex>}  
variants: VVnn Vnnn nVnn
```

The final two arguments should be in a regular expression form. These will be embedded into the main parsing regular expression with `\ur`.

2. Base Commands (*datatool-base package*)

```
\datatool_set_numberchars_regex_tl:nnnn{<format number  
group char>}{<format decimal char>}{<parse number group regex>}{<parse decimal  
char>}
```

variants: VVnn Vnnn nVnn nVnV nnnV

The third argument is a regular expression to match the number group character but the fourth is just the decimal character.

```
\datatool_set_numberchars_tl_regex:nnnn{<format number  
group char>}{<format decimal char>}{<parse number group char>}{<parse decimal  
regex>}
```

variants: VVnn Vnnn nVnn VnVn nnVn

The third is just the decimal character but the fourth argument is a regular expression to match the decimal character.

The following are just shortcuts that use one of the above.

```
\datatool_set_thinspace_group_decimal_char:n{<decimal  
char>}
```

variant: V

A special case for thin space number group separators. This command is similar to `\datatool_set_numberchars:nn` but uses `\,` (thin space) for the number group character when formatting, and allows `\,` or a normal space or the Unicode character U+2009 (thin space) as the number group character when parsing. The decimal character for both formatting and parsing is set to `<decimal char>`.

```
\datatool_set_underscore_group_decimal_char:n{<decimal  
char>}
```

variant: V

Similarly, but uses `\_` for the number group character when formatting but accepts both `\_` or the underscore character when parsing.

```
\datatool_set_apos_group_decimal_char:n{<decimal char>}
```

variant: V

Similarly, but uses an apostrophe (') for the number group character when formatting but will match on:

```
\c_datatool_apostrophe_regex
```

when parsing. This matches either the straight apostrophe (U+27) or the curly apostrophe (U+2019).

```
\DTLsetdefaultcurrency{<ISO or symbol>}
```

Sets the default currency. If the argument is an ISO code, then the currency must have first been defined with `\DTLdefcurrency` (see §2.6). This command also defines `\DTLCurrencyCode` to expand to the associated ISO code and redefines `\DTLfmtcurrency` to match the formatting associated with the currency.

To allow for backward-compatibility, if the argument hasn't been identified with `\DTLdefcurrency` then it's assumed to be just a currency symbol and `\DTLCurrencyCode` will be defined to "XXX". `\DTLfmtcurrency` won't be changed. This form is now discouraged and may be deprecated in future.

The region file should register the currency code with:

```
\datatool_register_regional_currency_code:nn  
{<region-code>} {<currency-code>}
```

This makes it easier for the currency parser to check for currency symbols that are prefixed by the region code (for example, US\$ or GB£). Note that this check is only performed if the region file defines:

```
\datatool<Region>symbolprefix{<tag>}
```

The prefix command allows the region code to be shown before the currency symbol, if applicable. It may be used in the definition of the currency formatting command.

The naming of the `\datatool<Region>symbolprefix` command is important as the parser used by commands like `\DTLparse` will check for it and, if defined, will also check for currency symbols prefixed by their region's code.

The prefix command may either expand to nothing or to:

```
\datatool_currency_symbol_region_prefix:n{<tag>}
```

This uses `\DTLcurrCodeOrSymOrChar` to only show the tag when that command expands to its second or third argument. (Since the tag is typically the region code, it's redundant to insert it before the currency code.) The tag is formatted with:


```
\datatoolcurrencysymbolprefixfmt{<tag>}
```

This may be redefined, which will change the way the tag is formatted for all regions that support it. For convenience, the `numeric` option `region-currency-prefix` may be used to redefine this formatting command to use small caps.

Region files should provide a hook called

```
\datatool<Region>SetCurrency
```

where `<Region>` is the two letter uppercase region code. This command should check the boolean variable:

```
\l_datatool_region_set_currency_bool
```

(which corresponds to the `region-currency` numeric option). The hook should only set the currency if this boolean value is true.

Similarly, a hook to set the current number group character and decimal character:

```
\datatool<Region>SetNumberChars
```

This command should check the boolean variable:

```
\l_datatool_region_set_numberchars_bool
```

(which corresponds to the `region-number-chars` numeric option). The hook should only set the number group and decimal characters if this boolean value is true.

If you simply want to typeset plain numbers as formatted numbers then consider using `siunitx` instead. However you can use the following, which picks up the above settings.

```
\DTLdecimaltolocale{<num>}{<cs>}
```

Converts a plain number `<num>` into a formatted number and stores the result in `<cs>`. If a currency symbol is required, use `\DTLdecimaltocurrency` instead. If `\datatool_set_numberchars:nnnn` was used, the characters supplied with the `<format number group char>` and `<format decimal char>` arguments will be used.

If the supplied value is not a plain number then a warning will occur and the result will be a string. This is to allow for databases that contain missing value markup, such as “N/A” or `\textendash`.

`\DTLdecimaltocurrency[<currency symbol>]{<num>}{<cs>}`

Converts a plain number *<num>* into a formatted number (as above) with the currency symbol supplied in the optional argument (or the default currency symbol if omitted) and stores the result in *<cs>*. The number of digits will be rounded according to:

`\DTLCurrentLocaleCurrencyDP`

initial: 2

If the expansion text is empty then `\DTLdecimaltocurrency` won't round the result. Otherwise, the expansion text should be the number of decimal places to round to. This command is redefined by localisation hooks.

For example

```
\documentclass{article}
\usepackage[en-GB]{datatool-base}
\begin{document}
\DTLdecimaltocurrency{1234.5672}{\result}
% parse number
Result: \result.
Value: \DTLdatumvalue{\result}.
\end{document}
```

2.3.3. Lexicographical

The commands described in this section are used by string sorting and initial letter commands to enable locale-sensitive functions to be used, if available.

`\DTLCurrentLocaleWordHandler{<cs>}`

This is the current locale word handler used by `\DTLDefaultLocaleWordHandler`. If no localisation support is provided, this command does nothing. If localisation support is added, this handler should make any appropriate adjustments to *<cs>* to convert its content to a byte sequence that will ensure the string is correctly sorted according to the locale's alphabet.

The handler definition will usually depend on the encoding. For example, `datatool-english-utf8.ldf` defines `\DTLenLocaleHandler` and the following is added (indirectly) to the language hook (see §2.3.5):

```
\let\DTLCurrentLocaleWordHandler\DTLenLocaleHandler
```

This allows accented characters, such as “Á”, to be converted to non-accented Basic Latin characters, such as “A”. This command is also defined by `datatool-english-latin1.ldf` and `datatool-english-ascii.ldf` but has less support.

Remember that the purpose of the handler is to convert a string into a byte sequence that reflects the desired ordering. This byte sequence is not intended to be typeset. It’s therefore possible to use ASCII control characters to influence the order. This is the method used by the marker commands, such as `\datatoolpersoncomma`.

For example, suppose you want to provide support for Icelandic, where Áá, Ðð, Éé, Íí, Óó, Úú, Ýý, Þþ, Ææ and Öö are all distinct letters of the alphabet. This means that the method used by the English handler isn’t appropriate.

10

As with the English handler, the punctuation characters can be adjusted to ensure that they are placed before “A”. This means that the final uppercase letters “P”, “Æ” and “Ö” can be reassigned to the character positions after “Z” and the lowercase “p”, “æ” and “ö” can be reassigned to the character positions after “z” (similar to `datatool-ang-Latn.ldf`). The other characters need to be positioned between Basic Latin characters. For example, “Á” needs to be between “A” and “B”. This can be achieved by replacing uppercase “Á” with “A” followed by the control character 0x7F (which is the final ASCII character). Similarly lowercase “á” is replaced by “a” followed by 0x7F and so on.

The language code for Icelandic is “is” so it will be used in the command names. Remember that `\l_datatool_current_language_tl` will need to be redefined to match. (Alternatively, “isl” or “ice” could also be used but the important thing is to be consistent in the event that a region file tries searching for a command name to determine if it’s supported for the current language.)

```
\ExplSyntaxOn
\newcommand {\DTLisLocaleHandler} [ 1 ]
{
  \regex_replace_case_all:nN
  {
    { Á } { A\cL\x{7f} } { á } { a\cL\x{7f} }
    { Ð } { D\cL\x{7f} } { ð } { d\cL\x{7f} }
    { É } { E\cL\x{7f} } { é } { e\cL\x{7f} }
    { Í } { I\cL\x{7f} } { í } { i\cL\x{7f} }
    { Ó } { O\cL\x{7f} } { ó } { o\cL\x{7f} }
    { Ú } { U\cL\x{7f} } { ú } { u\cL\x{7f} }
    { Ý } { Y\cL\x{7f} } { ý } { y\cL\x{7f} }
    { Þ } { \cL\x{5b} } { þ } { \cL\x{7b} }
    { Æ } { \cL\x{5c} } { æ } { \cL\x{7c} }
    { Ö } { \cL\x{5d} } { ö } { \cL\x{7d} }
```

```
% currency signs and punctuation
% [...]
% }
% #1
% }
\ExplSyntaxOff
```

Substitutions for foreign language letters (such as replacing “ß” with “ss”) should be added as applicable. The currency signs and punctuation are as for `\DTLenLocaleHandler`, shown earlier.

For example, the string “az” will be unchanged and has the byte sequence 0x61 0x7A. Whereas the string “áa” will be converted by the above Icelandic handler to the byte sequence 0x61 0x7F 0x61. Since 0x7A is less than 0x7F, “az” comes before “áa”. With the English handler, “áa” will be converted to “aa” which has the byte sequence 0x61 0x61. Since 0x61 is less than 0x7A, “áa” would come before “az”.

Note the use of `\cL` to ensure that the replacement characters have a letter category code (even though they’re not actually letters). This will allow the process to be reversed without changing punctuation characters that were originally present in the sort string (see Example 12).

The language hook (see §2.3.5) then needs to set the locale handler:

```
\let\DTLCurrentLocaleWordHandler\DTLisLocaleHandler
```

You may prefer to use `\renewcommand` if you want to provide options to adjust the handler (as with `datatool-ang-Runr.1df`).

Example 10 uses the above to sort a list of words:

```
\newcommand{\mylist}
{bókstafinn, vera, eða, ég, býsna,
þú, vakna, epli, bragðs, aldar, bað, bolli, ýmist, af,
óáreiðanleg, bær, dalur, ör, þorn, þau, október, esja,
öngull, dæmi, að, yfir, öðrum, orð, detta, áhrif, yngri,
óvinur, ætlað}

\DTLsortwordlist{\mylist}{\DTLsortletterhandler}
Sorted list: \DTLformatlist{\mylist}.
```

Example 10: Icelandic Alphabetic

Sorted list: að, af, aldar, áhrif, bað, bolli, bókstafinn, bragðs, býsna, bær, dalur, detta, dæmi, eða, epli, esja, ég, október, orð, óáreiðanleg, óvinur, vakna, vera, yfir, yngri, ýmist, þau, þorn, þú, ætlað, öðrum, öngull & ör.

```
\DTLCurrentLocaleGetGroupString{<actual>}{<sort value>}{<cs>}
```

This is used by `\DTLassignlettergroup` to set `<cs>` (a token list variable) to the content from which the letter group will be obtained (but only for string data types).

For example, `datatool-english.ldf` sets `<cs>` to the sort value as this ensures that any supported accented characters and ligatures will have already been converted to Basic Latin characters.

However `datatool-ang-Latn.ldf` and `datatool-ang-Runr.ldf` can't do this as the construction of the sort value means that the characters in the sort value may be significantly different from the actual letters. In this case, the original value must be used instead, but it's needs some processing to map extended characters to their equivalent sort group. For example, “*É*” needs to be mapped to “*Æ*”. Additionally, the actual value is likely to need pre-processing with `\datatool_sort_preprocess:Nn`.

```
\DTLCurrentLocaleGetInitialLetter{<text>}{<cs>}
```

This is used by `\DTLGetInitialLetter` and `\DTLassignlettergroup` to obtain the initial letter of the given text. The default definition just uses `\datatool_get_first_letter:nN` which skips leading non-letters.

This command is intended for use with sorting functions to obtain the letter group, so the actual letter returned may not be the initial letter. For example, if the word starts with the ligature “*Æ*” then the localisation may return “*A*” rather than “*Æ*”.

For example, `datatool-english.ldf` defines:

```
\newcommand{\DTLenLocaleGetInitialLetter}[2]{
  \datatool_get_first_letter:nN { #1 } #2
  \DTLenLocaleHandler #2
  \int_compare:nNnT { \tl_count:N #2 } >
{ \c_one_int }
{
  \exp_args:NV \datatool_get_first_
letter:nN #2 #2
}
}
```

and adds the following to the language hook:

```
\let\DTLCurrentLocaleGetInitialLetter
\DTLenLocaleGetInitialLetter
```

(See §2.3.5 for further details.)

Example 11 has a possible implementation for Dutch that will search for “IJ” or “ij”:

11

```
\newcommand{\DTLdutchLocaleGetInitialLetter}[2]{
  \tl_clear:N #2
  \text_map_inline:nn { #1 }
  {
    \tl_if_empty:NTF #2
    {
      \datatool_if_letter:nT { ##1 }
      {
        \tl_set:Nn #2 { ##1 }
        \tl_if_in:nnF { Ii } { ##1 }
      }
    }
    { \text_map_break: }
  }
  {
    \tl_if_in:nnT { Jj } { ##1 }
    {
      \tl_put_right:Nn #2 { ##1 }
    }
    \text_map_break:
  }
}
```

(Note that this will also find “Ij” and “iJ”. Some adjustment is required to exclude those cases.) Suppose that this has been implemented via a language hook (see §2.3.5):

```
\let\DTLCurrentLocaleGetInitialLetter
\DTLdutchLocaleGetInitialLetter
```

Then it will affect commands that fetch an initial letter, such as `\DTLinitials`:

```
IJsselmeer: \DTLinitials{IJsselmeer}
Industrieel: \DTLinitials{Industrieel}
```

```

``IJsselmeer``: \DTLinitials{``IJsselmeer``}
``Industrieel``: \DTLinitials{``Industrieel``}

```

The test for a letter (with `\datatool_if_letter:nT`) ensures that leading punctuation is skipped.

Example 11: IJ-Initial Support

IJsselmeer: IJ. Industrieel: I. “IJsselmeer”: IJ. “Industrieel”: I.

Remember that `\DTLCurrentLocaleGetInitialLetter` is also used to obtain the letter group (but not the non-letter group) from sort values with `\DTLsortwordlist`.

Example 12 adapts the earlier Icelandic Example 10 to show the letter groups. Recall that Example 10 substituted UTF-8 characters for ASCII characters with control codes or punctuation characters used to influencing sorting. This means that, for example, “ý” will be replaced with “y” followed by the control code 0x7F assigned with the letter category code.

12

The content used to obtain the group letter may be either the original (“actual”) string or the sort value. This is determined by `\DTLCurrentLocaleGetGroupString`. For example, `datatool-english.ldf` uses the sort value, since all the extended characters are mapped to Basic Latin letters. In this case, we have some awkward control characters which will mess up the letter group.

There are two ways of dealing with this. The first method is the case used by `datatool-ang-latn.ldf` which defines `\DTLangLatnLocaleGetGroupString`. That starts with the actual value and processes it with `\datatool_sort_preprocess:Nn` and then replaces any leading accented character with the unaccented letter.

The second method is used here. This starts with the sort value and reverses the mapping applied by the handler. In this case, a localisation file that provides `\DTLisLocaleHandler` would also need to provide a way of reversing the substitutions for the letter groups. Since the replacement (non-alphabetic) characters are assigned the letter category code, this makes them easier to distinguish from actual punctuation characters.

```

\newcommand{\DTLisLocaleGetGroupString}[3]{
  \tl_set:Nn #3 { #2 }
  \regex_replace_case_once:nN
  {
    { A\cL\x{7f} } { Á } { a\cL\x{7f} } { á }
    { D\cL\x{7f} } { Đ } { d\cL\x{7f} } { đ }
    { E\cL\x{7f} } { É } { e\cL\x{7f} } { é }
    { I\cL\x{7f} } { Í } { i\cL\x{7f} } { í }
    { O\cL\x{7f} } { Ó } { o\cL\x{7f} } { ó }
    { U\cL\x{7f} } { Ú } { u\cL\x{7f} } { ú }
  }
}

```

```

{ Y\cL\x{7f} } { Ý } { y\cL\x{7f} } { ý }
{ \cL\x{5b} } { Ð } { \cL\x{7b} } { þ }
{ \cL\x{5c} } { Æ } { \cL\x{7c} } { æ }
{ \cL\x{5d} } { Ö } { \cL\x{7d} } { ö }
} #3
}

```

Note that, unlike the handler function, this only needs to perform one replacement as we’re only interested in the start of the string. Unlike the first method (used by `\DTLangLatnLocaleGetGroupString`) we don’t need to worry about whether or not leading hyphens have been stripped. Deciding which method to use comes down to whether it’s more complex to reverse the mapping on the sort value or to process the actual value.

Suppose that this has been implemented via a language hook (see §2.3.5):

```

\let\DTLCurrentLocaleGetInitialLetter
\DTLisLocaleGetInitialLetter

```

Example 10 can now be adapted to show the letter groups:

```

\DTLsortwordlist{\mylist}{\DTLsortletterhandler}
\renewcommand{\DTLlistformatitem}[1]{#1 (\DTLsorted-
letter{#1})}
Sorted list: \DTLformatlist{\mylist}.

```

↑ Example 12: Icelandic Sorting and Letter Groups

Sorted list: að (A), af (A), aldar (A), áhrif (Á), það (B), bolli (B), bók-
stafinn (B), bragðs (B), býsna (B), bær (B), dalur (D), detta (D), dæmi (D),
eða (E), epli (E), esja (E), ég (É), október (O), orð (O), óáreiðanleg (Ó),
óvinur (Ó), vakna (V), vera (V), yfir (Y), yngri (Y), ýmist (Ý), þau (Þ),
þorn (Þ), þú (Þ), ætlað (Æ), öðrum (Ö), öngull (Ö) & ör (Ö).

```

\dtllettergroup{<character>}

```

By default, this expands to `\text_titlecase_first:n{<character>}`. In the case of Dutch, this would need to be changed to use `\text_uppercase:n` instead to ensure that “ij” becomes “IJ” instead of “Ij”.


```
\dtlnonlettergroup{⟨character⟩}
```

By default, this simply expands to $\langle character \rangle$. A language file may redefine this to produce a textual title. For example, “Symbols”.

For the Icelandic word sort handler in Example 10, the $\langle character \rangle$ will always be the double-quote " because of the final substitution case in the regular expression. For the handler provided in `datatool-english-utf8.ldf` (see §2.3.5), the character will either be a double-quote " or a literal dollar \$ (with category code other).

```
\dtlnumbergroup{⟨num⟩}
```

(Only used with `sort-datum={true}`.) By default, this simply expands to $\langle num \rangle$. A language file may redefine this to produce a textual title. For example, “Numbers”.

```
\dtlcurrencygroup{⟨sym⟩}{⟨num⟩}
```

(Only used with `sort-datum={true}`.) By default, this simply expands to $\langle sym \rangle$. A language file may redefine this to produce a textual title. For example, “Currency”.

2.3.4. Adding New Region Support

The language-independent region files are all bundled together in a single distribution `datatool-regions` which is separate from the core `datatool` distribution and available on GitHub (<https://github.com/nlct/datatool-regions>). There are currently only a limited number of regions supported but more can be added via a pull request and only the `datatool-regions` collection need be uploaded, without the extra overhead of producing a new version of `datatool`.

There is an interactive Perl script on GitHub that will create a `datatool-⟨region⟩.ldf` file based on your responses.

The region file deals with setting the default currency, number group character and decimal character, and also the numeric date formats for use with `parse=region` or `parse=iso+region`. Note that any date formats that have textual parts (such as month names) should be dealt with by the language support.

A more specific `datatool-⟨lang⟩-⟨region⟩.ldf` file may be used to override any of these settings but that file should be provided with the corresponding language support (see §2.3.5). For example, `datatool-english` provides `datatool-en-CA.ldf` to set the number group character and decimal character since it varies according to the language for that region.



For further details, see the `datatool-regions` documentation.

2.3.5. Adding New Language Support

The `datatool-english` package (distributed separately) may be used as an example. (The `datatool-english` bundle includes `databib-english.ldf` to provide localisation support for the `databib` package, and `person-english.ldf` to provide localisation support for the `person` package, see §§7.11 & 9.7.3 for further details.)

The `datatool-english` bundle also includes limited support for Old English (Anglo-Saxon) for Latin and Runic scripts, which may be used as examples for extended Latin or non-Latin languages.

The language file should be called `datatool-⟨language⟩.ldf` where `⟨language⟩` is the root language label (tracklang label). Using the root language label ensures that it's the last in tracklang's file search list, which means that it can be overridden by a more specific label, if required. So in the event that there is some particular language setting that is specific to a particular region, a language module may also include a file named `datatool-⟨lang⟩-⟨region⟩.ldf` where `⟨lang⟩` is the language code (such as “fr”) and `⟨region⟩` is the region code (such as “CA”). For example:



```
\TrackLangProvidesResource{fr-CA}
\TrackLangRequireResource{french}
\ExplSyntaxOn
\newcommand\datatoolfrCASetNumberChars
{
  \bool_if:NT \l_datatool_region_set_numberchars_bool

    \DTLsetnumberchars{.}{,}
  % number group and decimal symbol

}
\newcommand\DTLfrCALocaleHook
{
  \datatoolfrCASetNumberChars
}
\ExplSyntaxOff
\TrackLangAddToCaptions{\DTLfrCALocaleHook}
```

The `datatool-english` distribution provides a similar `datatool-en-CA.ldf` file.

In the case of `datatool-english`, the root language label is “english” (even if the language has been specified using a dialect label, such as “british”) so the filename is `datatool-english.ldf`. The file needs to identify itself (analogous to `\ProvidesPackage` for

packages):

```
\TrackLangProvidesResource{<language>}
[<yyy>/<mm>/<dd> v<version>]
```

Although pdfL^AT_EX now defaults to UTF-8, it can be helpful to provide some support for other encodings. The document encoding (as detected by tracklang) can be obtained by expanding `\TrackLangEncodingName` (`\inputencoding` isn't guaranteed to be defined).

The *datatool-english* bundle includes (limited) support for ISO-8859-1 (Latin-1) and ASCII in addition to UTF-8. The encoding support is provided in the files *datatool-english-latin1.ldf*, *datatool-english-ascii.ldf* and *datatool-english-utf8.ldf*. The following code will input the appropriate file or fallback on the ASCII file if the encoding isn't supported:

```
\TrackLangRequestResource{english-\TrackLangEncoding-
Name}
{
  \TrackLangRequireResource{english-ascii}
}
```

Note the difference between requesting a resource and requiring it.

Compare this with the Anglo-Saxon support. The root language label is “anglosaxon” so there is a file called *datatool-anglosaxon.ldf* but because there are two different scripts to cater for, it just ensures that the appropriate file is loaded.

```
\TrackLangProvidesResource{anglosaxon}
\TrackLangRequestResource
{ang-\CurrentTrackedDialectScript-\TrackLang-
EncodingName}
{%
  \PackageWarning{datatool-anglosaxon}{%
    {%
      No support for `anglosaxon' with script
      `\'CurrentTrackedDialectScript'
      and encoding `\'TrackLangEncodingName'%
    }%
  }%
}
```

This file is actually just a fallback as the files *datatool-ang-Latn.ldf* and *datatool-ang-Runr.ldf* should be found first. Note that the script indicates the script of the input or source text. That is, the text used in the document source code, which may not correspond to the glyphs visible in the PDF file.

For example, a package may provide a command called, say `\runic`, which expects Latin characters in the argument but the font encoding ensures that those characters appear as runes in

the PDF. In this case, the source is Latin and so “ang-Latn” is needed when specifying the locale.

If, however, the source code actually contains characters from the Runic Unicode block (with an appropriate font that supports those characters), the source is Runic and so “ang-Runr” is needed when specifying the locale.

The files `datatool-ang-Latn.ldf` and `datatool-ang-Runr.ldf` are similar to `datatool-english.ldf` but, in these cases, there’s no fallback to ASCII as it doesn’t cover all characters from the Latin script and doesn’t cover any for the Runic script. Instead, if the encoding isn’t supported, then no localisation can be provided. For example, `datatool-ang-Latn.ldf` starts with:

```
\TrackLangProvidesResource{ang-Latn}
\TrackLangRequestResource{ang-Latn-\TrackLang-
EncodingName}
{%
  \PackageWarning{datatool-ang-Latn}%
  {%
    No support for `anglosaxon' with script `Latn'
    and encoding `\'TrackLangEncodingName'.'.
  }%
\endinput
}
```

The code for `datatool-ang-Runr.ldf` is similar. Only UTF-8 is supported (`datatool-ang-Latn-utf8.ldf` and `datatool-ang-Runr-utf8.ldf`), but this method allows for other encodings to be added by simply creating a file with an appropriate name.

For both the English and Old English support, we will be using some $\text{\LaTeX}$ syntax, so the appropriate category codes must be changed:

```
\ExplSyntaxOn
```

The definition of `\DTLenLocaleGetGroupString` ensures that the letter group is obtained from the sort value rather than the actual value:

```
\newcommand\DTLenLocaleGetGroupString[3]
{
  \tl_set:Nn #3 { #2 }
}
```

This ensures that the accents are stripped, but it will mean that the currency and punctuation marks will have their initial marker that’s inserted by the handler function `\DTLenLocaleHandler`. Bear in mind that `\DTLenLocaleGetGroupString` is only used for values that have been identified as strings. It’s not used by other data types. The non-letter characters used to alter the order of currency and punctuation marks is usually not relevant, as the non-letter group title (`\dtlnonlettergroup`) typically ignores the character.

2. Base Commands (*datatool-base package*)

This conveniently works for English, which just maps extended characters to Basic Latin letters (A–Z, a–z), but will cause a problem for Anglo-Saxon, both Latin and Runic. In the case of `datatool-ang-Latn.ldf`, the extended characters $\P$ (wynn), $\eth$, $\AE$ (AE-ligature), $\P$ (thorn) are converted to the character codes following “Z” and, similarly, the lowercase $\p$, $\eth$, $\ae$, $\p$ are converted to the character codes following “z”. This means that if the sort value is used to obtain the letter group, then these extended characters will be assigned to the non-letter group.

Therefore, it’s necessary to use the actual value rather than the sort value, but some additional processing is required to ensure that characters with diacritics are placed in the same group as the unaccented character. For example, “ $\acute{A}$ ” needs to be mapped to “ $\AE$ ”. This is performed by a low-level function that performs a regular expression substitution.

Note that this doesn’t take into account a sort handler that strips content, such as the letter handler functions that remove spaces and hyphens. This will cause a problem for any words that start with a hyphen. Since the handler function `\DTLangLatnLocaleHandler` inserts a double-quote character in front of any punctuation, it’s possible to check if the actual value starts with a hyphen and if the sort value starts with a double-quote then the hyphen likely wasn’t stripped so it can be removed. This is done as follows:

```
\newcommand \DTLangLatnLocaleGetGroupString { 3 }
{
  \tl_set:Nn #3 { #1 }
  \datatool_angLatn_process_letter_group:N #3
  \bool_lazy_and:nnT
    { \tl_if_head_eq_charcode_p:nN { #1 } - }
    { \bool_not_p:n { \tl_if_head_eq_charcode_p:nN
{ #2 } " } }
    {
      \exp_args:NNe \tl_set:Nn #3 { \tl_tail:N #3 }
    }
}
```

In the case of `datatool-ang-Runr.ldf` there are no hyphens to worry about so it’s far simpler to just assign the token list variable to the actual value. Any further processing is down to whether or not the sort handler considers multiple runes to be considered equivalent for sorting purposes.

For both English and the two different scripts of Old English, the support for `\DTLCurrentLocaleGetInitialLetter` is the same as the default definition provided by `datatool-base`. For example, `datatool-english.ldf` defines:

```
\newcommand \DTLenLocaleGetInitialLetter [ 2 ]
{
  \datatool_get_first_letter:nN { #1 } #2
}
```

2. Base Commands (*datatool-base* package)

The only other support provided by `datatool-ang-Latn.ldf` and `datatool-ang-Runr.ldf` is to redefine `\DTLandname` to use the Tironian et.

Returning to `datatool-english.ldf`, support is provided to produce textual labels for the non-letter group, number group, currency group and temporal group commands:

```
\newcommand \DTLenSetLetterGroups
{
  \renewcommand \dtllettergroup [ 1 ]
    { \text_titlecase_first:n { ##1 } }
  \renewcommand \dtlnonlettergroup [ 1 ] { Symbols }
  \renewcommand \dtlnumbergroup [ 1 ] { Numbers }
  \renewcommand \dtlcurrencygroup [ 2 ] { Currency }
  \renewcommand \dtldatetimegroup [ 1 ]
{ Timestamps }
  \renewcommand \dtldategroup [ 1 ] { Dates }
  \renewcommand \dtltimegroup [ 1 ] { Times }
}
```

Aside from the above, the fixed-text commands for `datatool-base` are `\DTLandname`, `\DTLdatatypeunsetname`, `\DTLdatatypestringname`, `\DTLdatatypeintegername`, `\DTLdatatypedecimalname`, `\DTLdatatypecurrencyname`, `\DTLdatatypedatetimenname`, `\DTLdatatypedatename`, `\DTLdatatypepetimenname`, and `\DTLdatatypeinvalidname`.

(Some of the supplementary packages have additional fixed-text commands, but they are dealt with in their own `ldf` files.) An intermediate command is defined to set `\DTLandname`:

```
\newcommand \DTLenSetAndName
{
  \renewcommand \DTLandname { and }
}
```

This makes it easier to for the supplied option to redefine it:

2. Base Commands (*datatool-base* package)

```
\datatool_locale_define_keys:nn { en }
{
  and .choice:,
  and / word .code:n =
  {
    \renewcommand \DTLenSetAndName
    {
      \renewcommand \DTLandname { and }
    }
    \tl_if_eq:NnT \l_datatool_current_language_tl
{ en }
    { \DTLenSetAndName }
  } ,
  and / amp .code:n =
  {
    \renewcommand \DTLenSetAndName
    {
      \renewcommand \DTLandname { \& }
    }
    \tl_if_eq:NnT \l_datatool_current_language_tl
{ en }
    { \DTLenSetAndName }
  } ,
}
```

This is added to the hook that sets all the *datatool-base* textual commands:

```
\newcommand \DTLenTranslations
{
  \DTLenSetAndName
  \renewcommand \DTLdatatypeunsetname { unset }
  \renewcommand \DTLdatatypestringname { string }
  \renewcommand \DTLdatatypeintegername { integer }
  \renewcommand \DTLdatatypedecimalname { decimal }
  \renewcommand \DTLdatatypecurrencyname { currency }

  \renewcommand \DTLdatatypedatetimename { date-
time }
  \renewcommand \DTLdatatypedatetimename { date }
  \renewcommand \DTLdatatypetimename { time }
  \renewcommand \DTLdatatypeinvalidname { invalid }
}
```

2. Base Commands (*datatool-base* package)

After that comes the support for date and time formatting, but it's still experimental.

As with the region `datatool-GB.ldf` file, describe in §2.3.4, a single intermediate command is defined that will be added to the captions hook:

```
\newcommand \DTLenLocaleHook
{
  \renewcommand
    \DTLCurrentLocaleWordHandler
    { \DTLenLocaleHandler }
  \renewcommand
    \DTLCurrentLocaleGetInitialLetter
    { \DTLenLocaleGetInitialLetter }
  \renewcommand
    \DTLCurrentLocaleGetGroupString
    { \DTLenLocaleGetGroupString }
  \DTLenSetLetterGroups
% date and time assignments
% [...]
\tl_set:Nn \l_datatool_current_language_tl { en }
% Fixed text command:
  \DTLenTranslations
}
\ExplSyntaxOff
\TrackLangAddToCaptions{\DTLenLocaleHook}
```

If `babel` or `polyglossia` have been loaded, this will add `\DTLenLocaleHook` to the `\captions-⟨dialect⟩` hook. The command will be implemented at this point as well, which will make it the current setting if there's no hook.

Note that each language file should ensure that the caption hook sets the token list variable:


`\l_datatool_current_language_tl` *initial: empty*

to expand to the language code (as above). This may then be referenced by the region file, if necessary. Note that it's used for checking control sequence names to test if the language provides support for particular settings, therefore don't include a hyphen as it will make it harder to define the appropriate commands. For example, `datatool-ang-Latn.ldf` has:

```
\tl_set:Nn \l_datatool_current_language_tl
{ angLatn }
```

and `datatool-ang-Runr.ldf` has:


```
\tl_set:Nn \l_datatool_current_language_tl
{ angRunr }
```

The locale handlers are provided in the encoding files. For example, `\DTLenLocaleHandler` is provided in `datatool-english-utf8.ldf`, `datatool-english-latin1.ldf` and `datatool-english-ascii.ldf`. This is used to convert strings into byte sequences for lexicographical comparisons. For example, `datatool-english-utf8.ldf` replaces common extended Latin characters into the nearest ASCII equivalent, suitable for English ordering. This can conveniently be done with regular expression replacement.

```
\cs_new:Npn \DTLenLocaleHandler #1
{
  \regex_replace_case_all:nN
  {
    % alphabetical cases
    % [ ... ]
    { (\ur{l_datatool_currencysigns_regex}) }
  { \cO\x{24}\1 }
    { ' } { \cO" ' }
    { ‘ } { \cO" ` }
    { (“|”) } { \cO" \cO" }
    { (–|–) } { \cO" – }
    { ([[:punct:]]+) } { \cO" \1 }
  }
  #1
}
```

The final substitutions are for currency and any punctuation and is designed to gather together currency symbols and punctuation marks. (Otherwise they would be in their character code order which would spread them before and after letters.) Note that character classes such as `[[:punct:]]` and `[[:alpha:]]` only apply to Basic Latin characters. (The use of `\cO` ensures that the next character has category code “other”.)



The more complex the regular expression cases, the longer the document build time. There needs to be a trade-off between likely characters to support and processing time.

In the case of a non-Latin script, such as Runic, the conversion simply ensures that the characters follow the appropriate order when the character codes are compared. For example, `datatool-ang-Runr.ldf` provides two different ways of ordering the runes. The first mostly follows the order in the Runic Unicode block. So feoh (U+16A0) is mapped to character code 31, Runic V (U+16A1) is mapped to character code 32, etc. The second follows the Old English rune poem order (fuporc) so feoh (U+16A0) is mapped to character code 31, ur (U+16A2) is mapped to

character code 32, thorn (U+16A6) is mapped to character code 33, etc.

2.4. Conditionals

There are two types of conditional commands provided by *datatool-base*: those with $\{\langle true \rangle\}$ $\{\langle false \rangle\}$ arguments (such as `\DTLifint`) or case arguments (such as `\DTLifcasedata-type`) and those that are designed to be used in the conditional part of `\ifthenelse` (provided by the *ifthen* package). The first type have command names that start “DTLif” or “dtlif” and are described in §2.4.1, and the second type have command names starting “DTLis” and are described in §2.4.2.

2.4.1. If-Else or Case Conditionals

The robust commands listed in §2.4.1.2, such as `\DTLifstringeq`, treat their arguments as strings. For example, `\DTLifstringlt` is a test if one string is lexicographical less than another.

The robust numeric “DTLif” commands listed in §2.4.1.3, such as `\DTLifnumeq`, expect formatted numbers or datum control sequences in the numeric arguments. If you know that all your values are plain numbers, the “dtlif” listed in §2.4.1.4 commands are quicker.

Numeric commands listed in §2.4.1.4, such as `\dtlifnumeq`, don’t parse for the current decimal character and number group character or for a currency symbol. They require a plain number, either a bare integer (such as 12345) or a number with a decimal point (such as 1234.5). These commands are listed as being provided by *datatool-base*, but are actually defined in the maths processor file `datatool-processor.def` corresponding to the value of the `math` package option. With `math=13fp` or `math=lua`, these commands are expandable but with `math=fp` or `math=pgfmath` they are robust. Note that the `fp` package doesn’t support scientific notation.

The multi-type robust commands listed in §2.4.1.5, such as `\DTLifeq`, parse the arguments to determine the data type and then use the corresponding command from §2.4.1.3 or §2.4.1.2.

2.4.1.1. Data Type Conditionals

The commands described in this section test the data type of the argument according to the current settings for the number group character and decimal character and recognised currency symbols.

Note that you can also use `\DTLdatumtype` on a datum control sequence (obtained with `\DTLparse` or `\DTLxparse`) to determine the data type.

```
\DTLifint{\arg}{\true}{\false}
```

Parses $\langle arg \rangle$ and does $\langle true \rangle$ if $\langle arg \rangle$ is an integer formatted number, otherwise it does $\langle false \rangle$. Note that if $\langle arg \rangle$ is a decimal or currency this command will do $\langle false \rangle$. The number group character is optional but, if present, it must be at intervals of three digits (from the right). See Example 13.

```
\DTLifreal{\langle arg \rangle}{\langle true \rangle}{\langle false \rangle}
```

Parses $\langle arg \rangle$ and does $\langle true \rangle$ if $\langle arg \rangle$ is a real (decimal) formatted number or is in scientific notation, otherwise it does $\langle false \rangle$. Note that if $\langle arg \rangle$ is an integer or currency this command will do $\langle false \rangle$ (even though integers are technically a subset of real numbers). The number group character is optional but, if present, it must be at intervals of three digits (left of the decimal character). See Example 14.

```
\DTLifcurrency{\langle arg \rangle}{\langle true \rangle}{\langle false \rangle}
```

Parses $\langle arg \rangle$ and does $\langle true \rangle$ if $\langle arg \rangle$ is a currency formatted number, otherwise it does $\langle false \rangle$ (see Example 15). Note that if $\langle arg \rangle$ is an integer or decimal without a currency prefix this command will do $\langle false \rangle$.

```
\DTLifcurrencyunit{\langle arg \rangle}{\langle symbol \rangle}{\langle true \rangle}{\langle false \rangle}
```

Parses $\langle arg \rangle$ and does $\langle true \rangle$ if $\langle arg \rangle$ is a recognised currency formatted number and uses the currency $\langle symbol \rangle$, otherwise it does $\langle false \rangle$ (see Example 15). Note that if $\langle arg \rangle$ is an integer or decimal this command will do $\langle false \rangle$. Rather than repeatedly parsing the same $\langle arg \rangle$, you may prefer to use `\DTLparse`.

```
\DTLifnumerical{\langle arg \rangle}{\langle true \rangle}{\langle false \rangle}
```

Parses $\langle arg \rangle$ and does $\langle true \rangle$ if $\langle arg \rangle$ is numerical, otherwise it does $\langle false \rangle$, where numerical means a formatted number that may be an integer, real number, currency or temporal (see Example 16).

```
\DTLiftemporal{\langle arg \rangle}{\langle true \rangle}{\langle false \rangle}
```

Parses $\langle arg \rangle$ and does $\langle true \rangle$ if $\langle arg \rangle$ is temporal, otherwise it does $\langle false \rangle$, where temporal means a timestamp (date, time and, optionally, a time zone), a date (year, month, and day) or time (hours, minutes, and, optionally, seconds). Temporal types are considered numerical and may be used in numerical calculations but the result will be in UTC+0 for timestamps.

```
\DTLifstring{\langle arg \rangle}{\langle true \rangle}{\langle false \rangle}
```

Parses $\langle arg \rangle$ and does $\langle true \rangle$ if $\langle arg \rangle$ is a string, otherwise it does $\langle false \rangle$. This is essentially like the reverse of `\DTLifnumerical` except in the case of an empty argument, which has an unknown type, and so is neither numerical nor a string. See Example 17.

`\DTLifcasedatatype{ $\langle arg \rangle$ }{ $\langle string case \rangle$ }{ $\langle int case \rangle$ }{ $\langle real case \rangle$ }{ $\langle currency case \rangle$ }`

This command parses $\langle arg \rangle$ and does $\langle string case \rangle$ if $\langle arg \rangle$ is a string, $\langle int case \rangle$ if $\langle arg \rangle$ is an integer, $\langle real case \rangle$ if $\langle arg \rangle$ is a real number (decimal) or $\langle currency case \rangle$ if $\langle arg \rangle$ is a currency (according to the current number group character, decimal character and known currency symbols). Note that an empty argument, which has an unknown type, or a temporal value will do nothing. See Example 18. This command is retained for backward-compatibility but lacks the ability to detect new data types.

2.4.1.1.1. Test if Integer Example

Example 13 uses `\DTLifint` to determine if the argument is an integer according to the current localisation setting.

13

```
2536: \DTLifint{2536}{integer}{not an integer}.
2536.0: \DTLifint{2536.0}{integer}{not an integer}.
2,536: \DTLifint{2,536}{integer}{not an integer}.
2,5,3,6: \DTLifint{2,5,3,6}{integer}{not an integer}.
.
\DTLparse{\numA}{2,536}
\numA: \DTLifint{\numA}{integer}{not an integer}.
\DTLsetnumberchars{.}{,}%
2,536: \DTLifint{2,536}{integer}{not an integer}.
2.536: \DTLifint{2.536}{integer}{not an integer}.
\numA: \DTLifint{\numA}{integer}{not an integer}.
```

 Example 13: Test for Integer Value


```
2536: integer.
2536.0: not an integer.
2,536: integer.
2,5,3,6: not an integer.
2,536: integer.
2,536: not an integer.
2.536: integer.
2,536: integer.
```

Note that the datum control sequence `\numA` is still identified as an integer after `\DTLset-numberchars` even though it uses the original number group character and decimal character. This is because once the datum control sequence has had its data type set there's no need to reparse its value.

2.4.1.1.2. Test if Decimal Example

Example 14 uses `\DTLifreal` to determine if the argument is a decimal according to the current localisation setting. Note that although integers are a subset of real numbers, this test will only be true if the argument has a fractional part or is in scientific notation.

 14



```
1000.0: \DTLifreal{1000.0}{real}{not real}.
1,000: \DTLifreal{1,000}{real}{not real}.
1,000.0: \DTLifreal{1,000.0}{real}{not real}.
1e+3: \DTLifreal{1e+3}{real}{not real}.

\DTLsetnumberchars{.}{,}%
1,000.0: \DTLifreal{1,000.0}{real}{not real}.
1.000,0: \DTLifreal{1.000,0}{real}{not real}.
```

 Example 14: Test for Real Value


```
1000.0: real.
1,000: not real.
1,000.0: real.
1e+3: real.
1,000.0: not real.
1.000,0: real.
```

2.4.1.1.3. Test if Currency Example

Example 15 uses `\DTLifcurrency` and `\DTLifcurrencyunit` to determine if the argument is a currency value or a currency symbol according to the current localisation setting and defined currency symbols.

 15



```
\$5.99: \DTLifcurrency{\$5.99}{currency}
{not currency}.

\DTLcurrency{5.99}:
\DTLifcurrency{\DTLcurrency{5.99}}{currency}
{not currency}.

\pounds5.99:
\DTLifcurrency{\pounds5.99}{currency}{not currency}.

\textsterling5.99:
\DTLifcurrency{\textsterling5.99}{currency}
{not currency}.

\$6.99:
\DTLifcurrencyunit{\$6.99}{\$}{dollars}{not dollars}
.

\newcommand{\cost}{\pounds10.50}%
\cost: \DTLifcurrencyunit{\cost}{\pounds}{pounds}
{not pounds}.

US\$5.99:
\DTLifcurrency{US\$}{currency}{not currency}.

\DTLnewcurrencysymbol{US\$}%
```

```
US\$5.99:
\DTLifcurrency{US\$}{currency}{not currency}.
```

↑ Example 15: Test for Currency



```
$5.99: currency.
$5.99: currency.
£5.99: currency.
£5.99: not currency.
$6.99: dollars.
£10.50: pounds.
US$5.99: not currency.
US$5.99: not currency.
```

2.4.1.1.4. Test if Numerical Example

Example 16 uses `\DTLifnumerical` to determine if the argument is numerical (integer, real or currency value) according to the current localisation setting and defined currency symbols.

16

```
1,234: \DTLifnumerical{1,234}{numeric}{not numeric}.

1,234.0: \DTLifnumerical{1,234.0}{numeric}
{not numeric}.

\$1,234.0:
\DTLifnumerical{\$1,234.0}{numeric}{not numeric}.

1.234,0: \DTLifnumerical{1.234,0}{numeric}
{not numeric}.

\DTLsetnumberchars{.}{,}%
1.234,0: \DTLifnumerical{1.234,0}{numeric}
{not numeric}.

Empty: \DTLifnumerical{}{numeric}{not numeric}.
```

Example 16: Test for Numerical

```
1,234: numeric.
1,234.0: numeric.
$1,234.0: numeric.
1.234,0: not numeric.
1.234,0: numeric.
Empty: not numeric.
```

2.4.1.1.5. Test if String Example

Example 17 uses `\DTLifstring` to test if the argument is considered a string (that is, not numeric and not empty).

17

```
1,234: \DTLifstring{1,234}{string}{not string}.
\n$1,234.0: \DTLifstring{\n$1,234.0}{string}
{not string}.
1,2,3,4: \DTLifstring{1,2,3,4}{string}{not string}.
Empty: \DTLifstring{}{string}{not string}.
```

Example 17: Test for Strings

```
1,234: not string.
$1,234.0: not string.
1,2,3,4: string.
Empty: not string.
```

2.4.1.1.6. Test Data Type Example

Example 18 uses `\DTLifcasedatatype` to determine the data type of its argument, according to the current localisation setting and known currency symbols.

18

```
1,234: \DTLifcasedatatype{1,234}{string}{int}{real}
{currency}.
```



```
1,234.0: \DTLifcasedatatype{1,234.0}{string}{int}
{real}{currency}.

\$1,234: \DTLifcasedatatype{\$1,234}{string}{int}
{real}{currency}.

1,2,3,4: \DTLifcasedatatype{1,2,3,4}{string}{int}
{real}{currency}.

Empty: \DTLifcasedatatype{}{string}{int}{real}
{currency}.
```

↑ Example 18: Test for Data Type

```
1,234: int.
1,234.0: real.
$1,234: currency.
1,2,3,4: string.
Empty: .
```

2.4.1.2. String and List Conditionals

The implementation of these commands has changed in v3.0. You may find different behaviour in certain cases. You can rollback if necessary (see §1.1). The *datatool-base* package no longer loads the *substr* package. If you want to use any commands provided by that package you will need to load it separately.

```
\DTLifinlist{<element>}{<list>}{<true>}{<false>}
```

Does *<true>* if *<element>* is an element of the CSV *<list>*, otherwise does *<false>*. The *<list>* may be a command whose definition is a CSV list (see §2.9). No expansion on *<element>*. See Example 19.

The following comparison commands test for lexicographically equality, less than (comes before) and greater than (comes after). The string arguments have a single expansion applied on the first token and then they are expanded in the same way as for `\dtlcompare` and `\dtlcompare`, taking into account the `compare` settings (see Example 20).

`\DTLifstringeq{⟨str1⟩}{⟨str2⟩}{⟨true⟩}{⟨false⟩}`

modifier: *

Does *⟨true⟩* if *⟨str1⟩* is lexicographically equal to *⟨str2⟩*. This command is robust. The starred version ignores case (see Example 20).

`\DTLifstringlt{⟨str1⟩}{⟨str2⟩}{⟨true⟩}{⟨false⟩}`

modifier: *

Does *⟨true⟩* if *⟨str1⟩* is lexicographically less than (comes before) *⟨str2⟩*. This command is robust. The starred version ignores case (see Example 21).

`\DTLifstringgt{⟨str1⟩}{⟨str2⟩}{⟨true⟩}{⟨false⟩}`

modifier: *

Does *⟨true⟩* if *⟨str1⟩* is lexicographically greater than (comes after) *⟨str2⟩*. This command is robust. The starred version ignores case (see Example 22).

`\DTLifstringopenbetween{⟨str⟩}{⟨min⟩}{⟨max⟩}{⟨true⟩}{⟨false⟩}`

modifier: *

Does *⟨true⟩* if *⟨str⟩* is lexicographically between *⟨min⟩* and *⟨max⟩*, but is not equal to *⟨min⟩* or *⟨max⟩*. This command is robust. The starred version ignores case (see Example 23).

`\DTLifstringclosedbetween{⟨str⟩}{⟨min⟩}{⟨max⟩}{⟨true⟩}{⟨false⟩}`

modifier: *

Does *⟨true⟩* if *⟨str⟩* is lexicographically between *⟨min⟩* and *⟨max⟩*, inclusive. This command is robust. The starred version ignores case (see Example 23).

`\DTLifSubString{⟨string⟩}{⟨fragment⟩}{⟨true⟩}{⟨false⟩}`

modifier: *

Does *⟨true⟩* if *⟨fragment⟩* is a substring of *⟨string⟩* otherwise does *⟨false⟩*. This command purifies the string and fragment before searching for the substring. This command is robust. The starred version is case-insensitive. A space character, ~, `\nobreakspace` and `\space` are considered identical (see Example 24). Note that this does not take category codes into account.

`\DTLifStartsWith{⟨string⟩}{⟨fragment⟩}{⟨true⟩}{⟨false⟩}`

modifier: *

Similar to `\DTLifSubString` but tests if *⟨string⟩* starts with *⟨fragment⟩* (see Example 25). Note that this does not take category codes into account.



A bug in earlier versions of datatool–base meant that `\DTLifStartsWith` didn't ignore commands despite the documentation. This has now been corrected in v3.0.



`\DTLifEndsWith{⟨string⟩}{⟨fragment⟩}{⟨true⟩}{⟨false⟩}` *modifier: **

Similar to `\DTLifSubString` but tests if `⟨string⟩` ends with `⟨fragment⟩`. The starred version is case-insensitive. Note that this does not take category codes into account.



`\DTLifAllUpperCase{⟨string⟩}{⟨true⟩}{⟨false⟩}`

Does `⟨true⟩` if `⟨string⟩` contains only uppercase characters (disregarding punctuation and spaces), otherwise does `⟨false⟩`. The `⟨string⟩` is expanded before testing. This command is robust (see Example 26).



`\DTLifAllLowerCase{⟨string⟩}{⟨true⟩}{⟨false⟩}`

Does `⟨true⟩` if `⟨string⟩` contains only lowercase characters (disregarding punctuation and spaces), otherwise does `⟨false⟩`. The `⟨string⟩` is expanded before testing. This command is robust.



Robust commands like `\emph` are disregarded by the all upper/lower case conditionals, as illustrated in Example 27.

2.4.1.2.1. Element in List Example

Example 19 defines the following commands:

19



```
\newcommand{\goose}{goose}
\newcommand{\mylist}{duck,\goose,{ant},zebra}
```

`\DTLifinlist` is used to determine if certain items are the list:



```
`ant' in list? \DTLifinlist{ant}{\mylist}{true}
{false}.
```

The following tests if “goose” is an element of the list. This is false, because the actual element is `\goose`. The `\mylist` command is only expanded once not fully.

```

`goose' in list?  \DTLifinlist{goose}{\mylist}{true}
{false}.

`\goose' in list?  \DTLifinlist{\goose}{\mylist}
{true}{false}.

`duck' in list?  \DTLifinlist{duck}{\mylist}{true}
{false}.

`zebra' in list?  \DTLifinlist{zebra}{\mylist}{true}
{false}.

```

↑ Example 19: Testing if an Element is in a Comma-Separated List



```

`ant' in list? true.
`goose' in list? false.
`goose' in list? true.
`duck' in list? true.
`zebra' in list? true.

```

2.4.1.2.2. String Equality Example

Example 20 defines two commands that expand to “zebra” and “Zebra”.

20

```

\newcommand{\strA}{zebra}
\newcommand{\strB}{Zebra}

```

The initial first token expansion will expand these commands once before applying the rules according to the current `compare` setting.

```

`\strA' is
\DTLifstringeq{\strA}{\strB}{the same}{not the same}
as `\strB' (case).

`\strA' is
\DTLifstringeq*{\strA}{\strB}{the same}{not the same}
as `\strB' (no case).

```

2. Base Commands (datatool-base package)

```
\strA' is
\DTLifstringeq{\strA}{zebra}{the same}{not the same}
as `zebra' (case).
```

The command `\emph` is robust so it won't be expanded by the initial expand first token action in the following:

```
\emph{ant}' is
\DTLifstringeq{\emph{ant}}{ant}{the same}
{not the same}
as `ant'.
```

The default `expand-cs=false` and `skip-cs=false` settings mean that commands won't be skipped in the comparison. Note the difference when the setting is changed:

```
\DTLsetup{compare={skip-cs}}
\emph{ant}' is
\DTLifstringeq{\emph{ant}}{ant}{the same}
{not the same}
as `ant' (skip cs).
```

Only the first token is expanded, so `\strA` isn't expanded in the initial step:

```
`ant zebra' is
\DTLifstringeq{ant zebra}{ant \strA}{the same}
{not the same}
as `ant \strA' (no expansion).
```

With `expand-cs=true`, expansion will be applied in the second step:

```
\DTLsetup{compare={expand-cs}}
`ant zebra' is
\DTLifstringeq{ant zebra}{ant \strA}{the same}
{not the same}
as `ant \strA' (expansion).
```

Example 20: String Equality Tests

‘zebra’ is not the same as ‘Zebra’ (case).
 ‘zebra’ is the same as ‘Zebra’ (no case).
 ‘zebra’ is the same as ‘zebra’ (case).
 ‘ant’ is not the same as ‘ant’.
 ‘ant’ is the same as ‘ant’ (skip cs).
 ‘ant zebra’ is not the same as ‘ant zebra’ (no expansion).
 ‘ant zebra’ is the same as ‘ant zebra’ (expansion).

2.4.1.2.3. String Less Than Example

Example 21 uses `\DTLifstringlt` to determine if one string is “less than” (comes before) another.

```

`aardvark' is \DTLifstringlt{aardvark}{Zebra}{before}
{after}
`Zebra' (case).

`aardvark' is \DTLifstringlt*{aardvark}{Zebra}
{before}{after}
`Zebra' (no case).
  
```

Example 21: String Less Than

‘aardvark’ is after ‘Zebra’ (case).
 ‘aardvark’ is before ‘Zebra’ (no case).

2.4.1.2.4. String Greater Than Example

Example 22 produces the same result as Example 21 but tests for “greater than” (comes after) instead:

```

`aardvark' is \DTLifstringgt{aardvark}{Zebra}{after}
{before}
`Zebra' (case).

`aardvark' is \DTLifstringgt*{aardvark}{Zebra}{after}
{before}
  
```

```
`Zebra' (no case).
```

↑ Example 22: String Greater Than



‘aardvark’ is after ‘Zebra’ (case).
‘aardvark’ is before ‘Zebra’ (no case).

2.4.1.2.5. String Between Two Strings Example

Example 23 tests if a string is lexicographically between two other strings:

23

```
`duck' lies between `Duck' and `Duckling'
(exclusive, case)?
\DTLifstringopenbetween{duck}{Duck}{Duckling}{true}
{false}.

`duck' lies between `Duck' and `Duckling'
(exclusive, no case)?
\DTLifstringopenbetween*{duck}{Duck}{Duckling}{true}
{false}.

`duck' lies between `Duck' and `Duckling'
(inclusive, case)?
\DTLifstringclosedbetween{duck}{Duck}{Duckling}{true}
{false}.

`duck' lies between `Duck' and `Duckling'
(inclusive, no case)?
\DTLifstringclosedbetween*{duck}{Duck}{Duckling}
{true}{false}.
```

↑ Example 23: String Between Tests



‘duck’ lies between ‘Duck’ and ‘Duckling’ (exclusive, case)? false
‘duck’ lies between ‘Duck’ and ‘Duckling’ (exclusive, no case)? false
‘duck’ lies between ‘Duck’ and ‘Duckling’ (inclusive, case)? false
‘duck’ lies between ‘Duck’ and ‘Duckling’ (inclusive, no case)? true

2.4.1.2.6. Substring Example

Example 24 defines some commands that expand to text with a normal space and with a non-breakable space:

 24

```
\newcommand{\strA}{An apple}
\newcommand{\strB}{n~ap}
```

The `\DTLifSubString` command is used to test if the second argument is a substring of the first:

```
(First two arguments expanded) `\'strB'
\DTLifSubString{\strA}{\strB}{is substring}
{isn't substring}
of `\'strA'.

`app'
\DTLifSubString{An apple}{app}{is substring}
{isn't substring}
of `An apple'.

(Non-breakable space same as space) `n~a'
\DTLifSubString{An apple}{n~a}{is substring}
{isn't substring}
of `An apple'.

(Robust commands stripped) `app'
\DTLifSubString{An \MakeUppercase{a}pple}{app}
{is substring}{isn't substring}
of `An \MakeUppercase{a}pple'.

(Grouping stripped) `app'
\DTLifSubString{An {ap}ple}{app}{is substring}
{isn't substring}
of `An {ap}ple'.

(Case-sensitive) `app'
\DTLifSubString{An Apple}{app}{is substring}
{isn't substring}
of `An Apple'.

(Not case-sensitive) `app'
```



```
\DTLifSubString*{An Apple}{app}{is substring}
{isn't substring}
of `An Apple'.

(Leading space) ` app'
\DTLifSubString{Anapple}{ app}{is substring}
{isn't substring}
of `Anapple'.
```

↑ Example 24: Substring Tests



(First two arguments expanded) ‘n ap’ is substring of ‘An apple’.
 ‘app’ is substring of ‘An apple’.
 (Non-breakable space same as space) ‘n a’ is substring of ‘An apple’.
 (Robust commands stripped) ‘app’ is substring of ‘An Apple’.
 (Grouping stripped) ‘app’ is substring of ‘An apple’.
 (Case-sensitive) ‘app’ isn’t substring of ‘An Apple’.
 (Not case-sensitive) ‘app’ is substring of ‘An Apple’.
 (Leading space) ‘ app’ isn’t substring of ‘Anapple’.

2.4.1.2.7. String Prefix Example

Example 25 uses `\DTLifStartsWith` to test if the second argument is at the start (is a prefix) of the first:

25

```
\newcommand{\strA}{An apple}
\newcommand{\strB}{n~ap}
\newcommand{\strC}{An~ap}

(First two arguments expanded) ` \strB'
\DTLifStartsWith{\strA}{\strB}{is prefix}
{isn't prefix}
of ` \strA'.

(First two arguments expanded) ` \strC'
\DTLifStartsWith{\strA}{\strC}{is prefix}
{isn't prefix}
of ` \strA'.

(Non-breakable space same as space) `An~a'
```

```
\DTLifStartsWith{An apple}{An~a}{is prefix}
{isn't prefix}
of `An apple'.

(Robust commands stripped) `app'
\DTLifStartsWith{\MakeUppercase{a}pple}{app}
{is prefix}{isn't prefix}
of `\MakeUppercase{a}pple'.

(Case-sensitive) `app'
\DTLifStartsWith{Apple}{app}{is prefix}{isn't prefix}

of `Apple'.

(Ignore case) `app'
\DTLifStartsWith*{Apple}{app}{is prefix}
{isn't prefix}
of `Apple'.

(Trailing space) `an '
\DTLifStartsWith{an apple}{an }{is prefix}
{isn't prefix}
of `an apple'.

(Trailing space) `an '
\DTLifStartsWith{anapple}{an }{is prefix}
{isn't prefix}
of `anapple'.
```

↑ Example 25: Prefix Tests



(First two arguments expanded) ‘n ap’ isn’t prefix of ‘An apple’.
(First two arguments expanded) ‘An ap’ is prefix of ‘An apple’.
(Non-breakable space same as space) ‘An a’ is prefix of ‘An apple’.
(Robust commands stripped) ‘app’ isn’t prefix of ‘Apple’.
(Case-sensitive) ‘app’ isn’t prefix of ‘Apple’.
(Ignore case) ‘app’ is prefix of ‘Apple’.
(Trailing space) ‘an ’ is prefix of ‘an apple’.
(Trailing space) ‘an ’ isn’t prefix of ‘anapple’.

2.4.1.2.8. String Suffix Example

Example 26 uses `\DTLifEndsWith` to test if the second argument is at the end (is a suffix) of the first. It uses the same `\strA` and `\strB` as before:

26

```
\newcommand{\strA}{An apple}
\newcommand{\strB}{n~apple}
```

The tests are as follows:

```
(First two arguments expanded) ` \strB'
\DTLifEndsWith{\strA}{\strB}{is suffix}{isn't suffix}

of ` \strA'.

(Non-breakable space same as space) `n~apple'
\DTLifEndsWith{An apple}{n~apple}{is suffix}
{isn't suffix}
of `An apple'.

(Robust commands stripped) `apple'
\DTLifEndsWith{An \MakeUppercase{a}pple}{apple}
{is suffix}{isn't suffix}
of `An \MakeUppercase{a}pple'.

(Case-sensitive) `apple'
\DTLifEndsWith{An Apple}{apple}{is suffix}
{isn't suffix}
of `An Apple'.

(Ignore case) `apple'
\DTLifEndsWith*{An Apple}{apple}{is suffix}
{isn't suffix}
of `An Apple'.

(Leading space) ` apple'
\DTLifEndsWith{anapple}{ apple}{is suffix}
{isn't suffix}
of `anapple'.
```

 Example 26: Suffix Tests


(First two arguments expanded) ‘n apple’ is suffix of ‘An apple’.
 (Non-breakable space same as space) ‘n apple’ is suffix of ‘An apple’.
 (Robust commands stripped) ‘apple’ is suffix of ‘An Apple’.
 (Case-sensitive) ‘apple’ isn’t suffix of ‘An Apple’.
 (Ignore case) ‘apple’ is suffix of ‘An Apple’.
 (Leading space) ‘ apple’ is suffix of ‘an apple’.
 (Leading space) ‘ apple’ isn’t suffix of ‘anapple’.

2.4.1.2.9. String Case Example

Example 27 tests if the argument (once expanded and purified) is all the same case:

 27



```

café: \DTLifAllUpperCase{café}{all caps}
{not all caps}.

Café: \DTLifAllUpperCase{Café}{all caps}
{not all caps}.

CAFÉ: \DTLifAllUpperCase{CAFÉ}{all caps}
{not all caps}.

café: \DTLifAllLowerCase{café}{all lower}
{not all lower}.

Café: \DTLifAllLowerCase{Café}{all lower}
{not all lower}.

CAFÉ: \DTLifAllLowerCase{CAFÉ}{all lower}
{not all lower}.

bric-\`a-brac:
\DTLifAllLowerCase{bric-\`a-brac}{all lower}
{not all lower}.

\emph{HORS D'\OE UVRE}:
\DTLifAllUpperCase{\emph{HORS D'\OE UVRE}}{all caps}
{not all caps}.

```

↑ Example 27: All Upper/Lower Case Tests

café: not all caps.
 Café: not all caps.
 CAFÉ: all caps.
 café: all lower.
 Café: not all lower.
 CAFÉ: not all lower.
 bric-à-brac: all lower.
HORS D'ŒUVRE: all caps.

2.4.1.3. Formatted Number Conditionals

These commands expect formatted numbers or datum control sequences in the numerical arguments and compare their values. They internally use the corresponding command from §2.4.1.4 after parsing to perform the actual comparison.

`\DTLifnumeq{⟨num1⟩}{⟨num2⟩}{⟨true⟩}{⟨false⟩}`

Does *⟨true⟩* if *⟨num1⟩* equals *⟨num2⟩* (*⟨num1⟩ = ⟨num2⟩*) otherwise does *⟨false⟩*, where the values are formatted numbers. This command is robust. Internally uses `\dtlifnumeq` after parsing the values.

`\DTLifnumlt{⟨num1⟩}{⟨num2⟩}{⟨true⟩}{⟨false⟩}`

Does *⟨true⟩* if *⟨num1⟩* is less than *⟨num2⟩* (*⟨num1⟩ < ⟨num2⟩*) otherwise does *⟨false⟩*, where the values are formatted numbers. This command is robust. Internally uses `\dtlifnumlt` after parsing the values.

`\DTLifnumgt{⟨num1⟩}{⟨num2⟩}{⟨true⟩}{⟨false⟩}`

Does *⟨true⟩* if *⟨num1⟩* is greater than *⟨num2⟩* (*⟨num1⟩ > ⟨num2⟩*) otherwise does *⟨false⟩*, where the values are formatted numbers. This command is robust. Internally uses `\dtlifnumgt` after parsing the values.

`\DTLifnumopenbetween{⟨num⟩}{⟨min⟩}{⟨max⟩}{⟨true⟩}{⟨false⟩}`

Does *⟨true⟩* if *⟨num⟩* lies between *⟨min⟩* and *⟨max⟩*, excluding the end points (that is, *⟨min⟩ < ⟨num⟩ < ⟨max⟩*) otherwise does *⟨false⟩*, where the values are formatted numbers. This command is robust. Internally uses `\dtlifnumopenbetween` after parsing the values.



`\DTLifnumclosedbetween{⟨num⟩}{⟨min⟩}{⟨max⟩}{⟨true⟩}{⟨false⟩}`

Does $\langle true \rangle$ if $\langle num \rangle$ lies between $\langle min \rangle$ and $\langle max \rangle$, including the end points (that is, $\langle min \rangle \leq \langle num \rangle \leq \langle max \rangle$) otherwise does $\langle false \rangle$, where the values are formatted numbers. This command is robust. Internally uses `\dtlifnumclosedbetween` after parsing the values.

Note that the currency unit (if given) in the above comparisons is disregarded. Only the numeric value obtained from parsing is considered. Example 28 uses the default `math=13fp` setting.

28



```
$1,234.0=1234$? \DTLifnumeq{1,234.0}{1234}{true}
{false}.

$\$12.00=\pounds12$? \DTLifnumeq{\$12.00}{\pounds12}
{true}{false}.

$\$10.50<\pounds10$? \DTLifnumlt{\$10.50}{\pounds10}
{true}{false}.

$1,000.0 > 1,000$? \DTLifnumgt{1,000.0}{1,000}{true}
{false}.

$1000 < \$1,000.00 < 2000$?
\DTLifnumopenbetween{\$1,000.00}{1000}{2000}{true}
{false}.

$1000 \leq \$1,000.00 \leq 2000$?
\DTLifnumclosedbetween{\$1,000.00}{1000}{2000}{true}
{false}.
```



↑ Example 28: Numerical Comparisons (Parsed)



```
1,234.0 = 1234? true.
$12.00 = £12? true.
$10.50 < £10? false.
1,000.0 > 1,000? false.
1000 < $1,000.00 < 2000? false.
1000 ≤ $1,000.00 ≤ 2000? true.
```

2.4.1.4. Plain Number Conditionals

```
\dtlifnumeq{⟨num1⟩}{⟨num2⟩}{⟨true⟩}{⟨false⟩}
```

Does $\langle true \rangle$ if $\langle num1 \rangle$ equals $\langle num2 \rangle$ otherwise does $\langle false \rangle$. The numbers must be plain numbers. This command is expandable with $\mathsf{math=13fp}$ and $\mathsf{math=lua}$ and robust for $\mathsf{math=fp}$ and $\mathsf{math=pgfmath}$.

```
\dtlifnumlt{⟨num1⟩}{⟨num2⟩}{⟨true⟩}{⟨false⟩}
```

Does $\langle true \rangle$ if $\langle num1 \rangle$ is less than $\langle num2 \rangle$ otherwise does $\langle false \rangle$. The numbers must be plain numbers. This command is expandable with $\mathsf{math=13fp}$ and $\mathsf{math=lua}$ and robust for $\mathsf{math=fp}$ and $\mathsf{math=pgfmath}$.

```
\dtlifnumgt{⟨num1⟩}{⟨num2⟩}{⟨true⟩}{⟨false⟩}
```

Does $\langle true \rangle$ if $\langle num1 \rangle$ is greater than $\langle num2 \rangle$ otherwise does $\langle false \rangle$. The numbers must be plain numbers. This command is expandable with $\mathsf{math=13fp}$ and $\mathsf{math=lua}$ and robust for $\mathsf{math=fp}$ and $\mathsf{math=pgfmath}$.

```
\dtlifnumopenbetween{⟨num⟩}{⟨min⟩}{⟨max⟩}{⟨true⟩}{⟨false⟩}
```

Does $\langle true \rangle$ if $\langle num \rangle$ lies between $\langle min \rangle$ and $\langle max \rangle$, excluding the end points (that is, $\langle min \rangle < \langle num \rangle < \langle max \rangle$) otherwise does $\langle false \rangle$. The numbers must be plain numbers.

```
\DTLiFFPopenbetween{⟨num⟩}{⟨min⟩}{⟨max⟩}{⟨true⟩}{⟨false⟩}
```

Synonym of `\dtlifnumopenbetween`.

```
\dtlifintopenbetween{⟨num⟩}{⟨min⟩}{⟨max⟩}{⟨true⟩}{⟨false⟩}
```

As `\dtlifnumopenbetween` but specifically for integers. This simply uses `\ifnum` for the comparisons and is not dependent on the math option.

```
\dtlifnumclosedbetween{⟨num⟩}{⟨min⟩}{⟨max⟩}{⟨true⟩}{⟨false⟩}
```

Does $\langle true \rangle$ if $\langle num \rangle$ lies between $\langle min \rangle$ and $\langle max \rangle$, including the end points (that is, $\langle min \rangle \leq \langle num \rangle \leq \langle max \rangle$) otherwise does $\langle false \rangle$. The numbers must be plain numbers.



```
\DTLifFPclosedbetween{<num>}{<min>}{<min>}{<true>}{<false>}
```

Synonym of `\dtlifnumclosedbetween`.



```
\dtlifintclosedbetween{<num>}{<min>}{<min>}{<true>}{<false>}
```

As `\dtlifnumclosedbetween` but specifically for integers. This simply uses `\ifnum` for the comparisons and is not dependent on the `math` option.

2.4.1.4.1. Example (13fp)

Example 29 uses `\edef` (which defines a command with its provided definition expanded) and `\meaning` (which writes the command's definition to the PDF) to demonstrate commands that can expand. Compare the results with using `math=fp` (Example 31) and `math=pgfmath` (Example 32).

 29



```
\usepackage[math=13fp]{datatool-base}
\newcommand{\numducks}{4}
\begin{document}
\edef\test{There
  \dtlifnumeq{\numducks}{1}{is 1 duck}
{are \numducks\space ducks}.}
\texttt{\meaning\test}

Test text: \test

\edef\test{There are
  \dtlifnumlt{\numducks}{10}{less than}{not less than}

  10 ducks.}
\texttt{\meaning\test}

Test text: \test

\edef\test{There are
  \dtlifnumgt{\numducks}{10}{more than}{not more than}

  10 ducks.}
\texttt{\meaning\test}

Test text: \test
```



```
\edef\test{There
\dtlifnumopenbetween{\numducks}{4}{10}{are}{are not}

between 4 and 10 ducks (exclusive).}
\texttt{\meaning\test}

Test text: \test

\edef\test{There
\dtlifnumclosedbetween{\numducks}{4}{10}{are}
{are not}
between 4 and 10 ducks (inclusive).}
\texttt{\meaning\test}

Test text: \test
\end{document}
```

↑ Example 29: Conditionals (13fp)



```
macro:->There are 4 ducks.
Test text: There are 4 ducks.
macro:->There are less than 10 ducks.
Test text: There are less than 10 ducks.
macro:->There are not more than 10 ducks.
Test text: There are not more than 10 ducks.
macro:->There are not between 4 and 10 ducks (exclusive).
Test text: There are not between 4 and 10 ducks (exclusive).
macro:->There are between 4 and 10 ducks (inclusive).
Test text: There are between 4 and 10 ducks (inclusive).
```

2.4.1.4.2. Example (lua)

Example 30 is the same as Example 29 except that it uses `math=lua` (and so requires Lua³⁰LaTeX):

30

```
\usepackage[math=lua]{datatool-base}
```

2.4.1.4.3. Example (fp)

Example 31 is the same as Example 29 except that it uses `math=fp`:

31

↑ Example 30: Conditionals (lua)



```
macro:->There are 4 ducks.
Test text: There are 4 ducks.
macro:->There are less than 10 ducks.
Test text: There are less than 10 ducks.
macro:->There are not more than 10 ducks.
Test text: There are not more than 10 ducks.
macro:->There are not between 4 and 10 ducks (exclusive).
Test text: There are not between 4 and 10 ducks (exclusive).
macro:->There are between 4 and 10 ducks (inclusive).
Test text: There are between 4 and 10 ducks (inclusive).
```

```
\usepackage[math=fp]{datatool-base}
```

However, note that commands like `\dtlifnumeq` are now robust and so can't expand (but `\numducks` does expand).

↑ Example 31: Conditionals (fp)



```
macro:->There \dtlifnumeq {4}{1}{is 1 duck}{are 4 ducks}.
Test text: There are 4 ducks.
macro:->There are \dtlifnumlt {4}{10}{less than}{not less than}
10 ducks.
Test text: There are less than 10 ducks.
macro:->There are \dtlifnumgt {4}{10}{more than}{not more than}
10 ducks.
Test text: There are not more than 10 ducks.
macro:->There \dtlifnumopenbetween {4}{4}{10}{are}{are not} between
4 and 10 ducks (exclusive).
Test text: There are not between 4 and 10 ducks (exclusive).
macro:->There \dtlifnumclosedbetween {4}{4}{10}{are}{are not}
between 4 and 10 ducks (inclusive).
Test text: There are between 4 and 10 ducks (inclusive).
```

2.4.1.4.4. Example (*pgfmath*)

Example 32 is the same as for Example 29 except that it uses `math=pgfmath`:

32

```
\usepackage[math=pgfmath]{datatool-base}
```

However, note that commands like `\dtlifnumeq` are now robust and so can't expand (but `\numducks` does expand).

Example 32: Conditionals (*pgfmath*)

```
macro:->There \dtlifnumeq {4}{1}{is 1 duck}{are 4 ducks}.
Test text: There are 4 ducks.
macro:->There are \dtlifnumlt {4}{10}{less than}{not less than}
10 ducks.
Test text: There are less than 10 ducks.
macro:->There are \dtlifnumgt {4}{10}{more than}{not more than}
10 ducks.
Test text: There are not more than 10 ducks.
macro:->There \dtlifnumopenbetween {4}{4}{10}{are}{are not} between
4 and 10 ducks (exclusive).
Test text: There are not between 4 and 10 ducks (exclusive).
macro:->There \dtlifnumclosedbetween {4}{4}{10}{are}{are not}
between 4 and 10 ducks (inclusive).
Test text: There are between 4 and 10 ducks (inclusive).
```

2.4.1.5. String or Number Conditionals

The commands listed in this section parse the $\langle arg1 \rangle$ and $\langle arg2 \rangle$ arguments to determine whether to use the applicable string (§2.4.1.2) or numeric (§2.4.1.3) command. Those arguments may also be datum control sequences.

```
\DTLlifeq{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }{ $\langle true \rangle$ }{ $\langle false \rangle$ }
```

modifier: *

If $\langle arg1 \rangle$ and $\langle arg2 \rangle$ are both numeric (formatted numbers) then `\DTLifnumeq` is used otherwise `\DTLifstringeq` is used. The starred version is only applicable for string equality and will ignore the case. This command is robust.

```
\DTLiflt{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }{ $\langle true \rangle$ }{ $\langle false \rangle$ }
```

modifier: *

If $\langle arg1 \rangle$ and $\langle arg2 \rangle$ are both numeric (formatted numbers) then `\DTLifnumlt` is used otherwise `\DTLifstringlt` is used. The starred version is only applicable for a string comparison and will ignore the case. This command is robust.

`\DTLifgt{⟨arg1⟩}{⟨arg2⟩}{⟨true⟩}{⟨false⟩}`

modifier: *

If $\langle arg1 \rangle$ and $\langle arg2 \rangle$ are both numeric (formatted numbers) then `\DTLifnumgt` is used otherwise `\DTLifstringgt` is used. The starred version is only applicable for a string comparison and will ignore the case. This command is robust.

`\DTLifopenbetween{⟨value⟩}{⟨min⟩}{⟨min⟩}{⟨true⟩}{⟨false⟩}` modifier: *

If $\langle value \rangle$, $\langle min \rangle$ and $\langle max \rangle$ are all numeric (formatted numbers) then `\DTLifnumopenbetween` is used otherwise `\DTLifstringopenbetween` is used. The starred version is only applicable for a string comparison and will ignore the case. This command is robust.

`\DTLifclosedbetween{⟨value⟩}{⟨min⟩}{⟨min⟩}{⟨true⟩}{⟨false⟩}`

modifier: *

If $\langle value \rangle$, $\langle min \rangle$ and $\langle max \rangle$ are all numeric (formatted numbers) then `\DTLifnumclosedbetween` is used otherwise `\DTLifstringclosedbetween` is used. The starred version is only applicable for a string comparison and will ignore the case. This command is robust.

Example 33 uses the above conditional commands that determine from the arguments whether to use string or numeric comparisons:

33

```
1 = 1.0? (numeric) \DTLifeq{1}{1.0}{true}{false}.
1p = 1.0p? (string) \DTLifeq{1p}{1.0p}{true}{false}.
2 lt 10? (numeric) \DTLiflt{2}{10}{true}{false}.
A2 lt A10? (string) \DTLiflt{A2}{A10}{true}{false}.
2.0 gt 10.0? (numeric) \DTLifgt{2}{10}{true}{false}.
A2.0 gt A10.0? (string) \DTLifgt{A2.0}{A10.0}{true}{false}.

10 between 1 and 20 (numeric, exclusive)?
\DTLifopenbetween{10}{1}{20}{true}{false}.

10p between 1p and 20p (string, exclusive)?
\DTLifopenbetween{10p}{1p}{20p}{true}{false}.
```

```
1 between 1.0 and 2 (numeric, inclusive)?
\DTLifclosedbetween{1}{1.0}{2}{true}{false}.

1 between 1.0 and 2A (string, inclusive)?
\DTLifclosedbetween{1}{1.0}{2A}{true}{false}.
```

↑ Example 33: Numerical/String Comparisons



```
1 = 1.0? (numeric) true.
1p = 1.0p? (string) false.
2 lt 10? (numeric) true.
A2 lt A10? (string) false.
2.0 gt 10.0? (numeric) false.
A2.0 gt A10.0? (string) true.
10 between 1 and 20 (numeric, exclusive)? true.
10p between 1p and 20p (string, exclusive)? false.
1 between 1.0 and 2 (numeric, inclusive)? true.
1 between 1.0 and 2A (string, inclusive)? false.
```

2.4.2. ifthen conditionals

The commands described in §2.4.1 can not be used in the conditional part of the `\ifthenelse` or `\whiledo` commands provided by the `ifthen` package. This section describes analogous commands which may only be in the conditional part of the `\ifthenelse` or `\whiledo`. These may be used with the boolean operations `\not`, `\and` and `\or` provided by the `ifthen` package. See the `ifthen` documentation for further details.

```
texdoc ifthen
```



Be aware of protected expansion in the argument of commands like `\ifthenelse` that can cause a different result from using `\DTLis...` compared to the corresponding `\DTLif...` (see Example 34).



```
\DTLisint{<arg>}
```

As `\DTLifint` but for use in `ifthen` conditionals (see Example 2.4.2.1).

`\DTLisreal{⟨arg⟩}`

As `\DTLifreal` but for use in `ifthen` conditionals (see Example 2.4.2.1).

`\DTLiscurrency{⟨arg⟩}`

As `\DTLifcurrency` but for use in `ifthen` conditionals. Note that `\DTLfmtcurr`, `\DTLfmtcurrency` and `\DTLcurrency` are designed to expand so if you have data that contains those commands it's better to use `\DTLifcurrency` (see Example 2.4.2.1).

`\DTLiscurrencyunit{⟨arg⟩}`

As `\DTLifcurrencyunit` but for use in `ifthen` conditionals (see Example 2.4.2.1).

`\DTLisnumerical{⟨arg⟩}`

As `\DTLifnumerical` but for use in `ifthen` conditionals (see Example 2.4.2.1).

`\DTLisstring{⟨arg⟩}`

As `\DTLifstring` but for use in `ifthen` conditionals (see Example 2.4.2.1).

`\DTLiseq{⟨arg1⟩}{⟨arg2⟩}`

As the unstarred `\DTLifeq` but for use in `ifthen` conditionals (see Example 2.4.2.2).

`\DTLisieq{⟨arg1⟩}{⟨arg2⟩}`

As the starred `\DTLifeq*` but for use in `ifthen` conditionals (see Example 2.4.2.2).

`\DTLisnumeq{⟨arg1⟩}{⟨arg2⟩}`

As `\DTLifnumeq` but for use in `ifthen` conditionals.

`\DTLisFPeq{⟨arg1⟩}{⟨arg2⟩}`

Synonym of `\DTLisnumeq`.

`\DTLislt{⟨arg1⟩}{⟨arg2⟩}`

As the unstarred `\DTLiflt` but for use in `ifthen` conditionals (see Example 2.4.2.2).

`\DTLisilt{⟨arg1⟩}{⟨arg2⟩}`

As the starred `\DTLiflt*` but for use in `ifthen` conditionals (see Example 2.4.2.2).

`\DTLisnumlt{⟨arg1⟩}{⟨arg2⟩}`

As `\DTLifnumlt` but for use in `ifthen` conditionals.

`\DTLisFPlt{⟨arg1⟩}{⟨arg2⟩}`

Synonym of `\DTLisnumlt`.

`\DTLisnumlteq{⟨arg1⟩}{⟨arg2⟩}`

There isn't a `\DTLif...` direct equivalent of this command, except using `\DTLifnumgt` with the final two arguments flipped. Evaluates to true if $\langle arg1 \rangle \leq \langle arg2 \rangle$, where the arguments are formatted numbers.

`\DTLisFPlteq{⟨arg1⟩}{⟨arg2⟩}`

Synonym of `\DTLisnumlteq`.

`\DTLisgt{⟨arg1⟩}{⟨arg2⟩}`

As the unstarred `\DTLifgt` but for use in `ifthen` conditionals (see Example 2.4.2.2).

`\DTLisigt{⟨arg1⟩}{⟨arg2⟩}`

As the starred `\DTLifgt*` but for use in `ifthen` conditionals (see Example 2.4.2.2).

`\DTLisnumgt{⟨arg1⟩}{⟨arg2⟩}`

As `\DTLifnumgt` but for use in `ifthen` conditionals.

`\DTLisFPgt{⟨arg1⟩}{⟨arg2⟩}`

Synonym of `\DTLisnumgt`.

`\DTLisnumgteq{⟨arg1⟩}{⟨arg2⟩}`

There isn't a `\DTLif...` direct equivalent of this command, except using `\DTLifnumlt` with the final two arguments flipped. Evaluates to true if $\langle arg1 \rangle \geq \langle arg2 \rangle$, where the arguments are formatted numbers.

`\DTLisFPgteq{⟨arg1⟩}{⟨arg2⟩}`

Synonym of `\DTLisnumgteq`.

`\DTLisopenbetween{⟨num⟩}{⟨min⟩}{⟨min⟩}`

As the unstarred `\DTLifopenbetween` but for use in `ifthen` conditionals.

`\DTLisiopenbetween{⟨num⟩}{⟨min⟩}{⟨min⟩}`

As the starred `\DTLifopenbetween*` but for use in `ifthen` conditionals.

`\DTLisnumopenbetween{⟨num⟩}{⟨min⟩}{⟨min⟩}`

As `\DTLifnumopenbetween` but for use in `ifthen` conditionals.

`\DTLisFPopenbetween{⟨num⟩}{⟨min⟩}{⟨min⟩}`

Synonym of `\DTLisnumopenbetween`.

`\DTLisclosedbetween{⟨num⟩}{⟨min⟩}{⟨min⟩}`

As the unstarred `\DTLifclosedbetween` but for use in `ifthen` conditionals.

`\DTLisiclosedbetween{⟨num⟩}{⟨min⟩}{⟨min⟩}`

As the starred `\DTLifclosedbetween*` but for use in `ifthen` conditionals.

`\DTLisnumclosedbetween{<num>}{<min>}{<min>}`

As `\DTLifnumclosedbetween` but for use in ifthen conditionals.

`\DTLisFPclosedbetween{<num>}{<min>}{<min>}`

Synonym of `\DTLisnumclosedbetween`.

`\DTLisinlist{<element>}{<list>}`

As `\DTLifinlist` but for use in ifthen conditionals (see Example 36).

`\DTLisSubString{<string>}{<fragment>}`

As the unstarred `\DTLifSubString` but for use in ifthen conditionals (see Example 36).

`\DTLisiSubString{<string>}{<fragment>}`

As the starred `\DTLifSubString*` but for use in ifthen conditionals (see Example 36).

`\DTLisPrefix{<string>}{<fragment>}`

As the unstarred `\DTLifStartsWith` but for use in ifthen conditionals (see Example 36).

`\DTLisiPrefix{<string>}{<fragment>}`

As the starred `\DTLifStartsWith*` but for use in ifthen conditionals (see Example 36).

`\DTLisSuffix{<string>}{<fragment>}`

As the unstarred `\DTLifEndsWith` but for use in ifthen conditionals (see Example 36).

`\DTLisiSuffix{<string>}{<fragment>}`

As the starred `\DTLifEndsWith*` but for use in ifthen conditionals (see Example 36).

2.4.2.1. Data Type Conditionals Example

Example 34 tests for the data type of the given argument, which will be parsed according to the current locale settings.

34

```
1,234.0: \ifthenelse{\DTLisint{1,234.0}}{int}
{not int}.

1,234: \ifthenelse{\DTLisint{1,234}}{int}{not int}.

1,234.0: \ifthenelse{\DTLisreal{1,234.0}}{real}
{not real}.

1,234: \ifthenelse{\DTLisreal{1,234}}{real}{not real}
.

Compare:
\$1,234: \DTLifcurrency{\$1,234}{currency}
{not currency}.
With: \$1,234: \ifthenelse{\DTLiscurrency{\$1,234}}
{currency}{not currency}.

\DTLnewcurrencysymbol{\protect\$}%
\$1,234: \ifthenelse{\DTLiscurrency{\$1,234}}
{currency}{not currency}.

1.234,0: \ifthenelse{\DTLisnumerical{1.234,0}}
{numerical}{not numerical};
\ifthenelse{\DTLisstring{1.234,0}}{string}
{not string}.

\DTLsetnumberchars{.}{,}%
1.234,0: \ifthenelse{\DTLisnumerical{1.234,0}}
{numerical}{not numerical};
\ifthenelse{\DTLisstring{1.234,0}}{string}
{not string}.

Empty: \ifthenelse{\DTLisnumerical{}}{numerical}
{not numerical};
\ifthenelse{\DTLisstring{}}{string}{not string}.
```

Note the difference between `\DTLifcurrency` and `\DTLiscurrency`. This is because `\ifthenelse` causes `\$` to expand to `\protect\$`, which isn't recognised as a currency

unit by default.

↑ Example 34: Data Type Conditionals for use with `ifthen`



```
1,234.0: not int.
1,234: int.
1,234.0: real.
1,234: not real.
Compare: $1,234: currency. With: $1,234: not currency.
$1,234: currency.
1.234,0: not numerical; string.
1.234,0: numerical; not string.
Empty: not numerical; not string.
```

2.4.2.2. Order Conditionals Example

Example 35 demonstrates the order conditionals in `\ifthenelse`:

35

```
$1 = 1.0$?
\ifthenelse{\DTLisseq{1}{1.0}}{true}{false}.

duck = Duck? (case-sensitive)
\ifthenelse{\DTLisseq{duck}{Duck}}{true}{false}.

duck = Duck? (ignore case)
\ifthenelse{\DTLisieq{duck}{Duck}}{true}{false}.

$2 < 10$? \ifthenelse{\DTLislt{2}{10}}{true}{false}.

a before Z? (case-sensitive)
\ifthenelse{\DTLislt{a}{Z}}{true}{false}.

a before Z? (ignore case)
\ifthenelse{\DTLisilt{2}{10}}{true}{false}.

$1.5 > 1$?
\ifthenelse{\DTLisgt{1.5}{1}}{true}{false}.

a after Z? (case-sensitive)
\ifthenelse{\DTLisgt{a}{Z}}{true}{false}.
```

```
a after Z? (ignore case)
\ifthenelse{\DTLisigt{2}{10}}{true}{false}.
```

↑ Example 35: Order Conditionals for use with ifthen



```
1 = 1.0? true.
duck = Duck? (case-sensitive) false.
duck = Duck? (ignore case) true.
2 < 10? true.
a before Z? (case-sensitive) false.
a before Z? (ignore case) true.
1.5 > 1? true.
a after Z? (case-sensitive) true.
a after Z? (ignore case) false.
```

2.4.2.3. List Element and Substring Conditionals Example

Example 36 uses the list element conditional and substring conditionals:

36

```
`goose' element of list `ant,duck,goose'?
\ifthenelse{\DTLisinlist{goose}{ant,duck,goose}}
{true}{false}.

`oo' element of list `ant,duck,goose'?
\ifthenelse{\DTLisinlist{oo}{ant,duck,goose}}{true}
{false}.

`oo' in `goose'?
\ifthenelse{\DTLisSubString{goose}{oo}}{true}{false}
.

`oo' in `GOOSE' (case-sensitive)?
\ifthenelse{\DTLisSubString{GOOSE}{oo}}{true}{false}
.

`oo' in `GOOSE' (ignore case)?
\ifthenelse{\DTLisiSubString{GOOSE}{oo}}{true}{false}
.
```

```

`go' prefix of `goose'?
\ifthenelse{\DTLisPrefix{goose}{go}}{true}{false}.

`go' prefix of `GOOSE' (case-sensitive)?
\ifthenelse{\DTLisPrefix{GOOSE}{go}}{true}{false}.

`go' prefix of `GOOSE' (ignore case)?
\ifthenelse{\DTLisiPrefix{GOOSE}{go}}{true}{false}.

`se' suffix of `goose'?
\ifthenelse{\DTLisSuffix{goose}{se}}{true}{false}.

`se' suffix of `GOOSE' (case-sensitive)?
\ifthenelse{\DTLisSuffix{GOOSE}{se}}{true}{false}.

`se' suffix of `GOOSE' (ignore case)?
\ifthenelse{\DTLisiSuffix{GOOSE}{se}}{true}{false}.

```



↑ Example 36: Substring Conditionals for use with ifthen



```

'goose' element of list 'ant,duck,goose'? true.
'oo' element of list 'ant,duck,goose'? false.
'oo' in 'goose'? true.
'oo' in 'GOOSE' (case-sensitive)? false.
'oo' in 'GOOSE' (ignore case)? true.
'go' prefix of 'goose'? true.
'go' prefix of 'GOOSE' (case-sensitive)? false.
'go' prefix of 'GOOSE' (ignore case)? true.
'se' suffix of 'goose'? true.
'se' suffix of 'GOOSE' (case-sensitive)? false.
'se' suffix of 'GOOSE' (ignore case)? true.

```

2.5. Decimal Functions

Commands with a name prefixed with “`dtl`” (such as `\dtladd`) that are described in §2.5.1 don’t parse for the current decimal character and number group character or for a currency symbol. They require a plain number, either a bare integer (such as 12345) or a number with a decimal point (such as 1234.5). The definition of these commands depends on the value of the `math` package option.

Commands with a name prefixed with “`DTL`” (such as `\DTLadd`) that are described in §2.5.2 expect formatted numbers in the supplied values. These commands are provided by

datatool-base and use `\DTLconverttodecimal` to convert the supplied values to plain numbers.

2.5.1. Plain Numbers



If you have complex calculations, you may prefer to use $\text{\LaTeX}$ commands directly, as shown in Example 3. Alternatively, if you are using $\text{\Lua\LaTeX}$, you may prefer to use `\directlua`, as shown in Example 4.

Commands with a CSV list argument, such as `\dtladdall`, will do at least one expansion. The `math=l3fp` and `math=lua` options will fully expand $\langle num\ list \rangle$, but the `math=fp` and `math=pgfmath` options will only do a single expansion. This is different to most CSV list arguments provided by *datatool-base* (see §2.9). Since the list is expected to only contain comma-separated plain numbers there should be no expansion issues. Avoid empty elements.



```
\dtlpadleadingzeros{ $\langle num-digits \rangle$ }{ $\langle value \rangle$ }
```

Expands to a plain number that is the supplied $\langle value \rangle$ padded with leading zeros to the number of digits identified in the $\langle num-digits \rangle$ argument. Both arguments must be plain numbers. The $\langle num-digits \rangle$ argument should lie between 1 and 7. No error will occur if $\langle num-digits \rangle$ is outside that range. This command is primarily designed for sorting where the numbers are mixed with strings where a character code comparison will be used, and so is expandable. Unlike `\two@digits`, the $\langle value \rangle$ may be a decimal.



```
\dtlpadleadingzerosminus
```

initial: -

This will be inserted by `\dtlpadleadingzeros` if the value is negative.



```
\dtlpadleadingzerosplus
```

initial: empty

This will be inserted by `\dtlpadleadingzeros` if the value is positive. Note that this expands to nothing by default. This is because the plus (+) character has a lower character code than the hyphen-minus (-) character, which would put positive numbers before negative numbers in a character code sort.



```
\dtladd{ $\langle cs \rangle$ }{ $\langle num1 \rangle$ }{ $\langle num2 \rangle$ }
```

Calculates $\langle num1 \rangle + \langle num2 \rangle$ (addition) and stores the result in the control sequence $\langle cs \rangle$, where the numbers are plain numbers.

`\dtladdall{<cs>}{<num list>}`

Adds all the numbers in the comma-separated list $\langle num\ list \rangle$ and stores the result in the control sequence $\langle cs \rangle$, where the numbers are plain numbers.

The number list should not contain empty elements.

`\dtlsub{<cs>}{<num1>}{<num2>}`

Calculates $\langle num1 \rangle - \langle num2 \rangle$ (subtraction) and stores the result in the control sequence $\langle cs \rangle$, where the numbers are plain numbers.

`\dtlmul{<cs>}{<num1>}{<num2>}`

Calculates $\langle num1 \rangle \times \langle num2 \rangle$ (multiplication) and stores the result in the control sequence $\langle cs \rangle$, where the numbers are plain numbers.

`\dtldiv{<cs>}{<num1>}{<num2>}`

Calculates $\langle num1 \rangle \div \langle num2 \rangle$ (division) and stores the result in the control sequence $\langle cs \rangle$, where the numbers are plain numbers.

`\dtlsqrt{<cs>}{<num>}`

Calculates the square root of $\langle num \rangle$ and stores the result in the control sequence $\langle cs \rangle$, where the number is a plain number.

`\dtlroot{<cs>}{<num>}{<n>}`

Calculates the $\langle n \rangle$ th root of $\langle num \rangle$ and stores the result in the control sequence $\langle cs \rangle$, where the number is a plain number.

`\dtlround{<cs>}{<num>}{<dp>}`

Rounds $\langle num \rangle$ to $\langle dp \rangle$ decimal places and stores the result in the control sequence $\langle cs \rangle$, where the number is a plain number.

`\dtltrunc{<cs>}{<num>}{<dp>}`

Truncates $\langle num \rangle$ to $\langle dp \rangle$ decimal places and stores the result in the control sequence $\langle cs \rangle$, where the number is a plain number.

`\dtlclip{<cs>}{<num>}`

Removes redundant trailing zeros from $\langle num \rangle$ and stores the result in the control sequence $\langle cs \rangle$, where the number is a plain number.

`\dtlmin{<cs>}{<num1>}{<num2>}`

Defines the control sequence $\langle cs \rangle$ to the smaller of the two numbers, where the numbers are plain numbers.

`\dtlminall{<cs>}{<num list>}`

Defines the control sequence $\langle cs \rangle$ to the minimum value in the given comma-separated list $\langle num-list \rangle$ of numbers, where the numbers are plain numbers.

The number list should not contain empty elements.

`\dtlmax{<cs>}{<num1>}{<num2>}`

Defines the control sequence $\langle cs \rangle$ to the larger of the two numbers, where the numbers are plain numbers.

`\dtlmaxall{<cs>}{<num list>}`

Defines the control sequence $\langle cs \rangle$ to the maximum value in the given comma-separated list $\langle num-list \rangle$ of numbers, where the numbers are plain numbers.

The number list should not contain empty elements.

`\dtlabs{<cs>}{<num>}`

2. Base Commands (*datatool*–base package)

Defines the control sequence $\langle cs \rangle$ to the absolute value of the number $\langle num \rangle$, where the number is a plain numbers.

```
\dtlneg{ $\langle cs \rangle$ }{ $\langle num \rangle$ }
```

Defines the control sequence $\langle cs \rangle$ to the negative of the number $\langle num \rangle$, where the number is a plain numbers.

```
\dtlmeanforall{ $\langle cs \rangle$ }{ $\langle num list \rangle$ }
```

Calculates the mean (average) of all the numbers in the comma-separated list $\langle num list \rangle$ and stores the result in the control sequence $\langle cs \rangle$, where the numbers are plain numbers.

The number list should not contain empty elements.

```
\dtlvarianceforall[ $\langle mean \rangle$ ]{ $\langle cs \rangle$ }{ $\langle num list \rangle$ }
```

Calculates the variance of all the numbers in the comma-separated list $\langle num list \rangle$ and stores the result in the control sequence $\langle cs \rangle$, where the numbers are plain numbers. If the mean has already been calculated, it can be supplied in the optional argument $\langle mean \rangle$. If omitted, the mean will be calculated before calculating the variance.

The number list should not contain empty elements.

```
\dtlsdforall[ $\langle mean \rangle$ ]{ $\langle cs \rangle$ }{ $\langle num list \rangle$ }
```

Calculates the standard deviation of all the numbers in the comma-separated list $\langle num list \rangle$ and stores the result in the control sequence $\langle cs \rangle$, where the numbers are plain numbers. If the mean has already been calculated, it can be supplied in the optional argument $\langle mean \rangle$. If omitted, the mean will be calculated before calculating the standard deviation. If you have already calculated the variance you can simply use `\dtlsqrt`.

The number list should not contain empty elements.

2.5.1.1. Example (13fp)

Example 37 explicitly sets the processor to 13fp, which uses L^AT_EX3 floating point commands. This is now the default setting unless LuaL^AT_EX is used.  37

```
\documentclass{article}
\usepackage[math=13fp]{datatool-base}
\newcommand{\numA}{1023.5}
\newcommand{\numB}{54.75000}
\newcommand{\numC}{-20648.68}
\newcommand{\numlist}{32.456,0.15,-25,48.7,92}
\begin{document}
\dtladd{\result}{\numA}{\numB}
$\numA + \numB = \result$.

\dtladd{\result}{\result}{\numC}
Add $\numC$ to previous result.
Updated result: \result.

\dtladdall{\result}{\numlist}
Sum of all numbers in the set $\{\numlist\}$: \result.

\dtlsub{\result}{\numA}{\numB}
$\numA - \numB = \result$.

\dtlsub{\result}{\result}{\numC}
Subtract $\numC$ from previous result.
Updated result: \result.

\dtlmul{\result}{\numA}{\numB}
$\numA \times \numB = \result$.

\dtlmul{\result}{\result}{\numC}
Multiply previous result by $\numC$.
Updated result: \result.

\dtldiv{\result}{\numA}{\numB}
$\numA \div \numB = \result$.

\dtldiv{\result}{\result}{\numC}
Divide previous result by $\numC$.
Updated result: \result.
```

```
\dtlsqrt{\result}{\numA}
 $\sqrt{\numA} = \result$ .

\dtlsqrt{\result}{9}
 $\sqrt{9} = \result$ .

\dtlroot{\result}{\numA}{3}
 $\sqrt[3]{\numA} = \result$ .

\dtlroot{\result}{8}{3}
 $\sqrt[3]{8} = \result$ .

\dtlround{\result}{\numB}{1}
Round  $\numB$  to 1dp:  $\result$ .

\dtltrunc{\result}{\numB}{1}
Truncate  $\numB$  to 1dp:  $\result$ .

\dtlclip{\result}{\numB}
Clip  $\numB$ :  $\result$ .

\dtlmin{\result}{\numA}{\numB}
Minimum of  $\numA$  and  $\numB$ :  $\result$ .

\dtlminall{\result}{\numlist}
Minimum value in the set  $\{\numlist\}$ :  $\result$ .

\dtlmax{\result}{\numA}{\numB}
Maximum of  $\numA$  and  $\numB$ :  $\result$ .

\dtlmaxall{\result}{\numlist}
Maximum value in the set  $\{\numlist\}$ :  $\result$ .

\dtlabs{\result}{\numC}
Absolute value of  $\numC$ :  $\result$ .

\dtlneg{\result}{\numC}
Negate value of  $\numC$ :  $\result$ .

\dtlmeanforall{\meanvalue}{\numlist}
Mean of all numbers in the set  $\{\numlist\}$ :
 $\meanvalue$ .
```

```

\dtlvarianceforall[\meanvalue]{\result}{\numlist}
Variance of all numbers in the set $\{\numlist\}$
(using previously calculated mean): \result.

\dtlvarianceforall{\result}{\numlist}
Variance of all numbers in the set $\{\numlist\}$
(not using previously calculated mean): \result.

\dtlstdforall[\meanvalue]{\result}{\numlist}
Standard deviation of all numbers in the set
$\{\numlist\}$
(using previously calculated mean): \result.

\dtlstdforall{\result}{\numlist}
Standard deviation of all numbers in the set
$\{\numlist\}$
(not using previously calculated mean): \result.
\end{document}

```

2.5.1.2. Example (lua)

Example 38 uses the lua processor, which uses `\directlua` to perform the calculations, and so requires Lua_{TeX}. The only difference to Example 37 is the package option:

38

```
\usepackage[math=lua]{datatool-base}
```

(and the need to use Lua_{TeX}).

Note that this produces slightly different results from examples 37 & 39. For the division $1023.5 \div 54.75000$, `math=lua` produces 18.694063926941 whereas `math=l3fp` produces the result 18.69406392694064. This is due to rounding when the result from Lua is input into the $\TeX$ stream. With `math=fp` the result is 18.694063926940639269, which has even more significant digits. On the other hand, for the square root $\sqrt{9}$ and cubic root $\sqrt[3]{8}$, `math=l3fp` produces integers 3 and 2, `math=lua` returns equivalent decimals 3.0 and 2.0 but `math=fp` has rounding errors.

2.5.1.3. Example (fp)

Example 39 is almost identical to Example 37 but uses the `fp` processor, which uses the commands provided by the `fp` package. Note that the results have trailing redundant zeros and there are rounding errors for $\sqrt{9}$ and $\sqrt[3]{8}$.

39

↑ Example 37: Decimal Functions (13 fp)



$$1023.5 + 54.75000 = 1078.25.$$

Add -20648.68 to previous result. Updated result: -19570.43 .

Sum of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$: 148.306 .

$$1023.5 - 54.75000 = 968.75.$$

Subtract -20648.68 from previous result. Updated result: 21617.43 .

$$1023.5 \times 54.75000 = 56036.625.$$

Multiply previous result by -20648.68 . Updated result: -1157082337.905 .

$$1023.5 \div 54.75000 = 18.69406392694064.$$

Divide previous result by -20648.68 . Updated result: -0.0009053394176741874 .

$$\sqrt{1023.5} = 31.99218654609278.$$

$$\sqrt{9} = 3.$$

$$\sqrt[3]{1023.5} = 10.07772760987407.$$

$$\sqrt[3]{8} = 2.$$

Round 54.75000 to 1dp: 54.8 .

Truncate 54.75000 to 1dp: 54.7 .

Clip 54.75000 : 54.75 .

Minimum of 1023.5 and 54.75000 : 54.75 .

Minimum value in the set $\{32.456, 0.15, -25, 48.7, 92\}$: -25 .

Maximum of 1023.5 and 54.75000 : 1023.5 .

Maximum value in the set $\{32.456, 0.15, -25, 48.7, 92\}$: 92 .

Absolute value of -20648.68 : 20648.68 .

Negate value of -20648.68 : 20648.68 .

Mean of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$: 29.6612 .

Variance of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$ (using previously calculated mean): 1623.03410176 .

Variance of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$ (not using previously calculated mean): 1623.03410176 .

Standard deviation of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$ (using previously calculated mean): 40.28689739555529 .

Standard deviation of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$ (not using previously calculated mean): 40.28689739555529 .

↑ Example 38: Decimal Functions (lua)



$1023.5 + 54.75000 = 1078.25$.

Add -20648.68 to previous result. Updated result: -19570.43 .

Sum of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$: 148.306 .

$1023.5 - 54.75000 = 968.75$.

Subtract -20648.68 from previous result. Updated result: 21617.43 .

$1023.5 \times 54.75000 = 56036.625$.

Multiply previous result by -20648.68 . Updated result: -1157082337.905 .

$1023.5 \div 54.75000 = 18.694063926941$.

Divide previous result by -20648.68 . Updated result: -0.0009053394176742 .

$\sqrt{1023.5} = 31.992186546093$.

$\sqrt{9} = 3.0$.

$\sqrt[3]{1023.5} = 10.077727609874$.

$\sqrt[3]{8} = 2.0$.

Round 54.75000 to 1dp: 54.8 .

Truncate 54.75000 to 1dp: 54.7 .

Clip 54.75000 : 54.75 .

Minimum of 1023.5 and 54.75000 : 54.75 .

Minimum value in the set $\{32.456, 0.15, -25, 48.7, 92\}$: -25 .

Maximum of 1023.5 and 54.75000 : 1023.5 .

Maximum value in the set $\{32.456, 0.15, -25, 48.7, 92\}$: 92 .

Absolute value of -20648.68 : 20648.68 .

Negate value of -20648.68 : 20648.68 .

Mean of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$: 29.6612 .

Variance of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$ (using previously calculated mean): 1623.03410176 .

Variance of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$ (not using previously calculated mean): 1623.03410176 .

Standard deviation of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$ (using previously calculated mean): 40.286897395555 .

Standard deviation of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$ (not using previously calculated mean): 40.286897395555 .

```
\usepackage[math=fp]{datatool-base}
```

⌈ Example 39: Decimal Functions (fp)

$$1023.5 + 54.75000 = 1078.250000000000000000.$$

Add -20648.68 to previous result. Updated result: $-19570.43000000000000000000$.

Sum of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$: 148.306000000000000000.

$$1023.5 - 54.75000 = 968.750000000000000000.$$

Subtract -20648.68 from previous result. Updated result: 21617.43000000000000000000.

$$1023.5 \times 54.75000 = 56036.625000000000000000.$$

Multiply previous result by -20648.68 . Updated result: $-1157082337.905000000000000000$.

$$1023.5 \div 54.75000 = 18.694063926940639269.$$

Divide previous result by -20648.68 . Updated result: -0.000905339417674187 .

$$\sqrt{1023.5} = 31.992186546092781616.$$
$$\sqrt{9} = 2.99999999999999999972.$$
$$\sqrt[3]{1023.5} = 10.077727609874069159.$$
$$\sqrt[3]{8} = 1.999999999999999999990.$$

Round 54.75000 to 1dp: 54.8.

Truncate 54.75000 to 1dp: 54.7.

Clip 54.75000: 54.75.

Minimum of 1023.5 and 54.75000: 54.75000.

Minimum value in the set $\{32.456, 0.15, -25, 48.7, 92\}$: -25.

Maximum of 1023.5 and 54.75000: 1023.5.

Maximum value in the set $\{32.456, 0.15, -25, 48.7, 92\}$: 92.

Absolute value of -20648.68 : $20648.68000000000000000000$.

Negate value of -20648.68: 20648.680000000000000000.

Mean of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$: 29.661200000000000000.

Variance of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$ (using previously calculated mean): 1623.03410176000000000000.

Variance of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$ (not using previously calculated mean): 1623.03410176000000000000.

Standard deviation of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$ (using previously calculated mean): 40.286897395555294372.

Standard deviation of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$
(not using previously calculated mean): 40.286897395555294372.

2.5.1.4. Example (pgfmath)

If Example 37 is modified to use the `pgfmath` processor, which uses the commands provided by the `pgfmath` package, then the $\LaTeX$ run will fail with the error:



! Dimension too large

Example 40 has the commands `\numA`, `\numB` and `\numC` defined to smaller numbers. The rest of the document is as Example 37.

 40



```
\usepackage[math=pgfmath]{datatool-base}
\newcommand{\numA}{10.235}
\newcommand{\numB}{0.5475000}
\newcommand{\numC}{-206.4868}
```

Note that there are rounding errors.

2.5.2. Formatted Numbers

The commands listed in this section expect formatted numbers in the values according to the current number group character and decimal character settings. Use `\DTLsetnumberchars` to set these first. In general, if calculations are required, it's better to store the values as plain numbers if possible and only format them (for example, using `siunitx`) when they need to be typeset. That way the formatted values don't need to be repeatedly parsed.



Commands that have a `<num list>` argument, such as `\DTLaddall`, expect a CSV list or a command with a CSV list definition (see §2.9). The argument isn't fully expanded to allow for non-robust currency symbols. Any elements that aren't numeric will be treated as zero.



```
\DTLadd{<cs>}{<num1>}{<num2>}
```

Converts the formatted numbers `<num1>` and `<num2>` to plain numbers and adds them together (`<num1> + <num2>`). If parsing determines that both `<num1>` and `<num2>` are integers, integer arithmetic is performed with `\numexpr` otherwise `\dtladd` is used. The result is stored as a formatted number in the command `<cs>`.



```
\DTLgadd{<cs>}{<num1>}{<num2>}
```

As `\DTLadd` but globally sets `<cs>`.



```
\DTLaddall{<cs>}{<num list>}
```


↑ Example 40: Decimal Functions (*pgfmath*)



$10.235 + 0.5475000 = 10.7825$.

Add -206.4868 to previous result. Updated result: -195.7043 .

Sum of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$: 148.30598 .

$10.235 - 0.5475000 = 9.6875$.

Subtract -206.4868 from previous result. Updated result: 216.1743 .

$10.235 \times 0.5475000 = 5.60367$.

Multiply previous result by -206.4868 . Updated result: -1157.08351 .

$10.235 \div 0.5475000 = 18.69524$.

Divide previous result by -206.4868 . Updated result: -0.09055 .

$\sqrt{10.235} = 3.19921$.

$\sqrt{9} = 3.00000$.

$\sqrt[3]{10.235} = 2.17104$.

$\sqrt[3]{8} = 1.9999$.

Round 0.5475000 to 1dp: 0.5 .

Truncate 0.5475000 to 1dp: 0.5 .

Clip 0.5475000 : 0.5475 .

Minimum of 10.235 and 0.5475000 : 0.5475 .

Minimum value in the set $\{32.456, 0.15, -25, 48.7, 92\}$: -25.0 .

Maximum of 10.235 and 0.5475000 : 10.235 .

Maximum value in the set $\{32.456, 0.15, -25, 48.7, 92\}$: 92.0 .

Absolute value of -206.4868 : 206.4868 .

Negate value of -206.4868 : 206.4868 .

Mean of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$: 29.6612 .

Variance of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$ (using previously calculated mean): 1623.03413 .

Variance of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$ (not using previously calculated mean): 1623.03413 .

Standard deviation of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$ (using previously calculated mean): 40.28689 .

Standard deviation of all numbers in the set $\{32.456, 0.15, -25, 48.7, 92\}$ (not using previously calculated mean): 40.28689 .

2. Base Commands (*datatool-base package*)

Converts all the formatted numbers in the comma-separated list to plain numbers, adds them all, and stores the result as a formatted number in the command $\langle cs \rangle$.

```
\DTLgaddall{ $\langle cs \rangle$ }{ $\langle num\ list \rangle$ }
```

As `\DTLaddall` but globally sets $\langle cs \rangle$.

```
\DTLsub{ $\langle cs \rangle$ }{ $\langle num1 \rangle$ }{ $\langle num2 \rangle$ }
```

Converts the formatted numbers $\langle num1 \rangle$ and $\langle num2 \rangle$ to plain numbers and subtracts $\langle num2 \rangle$ from $\langle num1 \rangle$ ($\langle num1 \rangle - \langle num2 \rangle$). If parsing determines that both $\langle num1 \rangle$ and $\langle num2 \rangle$ are integers, integer arithmetic is performed with `\numexpr` otherwise `\dtlsub` is used. The result is stored as a formatted number in the command $\langle cs \rangle$.

```
\DTLgsub{ $\langle cs \rangle$ }{ $\langle num1 \rangle$ }{ $\langle num2 \rangle$ }
```

As `\DTLsub` but globally sets $\langle cs \rangle$.

```
\DTLmul{ $\langle cs \rangle$ }{ $\langle num1 \rangle$ }{ $\langle num2 \rangle$ }
```

Converts the formatted numbers $\langle num1 \rangle$ and $\langle num2 \rangle$ to plain numbers and multiplies them ($\langle num1 \rangle \times \langle num2 \rangle$). If parsing determines that both $\langle num1 \rangle$ and $\langle num2 \rangle$ are integers, integer arithmetic is performed with `\numexpr` otherwise `\dtlmul` is used. The result is stored as a formatted number in the command $\langle cs \rangle$.

```
\DTLgmul{ $\langle cs \rangle$ }{ $\langle num1 \rangle$ }{ $\langle num2 \rangle$ }
```

As `\DTLmul` but globally sets $\langle cs \rangle$.

```
\DTLdiv{ $\langle cs \rangle$ }{ $\langle num1 \rangle$ }{ $\langle num2 \rangle$ }
```

Converts the formatted numbers $\langle num1 \rangle$ and $\langle num2 \rangle$ to plain numbers and divides them ($\langle num1 \rangle \div \langle num2 \rangle$) using `\dtldiv`. The result is stored as a formatted number in the command $\langle cs \rangle$.

```
\DTLgdiv{ $\langle cs \rangle$ }{ $\langle num1 \rangle$ }{ $\langle num2 \rangle$ }
```

As `\DTLdiv` but globally sets $\langle cs \rangle$.

```
\DTLabs{ $\langle cs \rangle$ }{ $\langle num \rangle$ }
```

2. Base Commands (*datatool-base package*)

Converts the formatted numbers $\langle num \rangle$ to a plain number and stores the absolute value as a formatted number in the command $\langle cs \rangle$. If parsing determines that $\langle num \rangle$ is an integer then $\backslash ifnum$ and $\backslash numexpr$ are used to negate the number if it's negative. If $\langle num \rangle$ is determined to be a decimal or currency, then $\backslash dtlabs$ is used.

$\backslash DTLgabs\{\langle cs \rangle\}\{\langle num \rangle\}$

As $\backslash DTLabs$ but globally sets $\langle cs \rangle$.

$\backslash DTLneg\{\langle cs \rangle\}\{\langle num \rangle\}$

Converts the formatted numbers $\langle num \rangle$ to a plain number and stores the negation ($-\langle num \rangle$) as a formatted number in the command $\langle cs \rangle$. If parsing determines that $\langle num \rangle$ is an integer then $\backslash numexpr$ is used to negate the number. If $\langle num \rangle$ is determined to be a decimal or currency, then $\backslash dtlneg$ is used.

$\backslash DTLgneg\{\langle cs \rangle\}\{\langle num \rangle\}$

As $\backslash DTLneg$ but globally sets $\langle cs \rangle$.

$\backslash DTLsqrt\{\langle cs \rangle\}\{\langle num \rangle\}$

Converts the formatted numbers $\langle num \rangle$ to a plain number and stores the square root ($\sqrt{\langle num \rangle}$) as a formatted number in the command $\langle cs \rangle$. The square root is calculated using $\backslash dtlsqrt$.

$\backslash DTLgsqrt\{\langle cs \rangle\}\{\langle num \rangle\}$

As $\backslash DTLsqrt$ but globally sets $\langle cs \rangle$.

There is no equivalent to $\backslash dtlroot$. If an arbitrary root is required for a formatted number, you will have to convert the formatted number to a plain number with $\backslash DTLconverttodecimal$ and use $\backslash dtlroot$.

$\backslash DTLround\{\langle cs \rangle\}\{\langle num \rangle\}\{\langle num digits \rangle\}$

Converts the formatted numbers $\langle num \rangle$ to a plain number, rounds it to $\langle num digits \rangle$ (using $\backslash dtlround$), and stores the result as a formatted number in the command $\langle cs \rangle$.

`\DTLground{⟨cs⟩}{⟨num⟩}{⟨num digits⟩}`

As `\DTLround` but globally sets `⟨cs⟩`.

`\DTLtrunc{⟨cs⟩}{⟨num⟩}{⟨num digits⟩}`

Converts the formatted numbers `⟨num⟩` to a plain number, truncates it to `⟨num digits⟩` (using `\dtltrunc`), and stores the result as a formatted number in the command `⟨cs⟩`.

`\DTLgtrunc{⟨cs⟩}{⟨num⟩}{⟨num digits⟩}`

As `\DTLtrunc` but globally sets `⟨cs⟩`.

`\DTLclip{⟨cs⟩}{⟨num⟩}`

Converts the formatted numbers `⟨num⟩` to a plain number, clips it (using `\dtlclip`), and stores the result as a formatted number in the command `⟨cs⟩`.

`\DTLgclip{⟨cs⟩}{⟨num⟩}`

As `\DTLclip` but globally sets `⟨cs⟩`.

When finding the maximum or minimum of formatted numbers the parsing of the values and formatting of the result may lead the result to have a different appearance to its original formatted value.

`\DTLmin{⟨cs⟩}{⟨num1⟩}{⟨num2⟩}`

Converts the formatted numbers `⟨num1⟩` and `⟨num2⟩` to plain numbers and determines the minimum. If parsing determines that `⟨num1⟩` and `⟨num2⟩` are integers then `\ifnum` is used otherwise `\dtlmin` is used. The result is stored as a formatted number in the command `⟨cs⟩`.

The number list should not contain empty elements.

`\DTLgmin{⟨cs⟩}{⟨num1⟩}{⟨num2⟩}`

2. Base Commands (*datatool-base package*)

As `\DTLmin` but globally sets $\langle cs \rangle$.

```
\DTLminall{ $\langle cs \rangle$ }{ $\langle num list \rangle$ }
```

Converts all the formatted numbers in the comma-separated list $\langle num list \rangle$ to plain numbers and determines the minimum (using `\dtlmin`). The result is stored as a formatted number in the command $\langle cs \rangle$.

```
\DTLgminall{ $\langle cs \rangle$ }{ $\langle num list \rangle$ }
```

As `\DTLminall` but globally sets $\langle cs \rangle$.

```
\DTLmax{ $\langle cs \rangle$ }{ $\langle num1 \rangle$ }{ $\langle num2 \rangle$ }
```

Converts the formatted numbers $\langle num1 \rangle$ and $\langle num2 \rangle$ to plain numbers and determines the maximum. If parsing determines that $\langle num1 \rangle$ and $\langle num2 \rangle$ are integers then `\ifnum` is used otherwise `\dtlmax` is used. The result is stored as a formatted number in the command $\langle cs \rangle$.

```
\DTLgmax{ $\langle cs \rangle$ }{ $\langle num1 \rangle$ }{ $\langle num2 \rangle$ }
```

As `\DTLmax` but globally sets $\langle cs \rangle$.

```
\DTLmaxall{ $\langle cs \rangle$ }{ $\langle num list \rangle$ }
```

Converts all the formatted numbers in the comma-separated list $\langle num list \rangle$ to plain numbers and determines the maximum (using `\dtlmax`). The result is stored as a formatted number in the command $\langle cs \rangle$.

```
\DTLgmaxall{ $\langle cs \rangle$ }{ $\langle num list \rangle$ }
```

As `\DTLmaxall` but globally sets $\langle cs \rangle$.

```
\DTLmeanforall{ $\langle cs \rangle$ }{ $\langle num list \rangle$ }
```

Converts all the formatted numbers in the comma-separated list $\langle num list \rangle$ to plain numbers and determines the mean (average) value. The result is stored as a formatted number in the command $\langle cs \rangle$.

```
\DTLgmeanforall{ $\langle cs \rangle$ }{ $\langle num list \rangle$ }
```

As `\DTLmeanforall` but globally sets $\langle cs \rangle$.

```
\DTLvarianceforall{ $\langle cs \rangle$ }{ $\langle num list \rangle$ }
```

Converts all the formatted numbers in the comma-separated list $\langle num list \rangle$ to plain numbers and determines the variance. The result is stored as a formatted number in the command $\langle cs \rangle$.

```
\DTLgvarianceforall{ $\langle cs \rangle$ }{ $\langle num list \rangle$ }
```

As `\DTLvarianceforall` but globally sets $\langle cs \rangle$.

```
\DTLsdforall{ $\langle cs \rangle$ }{ $\langle num list \rangle$ }
```

Converts all the formatted numbers in the comma-separated list $\langle num list \rangle$ to plain numbers and determines the standard deviation. The result is stored as a formatted number in the command $\langle cs \rangle$.

```
\DTLgsdforall{ $\langle cs \rangle$ }{ $\langle num list \rangle$ }
```

As `\DTLsdforall` but globally sets $\langle cs \rangle$.

2.6. Currency

The currency data type is represented by a currency symbol and a numerical value. There is no provision for exchange rates. Commands such as `\DTLadd` parse their arguments (which are provided as formatted numbers) to obtain the actual numerical value, which can then be passed to commands like `\dtladd`, which expect plain number arguments. The result is then formatted to match the dominant data type in the arguments. This means that if one or more of the arguments is a currency value, then the result will use the same currency symbol. Parsing is performed using the same method as `\DTLparse`.

In order for the parser to determine the difference between a currency value and a string (see §2.2), *datatool-base* needs to know the currency symbols. As from version 3.0, *datatool-base* can now load region files that setup the currency associated with the region.

If you don't want the default currency to change when the language changes, use:

```
\DTLsetup{numeric={region-currency=false}}
```

As described in §2.3.2, a plain number can be converted to a formatted currency with `\DTLdecimaltocurrency`. The formatting of the number is performed in the same manner as with `\DTLdecimaltolocale`. The way that the currency symbol is formatted in relation to the formatted number depends on the currency formatting style.

Example 41 has a simple document with no localisation support:

41

```
\documentclass{article}
\usepackage{datatool-base}
```

First the default currency code and symbol are displayed:

```
Currency code: \DTLCurrencyCode.
Currency symbol: \DTLCurrencySymbol.
```

Then a plain number is converted to a formatted currency:

```
\DTLdecimaltocurrency{12345.678}{\formattedresult}
Formatted: \formattedresult.
(Numeric value: \DTLdatumvalue{\formattedresult}.)
```

This will use the current number group character and decimal character to format the value and the current currency symbol and style to format the currency unit.

Next a formatted currency (using the current number group character and decimal character settings) is added to a formatted number. Note that the symbol doesn't need to match the current currency symbol:

```
\$1,234.57 add 1,236.59:
\DTLadd{\total}{\$1,234.57}{1,236.59}
Total: \total.

1,234.57 add £1,236.59:
\DTLadd{\total}{1,234.57}{£1,236.59}
Total: \total.
```

The symbol is ignored during the arithmetic computation. The result is formatted according to the current settings.

Rounding is determined by `\DTLCurrentLocaleCurrencyDP` which is adjusted by regional support.

```
€48,236.59 multiplied by 0.5:  
\DTLmul{\result}{€48,236.59}{0.5}  
\result\_{\DTLdatumvalue{\result}}.
```

Note that the rounding only affects the formatting, not the value stored within the datum control sequence.

To demonstrate currency parsing, `\DTLparse` is used to parse to different currencies. The first has a Euro symbol:

```
\DTLparse\parsed{€19,234.56}  
String value: \parsed.  
Numeric value: \DTLdatumvalue{\parsed}.
```

The second has a pound symbol:

```
\DTLparse\parsed{£28,342.64}  
String value: \parsed.  
Numeric value: \DTLdatumvalue{\parsed}.
```

Note that even though these symbols don't match the current default currency symbol, they are still recognised as currency.

The symbol may also occur after the value:

```
\DTLparse\parsed{19,234.56€}  
String value: \parsed.  
Data type:  
Numeric value: \DTLdatumvalue{\parsed}.
```

The currency style formatting is described in more detail later in this section, but `\DTLfmtcurrency` can be used to apply the current formatting style to the currency symbol provided in the first argument and the formatted number provided in the second argument. Note that this is just a style command, and doesn't parse or format the value. (It's redefined whenever the default currency setting is changed.) This means that the following works fine even though it's using different number group character and decimal character to the current default:

```
Formatting specific currency symbol:  
\DTLfmtcurrency{\texteuro}{12.345,65}
```


2. Base Commands (*datatool-base* package)

The command `\DTLcurrency` is simply a shortcut that uses `\DTLfmtcurrency` with the current default currency symbol:

```
Formatting default currency symbol:  
\DTLcurrency{12 345,65}
```

Again, the value argument is expected to be in the correct format.

The above uses the formatting style for the current default currency, but if a currency has been defined with a three-letter currency code, then `\DTLfmtcurr` may be used to format the currency according to the style and symbol associated with that currency code. Again, the value argument is expected to be in the correct format:

```
Formatting EUR:  
\DTLfmtcurr{EUR}{12.345,65}
```

The “EUR” currency code is predefined by *datatool-base* as it covers an number of regions (although any region that sets “EUR” as the currency should also redefine `\DTLdefaultEURcurrencyfmt` as applicable). Other currency codes need regional support to provide them, which will be covered in the next example.

↑ Example 41: Formatting and Parsing Currency (No Region)

```
Currency code: XXX. Currency symbol: $.  
Formatted: $12,345.68. (Numeric value: 12345.678.)  
$1,234.57 add 1,236.59: Total: $2,471.16.  
1,234.57 add £1,236.59: Total: £2,471.16.  
€48,236.59 multiplied by 0.5: €24,118.30 (24118.295).  
String value: €19,234.56. Numeric value: 19234.56.  
String value: £28,342.64. Numeric value: 28342.64.  
String value: 19,234.56€. Numeric value: 19234.56.  
Formatting specific currency symbol: €12.345,65  
Formatting default currency symbol: $12 345,65  
Formatting EUR: €12.345,65
```

Example 42 requires *datatool-regions* to be installed. The region needs to be established. This can be done by loading a language package first, where the dialect has an associated region. For example:

```
\usepackage[british]{babel}  
\usepackage{datatool-base}
```

42

2. Base Commands (*datatool-base* package)

Or if just the root language is specified, `locales` may be used to add the region to the language:

```
\usepackage[english]{babel}  
\usepackage[locales=GB]{datatool-base}
```

In this example, I'm not using a language package so I need to use the `locales` option with both the language and region in the tag:

```
\usepackage[locales={en-GB}]{datatool-base}
```

First the default currency code and symbol are displayed:

```
Currency code: \DTLCurrencyCode.  
Currency symbol: \DTLCurrencySymbol.
```

As with the previous example, I can use `\DTLdecimaltocurrency` to convert a plain number into formatted currency using the current style settings:

```
\DTLdecimaltocurrency{12345.678}{\formattedresult}  
Formatted: \formattedresult.  
(Numeric value: \DTLdatumvalue{\formattedresult}.)
```

As before, currency can be parsed.

```
\DTLparse\parsed{£28,342.64}  
String value: \parsed.  
Numeric value: \DTLdatumvalue{\parsed}.
```

The currency symbol needs to be known but doesn't need to be the current default. However, the number group character and decimal character must match the current setting.

```
\DTLparse\parsed{€19,234.56}  
String value: \parsed.  
Numeric value: \DTLdatumvalue{\parsed}.
```

A region may provide its own settings. For example, the GB region support provides different number styles: `official` (the default), `education` (a thin space for the number group

character) or `old` (a mid-dot for the decimal character). There is also an option to prefix the currency symbol with the region code:

```
\DTLsetLocaleOptions{GB}{  
  number-style=old,  
  currency-symbol-prefix  
}  
(GB settings: number-style=old,  
currency-symbol-prefix=true.)
```

This affects the formatting:

```
\DTLdecimaltocurrency{12345.678}{\formattedresult}  
Formatted: \formattedresult.  
(Numeric value: \DTLdatumvalue{\formattedresult}.)
```

The `old` number style uses `\textperiodcentered` when formatting but allows `\textperiodcentered` or a mid-dot character or a normal dot when parsing:

```
\DTLparse\parsed{£28,342.648}  
String value: \parsed.  
Numeric value: \DTLdatumvalue{\parsed}.
```

Note that this doesn't round the value or format it. The formatted string is simply parsed to determine its type, numeric value and currency symbol.

The `auto-reformat` option will make `\DTLparse` automatically reformat the string value and, since GBP supports a regional prefix, `region-currency-prefix` may be used to alter the prefix format:

```
\DTLsetup{  
  numeric={  
    auto-reformat,  
    region-currency-prefix=smallcaps  
  }  
}  
(Numeric settings: auto-reformat,  
region-currency-prefix=smallcaps.)
```

Note that the prefix isn't included with the currency symbol obtained with `\DTLdatum-currency`.

```
\DTLparse\parsed{£28,342.648}
String value: \parsed.
Numeric value: \DTLdatumvalue{\parsed}.
Currency symbol: \DTLdatumcurrency{\parsed}.
```

Example 42: Currency Formats (GB Region)

Currency code: GBP. Currency symbol: £.
 Formatted: £12,345.68. (Numeric value: 12345.678.)
 String value: £28,342.64. Numeric value: 28342.64.
 String value: €19,234.56. Numeric value: 19234.56.
 (GB settings: number-style=old, currency-symbol-prefix=true.)
 Formatted: GB£12,345 · 68. (Numeric value: 12345.678.)
 String value: £28,342.648. Numeric value: 28342.648.
 (Numeric settings: auto-reformat, region-currency-prefix=smallcaps.)
 String value: GB£28,342 · 65. Numeric value: 28342.648. Currency symbol: £.

Example 43 has two regions: GB and IE but the same language for both. No language package is loaded. This means that the region hook must be explicitly used to switch between the two regions. The locales are identified:

43

```
\usepackage[locales={en-GB,en-IE}]{datatool-base}
```

For the GB region, I'm going to use the “education” number style, which uses a thin space for the number group character when formatting. For parsing, it allows either a thin space or a normal space:

```
\DTLsetLocaleOptions{GB}{ number-style = education }
```

I'm also going to switch on the `auto-reformat` option:

```
\DTLsetup{numeric={auto-reformat}}
```

Switch to the GB region:

```
\DTLGBLocaleHook
```

and display the currency code and symbol:

```
Currency code: \DTLCurrencyCode.  
Currency symbol: \DTLCurrencySymbol.
```

Convert a plain number to a formatted currency:

```
\DTLdecimaltocurrency{12345.678}{\GBformattedresult}  
Formatted: \GBformattedresult.  
(Numeric value: \DTLdatumvalue{\GBformattedresult}.)
```

Parse a formatted currency:

```
Parsing £12 345.67.  
\DTLparse\GBparsed{£12 345.67}  
  
Parsed: \GBparsed.  
(Numeric value: \DTLdatumvalue{\GBparsed}.)
```

Since the `auto-reformat` option is on, the string value will be reformatted to use a thin space, instead of the normal space used in the original.

The code is similar for the IE region:

```
\DTLIELocaleHook  
Currency code: \DTLCurrencyCode.  
Currency symbol: \DTLCurrencySymbol.  
  
\DTLdecimaltocurrency{12345.678}{\IEformattedresult}  
Formatted: \IEformattedresult.  
(Numeric value: \DTLdatumvalue{\IEformattedresult}.)
```

Note that the number group character has been changed to a comma. The decimal character has been set to a dot, which is the same as before.

2. Base Commands (*datatool-base* package)

```
Parsing €12,345.67.  
\DTLparse\IEparsed{€12,345.67}  
  
Parsed: \IEparsed.  
(Numeric value: \DTLdatumvalue{\IEparsed}.)
```

The package-wide settings are changed:

```
\DTLsetup{numeric={currency-symbol-style=iso}}
```

Both the GB and IE regions support the `currency-symbol-position` setting:

```
\DTLsetLocaleOptions{GB,IE}  
{currency-symbol-position=after}
```

The datum control sequences are redisplayed:

```
\begin{enumerate}  
\item \GBformattedresult.  
\item \GBparsed.  
\item \IEformattedresult.  
\item \IEparsed.  
\end{enumerate}
```

Note that although this has changed the way that the currency symbol is formatted in relation to the value, the formatting of the value hasn't changed.

 Example 43: Currency Formats (GB and IE Regions)


Currency code: GBP. Currency symbol: £.
 Formatted: £12 345.68. (Numeric value: 12345.678.)
 Parsing £12 345.67.
 Parsed: £12 345.67. (Numeric value: 12345.67.)
 Currency code: EUR. Currency symbol: €.
 Formatted: €12,345.68. (Numeric value: 12345.678.)
 Parsing €12,345.67.
 Parsed: €12,345.67. (Numeric value: 12345.67.)

1. 12 345.68 GBP.
2. 12 345.67 GBP.
3. 12,345.68 EUR.
4. 12,345.67 EUR.

If there is no support for your region, or if you are using a currency that's not connected to your region (for example, Bitcoin), then you can use the commands described below to define a currency (if not already provided by *datatool-base*) and to switch to a previously defined currency.



```
\DTLnewcurrencysymbol{<symbol>}
```

This adds *<symbol>* to the list of known currencies (if not already in the list).



Be careful of non-robust currency symbol commands. If these are expanded before the parser scans the value, the symbol won't be detected and the value will be deemed a string instead. (See Example 34.) However, many commands that were previously partially expandable, and therefore susceptible to this problem, have been made robust with recent L^AT_EX kernels.

The set of known currencies is initialised to contain common currency symbols supported by the document encoding, and the currency commands: `\$`, `\pounds`, `\texteuro`, `\textdollar`, `\textsterling`, `\textyen`, `\textwon`, and `\textcurrency`.

The known currency list simply assists parsing, but it's also possible to define a currency with a corresponding ISO code and alternative representation to adjust the way a currency value is formatted.



```
\DTLdefcurrency[<fmt>]{<ISO>}{<symbol>}{<string>}
```

2. Base Commands (*datatool-base package*)

This locally defines a new currency (or redefines an existing currency) identified by the given ISO code. The $\langle symbol \rangle$ argument is the currency symbol using L^AT_EX markup, such as `\pounds` or `\$`, and the $\langle char \rangle$ argument is a string (non-command) representation of the currency symbol, such as £ or \$. (Note that \$ will have category code “other” within the $\langle char \rangle$ argument.)

The $\langle char \rangle$ argument is expanded when the currency is defined. The $\langle symbol \rangle$ argument isn't expanded.

The optional argument $\langle fmt \rangle$ indicates how this currency should be formatted and should end with (or solely consist of) a command that takes two arguments $\{ \langle sym \rangle \} \{ \langle value \rangle \}$. The default is `\dtlcurrdefaultfmt` (see below).

The following command is defined by `\DTLdefcurrency`:

`\DTLcurr $\langle ISO \rangle$`

which expands to:

```
\dtltxsortsort
{ \DTLcurrCodeOrSymOrChar {  $\langle ISO \rangle$  } {  $\langle symbol \rangle$  } {  $\langle char \rangle$  } }
{  $\langle string \rangle$  }
```

where $\langle string \rangle$ is the detokenized $\langle char \rangle$. Additionally, `\DTLdefcurrency` automatically implements:

```
\DTLnewcurrencysymbol {  $\langle symbol \rangle$  }
\DTLnewcurrencysymbol {  $\langle string \rangle$  }
\DTLnewcurrencysymbol { \DTLcurr $\langle ISO \rangle$  }
```

This ensures that the parser can identify $\langle symbol \rangle$, $\langle string \rangle$ and `\DTLcurr $\langle ISO \rangle$` as currency symbols. For example, the file `datatool-GB.ldf` (provided with `datatool-regions`) includes the equivalent to:

```
\DTLdefcurrency [ \datatoolGBcurrencyfmt ] { GBP } { \pounds }
{ £ }
```

(where `\datatoolGBcurrencyfmt` is also provided.) This locally defines a currency identified as GBP, with the associated symbol `\pounds` and character alternative “£”. It also defines the command `\DTLcurrGBP`, and adds `\DTLcurrGBP` to the set of known currencies (“£” and `\pounds` should typically already be in the set). So the above essentially does (where the second argument of `\dtltxsortsort` has been detokenized):

2. Base Commands (*datatool-base* package)

```
\def\DTLcurrGBP{%  
  \dtltxorsort{\DTLcurrCodeOrSymOrChar{GBP}{\pounds}  
  {\pounds}}{\pounds}}  
  \DTLnewcurrencysymbol{\pounds}% redundant  
  \DTLnewcurrencysymbol{\pounds}  
  \DTLnewcurrencysymbol{\DTLcurrGBP}
```

As well as setting the format for the GBP currency to `\datatoolGBcurrencyfmt`.

`\DTLdefcurrency` doesn't change the default currency (see Example 44). It simply defines a currency.

The underlying function used by `\DTLdefcurrency` is:

```
\datatool_def_currency:nnnn {<fmt>} {<ISO>} {<symbol>}  
{<string>} variants: nnnV nne
```

Note that, unlike `\DTLdefcurrency`, this doesn't perform any category code change or expansion for the final argument. (If expansion is needed, one of the variants may be used.) For example, the file `datatool-CA.ldf` has:

```
\datatool_def_currency:nnnV  
  \datatoolCAcurrencyfmt  
  CAD  
  \$  
  \c_dollar_str
```

There is a shortcut that sets the format to `\dtlcurrdefaultfmt`:

```
\datatool_def_currency:nnn {<ISO>} {<symbol>} {<string>}  
 variants: nnV nne
```

This internally calls the `\datatool_def_currency:nnnn` function.

The symbol associated with a defined currency may be changed with:

```
\datatool_set_currency_symbol:nn {<ISO>} {<symbol>}  
 variants: nV ne
```

Note that this also adds the symbol with `\DTLnewcurrencysymbol` but does not remove the previous symbol from the set of known currency symbols.

The symbol and associated string value for a currency that has been defined can be obtained with:

```
\DTLcurrSym{ISO}
```

Expands to the symbol (such as `\$` or `\pounds`) associated with currency *ISO* or to nothing if not defined.

```
\DTLcurrChar{ISO}
```

Expands to the character associated with currency *ISO* or to nothing if not defined (for example, `$` or `£`).

```
\DTLcurrStr{ISO}
```

Expands to the detokenised string value associated with currency *ISO* or to nothing if not defined.

If you don't know whether or not `\DTLcurrISO` has been defined for a given *ISO* code, you can use:

```
\DTLcurr{ISO}
```

This command will expand to `\DTLcurrISO`, if defined, otherwise it will expand to *ISO*. (The `datatooltk` application uses this for currency symbols when importing data that has been given an associated currency code.)

If you want to switch to a previously defined currency, you need to use `\DTLsetdefaultcurrency`. For example:

```
\DTLsetdefaultcurrency{GBP}
```

This is done by `datatool-GB.ldf` in the language hook.

If no localisation file has been loaded (see §2.3), then the default is ISO code “XXX” and symbol `\$`. (The default symbol is for backward-compatibility, and `\$` was one of the few currency commands guaranteed to be defined when the first version of `datatool` was written.)

The following currencies are defined by `datatool-base`: “XXX” (associated command `\DTLcurrXXX`), “XBT” (associated command `\DTLcurrXBT`), “EUR” (associated command `\DTLcurrEUR`).

The “XXX” and “XBT” currencies use the default currency formatting command `\dtlcurr-defaultfmt` but the “EUR” currency is associated with:

`\DTLdefaultEURcurrencyfmt`

The default definition is just `\dtlcurrdefaultfmt` but this makes it possible to vary the format of EUR specifically without affecting other currencies.

If you prefer a different symbol, you can use `\datatool_set_currency_symbol:nn`. For example:

```
\newfontfamily\liberationserif{Liberation Serif}
\NewDocumentCommand{\bitcoin}{}{{\liberationserif ₿}}

\ExplSyntaxOn
\datatool_set_currency_symbol:nn { XBT } { \bitcoin }
\ExplSyntaxOff
```

The currency string depends on the file encoding (see §2.3.1).

`\DTLCurrencySymbol`

This command is simply defined to the internal command used to store the default currency symbol. It's provided to allow access to the currency symbol without having to switch category code. Redefining this command will not change the default currency symbol. The command `\DTLCurrencySymbol` is not automatically added to the list of known currency symbols.

If you want to change the default currency, use `\DTLsetdefaultcurrency`. Don't redefine placeholder commands, such as `\DTLCurrencySymbol` and `\DTLCurrencyCode`.

`\DTLCurrencyCode`

This command is redefined by `\DTLsetdefaultcurrency` to expand to the associated ISO code.

`\DTLfmtcurrency{<symbol>}{<value>}`

This command is redefined by `\DTLsetdefaultcurrency` to expand to the associated currency formatting code (as supplied in the *<fmt>* optional argument of `\DTLdefcurrency`). The *<symbol>* argument doesn't need to have been identified as a known currency symbol, but

2. Base Commands (*datatool-base* package)

the $\langle value \rangle$ must be a formatted number with the correct rounding that uses the current number group character and decimal character.

The default definition of `\DTLfmtcurrency` just does:

```
\dtlcurrdefaultfmt{ $\langle symbol \rangle$ }{ $\langle value \rangle$ }
```

which is defined to use the following command.

```
\dtlcurrprefixfmt{ $\langle symbol \rangle$ }{ $\langle value \rangle$ }
```

This internally uses:

```
\datatool_prefix_adjust_sign:nnn { $\langle symbol \rangle$ } { $\langle sep \rangle$ }  
{ $\langle value \rangle$ }
```

(where the separator is `\dtlcurrfmtsep`) which tests if $\langle value \rangle$ starts with a plus (+) or minus (-) and, if so, shifts the sign in front of the symbol and encapsulates it with:

```
\datatool_adjust_sign_fmt:n{ $\langle sign \rangle$ }
```

This will convert the hyphen-minus sign (-) to `\textminus` if not in math mode.

For currencies that have the symbol at the end:

```
\dtlcurrsuffixfmt{ $\langle symbol \rangle$ }{ $\langle value \rangle$ }
```

This internally uses:

```
\datatool_suffix_adjust_sign:nnn { $\langle symbol \rangle$ } { $\langle sep \rangle$ }  
{ $\langle value \rangle$ }
```

(where the separator is `\dtlcurrfmtsep`) which similarly adjusts the leading sign (if present) but in this case puts the separator and symbol after the value.

Both `\dtlcurrprefixfmt` and `\dtlcurrsuffixfmt` use:

```
\dtlcurrfmtsep
```

as the separator. This defaults to:

```
\DTLcurrCodeOrSymOrChar
{~}
{\dtlcurrfmtsymsep}
{\dtlcurrfmtsymsep}
```

This expands to a space with `currency-symbol-style=iso`, otherwise to:

```
\dtlcurrfmtsymsep
```

initial: empty
region-sensitive

This should be redefined by region files.

Since `\DTLfmtcurrency` will change its format according to the current localisation settings, which may not be appropriate, you may prefer to use:

```
\DTLfmtcurr{<currency-code>}{<value>}
```

This will use the format associated with the given currency code. If the currency code hasn't been defined, then this simply expands to `\DTLcurrency{<num>}` instead.

Region files may provide their own format that inserts a tag before the currency symbol. For example, `datatool-GB.ldf` provides:

```
\newcommand\datatoolGBcurrencyfmt[2]{%
\dtlcurrprefixfmt
{\datatoolGBsymbolprefix{GB}#1}% symbol
{#2}% value
}
```

The default definition of `\datatoolGBsymbolprefix` does nothing, but the region provides an option to redefine this command to `\datatool_currency_symbol_region_prefix:n`.

Note that `\DTLfmtcurrency` requires the currency symbol as an argument, which doesn't have to be the default symbol (or even a recognised currency symbol). If you want the default symbol without having to specify it, you can use:

```
\DTLcurrency{<value>}
```

This expands to `\DTLfmtcurrency{<sym>}{<value>}` where `<sym>` is the default currency symbol, which is initially `\$` but will be changed to `\DTLcurr<ISO>` by `\DTLset-defaultcurrency{<ISO>}`.

`\DTLcurrCodeOrSymOrChar{ISO}{symbol}{character}`

This is used in both `\DTLcurrISO` and `\dtlcurrfmtsep` and should be defined to expand to one of its arguments (ignoring the other two). The default is to expand to *symbol*. This means that `\DTLcurrency` will use the symbol command associated with the default currency. You can redefine `\DTLcurrCodeOrSymOrChar` to expand to a different argument if you prefer. The `numeric` option `currency-symbol-style` redefines `\DTLcurrCodeOrSymOrChar`.

`\DTLdecimaltocurrency` internally uses `\DTLfmtcurrency` with the value rounded to the decimal places specified by `\DTLCurrentLocale-CurrencyDP` and formatted according to the current number group character and decimal character. The optional argument to `\DTLdecimaltocurrency` is used in the *symbol* argument of `\DTLfmtcurrency`.

The `datatool-GB.ldf` file provided with `datatool-regions` provides the GBP currency. The example below is provided to demonstrate how to define currencies and modify the formatting. If you want to add support for your region, there is a Perl script in the `datatool-regions` GitHub repository that can get you started. You can then add your region file via a pull request. See the “README” file at <https://github.com/nlct/datatool-regions> for further details.

Example 44 ensures that `\DTLcurrGBP`, `\pounds` and £ are all recognised as currency symbols when parsing currency values (although `\pounds` and £ are recognised by default). However, it’s necessary to explicitly change the default currency for instances where the currency symbol is omitted:

44

```
Default currency: \DTLCurrencyCode.
\DTLdecimaltocurrency{1234.567}{\result}
Formatted value: \result.

£1.99:
  \DTLifcurrency{£1.99}{currency}{not currency};
  \DTLfmtcurrency{£}{1.99}:
  \DTLifcurrency{\DTLfmtcurrency{£}{1.99}}{currency}
  {not currency}.

Defining GBP.
\DTLdefcurrency{GBP}{\pounds}{£}
Default currency: \DTLCurrencyCode.
```

```

£1.99:
  \DTLifcurrency{£1.99}{currency}{not currency}.

Switching to GBP. \DTLsetdefaultcurrency{GBP}
Default currency: \DTLCurrencyCode.

\DTLdecimaltocurrency{1234.567}{\result}
Formatted value: \result.

\renewcommand{\dtlcurrdefaultfmt}{\dtlcurrsuffixfmt}
\renewcommand{\DTLcurrCodeOrSymOrChar}[3]{#1}
Formatted value: \result.

\DTLaddall{\result}
{\pounds2.50,\DTLcurrGBP 1.25,£0.25}
Formatted value: \result.

```

↑ Example 44: Defining a Currency



```

Default currency: XXX. Formatted value: $1,234.57.
£1.99: currency; £1.99: currency.
Defining GBP. Default currency: XXX.
£1.99: currency.
Switching to GBP. Default currency: GBP.
Formatted value: £1,234.57.
Formatted value: 1,234.57 GBP.
Formatted value: 4.00 GBP.

```

Note that `\result` is defined as a datum control sequence. This means that the resulting command doesn't need to be reparsed to obtain its numerical value.

In the case of `\DTLaddall` the symbol from the final currency in the list is used (the character “£”). So the final `\result` (indirectly) expands to `\DTLfmtcurrency{£}{4}`. This now shows the currency unit as a suffix because of the redefinition of `\dtlcurr-defaultfmt`.

2.7. Dates and Times

The temporal data types (`datetime`, `date`, and `time`) were only added to *datatool-base* version 3.0 and are still experimental so this feature is off by default. Parsing can be enabled with the `datetime` option.

Options that govern date and time parsing can be set within the `datetime` setting value.

For example:

```
\DTLsetup{ datetime={parse=auto-reformat} }
```

Available options are listed below.

`parse=<value>` *initial: false*

Determines whether or not to check for temporal values when parsing. The default is:

```
\DTLsetup{
  datetime={
    parse=iso+region,
    auto-reformat=false,
    parse=false
  }
}
```

`parse=false`

Don't check for temporal values when parsing. (The default.)

`parse=true`

Check for temporal values when parsing. This switches on parsing without altering any of the other settings.

`parse=parse-only`

Check for temporal values when parsing but don't alter the original. This is equivalent to `parse=true`, `auto-reformat=false`.

`parse=auto-reformat`

Check for temporal values when parsing and reformat the original. This is equivalent to `parse=true`, `auto-reformat=true`.

`parse=iso-only`

Check for temporal values when parsing but only check for ISO formatted dates and times.

2. Base Commands (*datatool-base* package)

For example, 2025-01-14 (date) or 16:25:02 (time) or 2025-01-14T16:25:02 (timestamp with no offset) or 2025-01-14T16:25:02+01:00 (timestamp with offset).

`parse=region-only`

Check for temporal values when parsing but only parse the current region's date and time formatting. This requires localisation support. See §2.3.

`parse=iso+region`

First check for ISO format then check for current region's format. If there is no localisation support provided, this will be equivalent to `parse=iso-only`

`auto-reformat=<value>`

initial: **false**

If temporal parsing is on, this option determines whether or not the original value should be reformatted.

This option has no effect with `csv-content=no-parse` as the values aren't parsed. Use `convert-numbers` instead.

`auto-reformat=false`

If temporal parsing is on, don't reformat the original.

`auto-reformat=true`

If temporal parsing is on, reformat the original according to the current settings. (According to the `auto-reformat-types` setting.)

`auto-reformat=region`

If temporal parsing is on, reformat the original according to the region settings. (Provided the temporal type is included in the `auto-reformat-types` setting.) This essentially does

2. Base Commands (*datatool-base* package)

```
\DTLsetup{datetime=auto-reformat=true}
\renewcommand\DataToolDateFmt{%
  \DTLCurrentLocaleFormatDate}
\renewcommand\DataToolTimeFmt{%
  \DTLCurrentLocaleFormatTime}
\renewcommand\DataToolTimeZoneFmt{%
  \DTLCurrentLocaleFormatTimeZone}
\renewcommand\DataToolTimeStampWithZoneFmt{%
  \DTLCurrentLocaleFormatTimeStampWithZone}
\renewcommand\DataToolTimeStampNoZoneFmt{%
  \DTLCurrentLocaleFormatTimeStampNoZone}
\renewcommand\DataToolTimeStampFmtSep{%
  \DTLCurrentLocaleTimeStampFmtSep}
```

Note that regional support may simply defer to `datetime2`, if it has been installed, or may just use the ISO numeric format. See the applicable localisation documentation for further details. For example,

```
texdoc datatool-regions
```

```
auto-reformat=iso
```

If temporal parsing is on, redefine the formatting commands to use the ISO numeric format. (Provided the temporal type is included in the `auto-reformat-types` setting.)

```
auto-reformat=datetime2
```

If temporal parsing is on, redefine the formatting commands to use the applicable `datetime2` formatting commands. (Provided the temporal type is included in the `auto-reformat-types` setting.)

Note that this will require `datetime2` to be loaded and you will need to set the style using `datetime2`'s interface.

The `datetime2` style defaults to ISO unless regional support has been requested and provided.

Example 45 illustrates the different settings:

```
Parsing off by default.
\DTLparse\result{2025-01-09}
```

45

```
String value: \result.
Data type: \DTLdatumtype{\result}.
Value: \DTLdatumvalue{\result}.
\DTLparse\result{2025-01-09T14:42:01}
String value: \result.
Data type: \DTLdatumtype{\result}.
Value: \DTLdatumvalue{\result}.
\DTLparse\result{2025-01-09T15:42:01+01:00}
String value: \result.
Data type: \DTLdatumtype{\result}.
Value: \DTLdatumvalue{\result}.
\DTLparse\result{14:42:01}
String value: \result.
Data type: \DTLdatumtype{\result}.
Value: \DTLdatumvalue{\result}.
\DTLsetup{datetime={parse}}
Parsing on.
\DTLparse\result{2025-01-09}
String value: \result.
Data type: \DTLdatumtype{\result}.
Value: \DTLdatumvalue{\result}.
\DTLparse\result{2025-01-09T14:42:01}
String value: \result.
Data type: \DTLdatumtype{\result}.
Value: \DTLdatumvalue{\result}.
\DTLparse\result{2025-01-09T15:42:01+01:00}
String value: \result.
Data type: \DTLdatumtype{\result}.
Value: \DTLdatumvalue{\result}.
\DTLparse\result{14:42:01}
String value: \result.
Data type: \DTLdatumtype{\result}.
Value: \DTLdatumvalue{\result}.
```

 Example 45: Parsing Dates and Times


Parsing off by default.

String value: 2025-01-09. Data type: 0. Value: .

String value: 2025-01-09T14:42:01. Data type: 0. Value: .

String value: 2025-01-09T15:42:01+01:00. Data type: 0. Value: .

String value: 14:42:01. Data type: 0. Value: .

Parsing on.

String value: 2025-01-09. Data type: 5. Value: 2460685.

String value: 2025-01-09T14:42:01. Data type: 4. Value: 2460685.112511574.

String value: 2025-01-09T15:42:01+01:00. Data type: 4. Value: 2460685.112511574.

String value: 14:42:01. Data type: 6. Value: 0.1125115740740741.

Example 46 loads `datetime2` and not only parses but also reformats the string representation. (Advanced users: the ISO string can be extracted with `\datatool_extract_timestamp:NN`, see §2.2.4.)

 46



```

\usepackage[en-GB]{datetime2}
\usepackage{datatool-base}
\begin{document}
\DTLsetup{datetime={parse=auto-reformat}}
\DTLparse\result{2025-01-09}
String value: \result.
Data type: \DTLdatumtype{\result}.
Value: \DTLdatumvalue{\result}.
\DTLparse\result{2025-01-09T14:42:01}
String value: \result.
Data type: \DTLdatumtype{\result}.
Value: \DTLdatumvalue{\result}.
\DTLparse\result{2025-01-09T15:42:01+01:00}
String value: \result.
Data type: \DTLdatumtype{\result}.
Value: \DTLdatumvalue{\result}.
\DTLparse\result{14:42:01}
String value: \result.
Data type: \DTLdatumtype{\result}.
Value: \DTLdatumvalue{\result}.
\end{document}

```

Note that `datatool-base` will automatically pick up `datetime2`'s regional setting. This will require not only `datetime2` but also `datetime2-english` (which will be implicitly loaded by `datetime2` if it is installed).

Example 46: Parsing Dates and Times and Reformatting

String value: 9th January 2025. Data type: 5. Value: 2460685.
 String value: 9th January 2025 2:42pm GMT. Data type: 4. Value: 2460685.112511574.
 String value: 9th January 2025 3:42pm BST. Data type: 4. Value: 2460685.112511574.
 String value: 2:42pm. Data type: 6. Value: 0.1125115740740741.

The `auto-reformat=true` setting will cause commands like `\DTLparse` to replace the original string with the applicable commands listed below. These commands will be redefined by settings such as `auto-reformat=iso` or you can redefine them yourself.

```
\DataToolDateTimeFmt { <date-specs> } { <time-specs> } { <offset-specs> }
```

Use to format timestamps (datetime date types). Any of the arguments may be empty, which indicates to omit that part, but if not empty the arguments should be in the appropriate format to pass to `\DataToolDateTimeFmt` (`<date-specs>` should be `{ <year> } { <month> } { <day> } { <dow> }`), `\DataToolTimeFmt` (`<time-specs>` should be `{ <hour> } { <minute> } { <second> }`) and `\DataToolTimeZoneFmt` (`<offset-specs>` should be `{ <tzh> } { <tzm> }`).

If both the `<date-specs>` and `<time-specs>` are non-empty then, if `<offset-specs>` is empty `\DataToolTimeStampNoZoneFmt` will be used or if `<offset-specs>` is not empty `\DataToolTimeStampWithZoneFmt` will be used.

```
\DataToolTimeStampNoZoneFmt { <year> } { <month> } { <day> } { <dow> }
{ <hour> } { <minute> } { <second> }
```

Formats the date and time. The default definition is to use `\DTLCurrentLocaleFormatTimeStampNoZone`, which may be redefined by localisation support.

```
\DataToolTimeStampWithZoneFmt { <year> } { <month> } { <day> } { <dow> }
{ <hour> } { <minute> } { <second> } { <tzh> } { <tzm> }
```

Formats the date, time and time zone. The default definition is to use `\DTLCurrentLocaleFormatTimeStampWithZone`, which may be redefined by localisation support.

The above commands may use:

```
\DataToolTimeStampFmtSep
```

This is placed between the date and time. The default definition simply expands to `\DTLCurrentLocaleTimeStampFmtSep`.



```
\DataToolDateFmt { <year> } { <month> } { <day> } { <dow> }
```

Formats the date. The default definition is to use `\DTLCurrentLocaleFormatDate`, which may be redefined by localisation support. Note that the `<dow>` argument may be empty. Otherwise, all arguments must be integers.



```
\DataToolTimeFmt { <hour> } { <minute> } { <second> }
```

Formats the time. The default definition is to use `\DTLCurrentLocaleFormatTime`, which may be redefined by localisation support. Note that the `<second>` argument may be empty. Otherwise, all arguments must be integers.



```
\DataToolTimeZoneFmt { <tzh> } { <tzm> }
```

Formats the time zone offset. The default definition is to use `\DTLCurrentLocaleFormatTimeZone`, which may be redefined by localisation support. All arguments must be integers.

2.8. Strings



The string related code provided by *datatool-base* has been rewritten in v3.0. This means that there may be some differences in the results from earlier versions. You may need to use rollback if this causes a problem for existing documents.

The string data type is any non-empty content that can't be parsed as a number (or currency). For more information on data types, see §2.2. For conditionals, see §2.4. For CSV lists, see §2.9. The commands described below assume that the text arguments are strings without parsing them to determine their data type. Unexpected results may occur if the text includes math-mode content.

2.8.1. Substitution and String Splitting



```
\DTLsubstitute { <cs> } { <original> } { <replacement> }
```

Substitutes the first occurrence of `<original>` with `<replacement>` within the expansion text of the command `<cs>`.



```
\DTLsubstituteall{⟨cs⟩}{⟨original⟩}{⟨replacement⟩}
```

Substitutes all occurrences of *⟨original⟩* with *⟨replacement⟩* within the expansion text of the command *⟨cs⟩*.



```
\DTLsplitstring{⟨string⟩}{⟨split text⟩}{⟨before cmd⟩}{⟨after cmd⟩}
```

Splits *⟨string⟩* at *⟨split text⟩* and defines *⟨before cmd⟩* to the pre-split text and *⟨after cmd⟩* to the post-split text. Note that *⟨string⟩* and *⟨split text⟩* are not expanded.



```
\DTLxsplitstring{⟨string⟩}{⟨split text⟩}{⟨before cmd⟩}{⟨after cmd⟩}
```

As *\DTLsplitstring* but expands *⟨string⟩* and *⟨split text⟩* once.

Note that in each case the change is localised to the current scope. Example 47 demonstrates this by adding grouping to limit the effect of the change.

 47



```
\newcommand{\test}
{The goose looked at a book and said \emph{ooh}..}
{% local scope
  Original: \test
  \DTLsubstitute{\test}{oo}{ee}
  Substituted first: \test
}

{% local scope
  Original: \test
  \DTLsubstituteall{\test}{oo}{ee}
  Substituted all: \test
}

Split on `looked' (no expansion)
\DTLsplitstring{\test}{looked}{\before}{\after}

Before: ` \before'. After: ` \after'

Split on `looked' (with expansion)
\DTLxsplitstring{\test}{looked}{\before}{\after}

Before: ` \before'. After: ` \after'
```

Note that the “oo” in `\emph{ ooh }` isn’t substituted as it’s inside a group, which hides it from the search.

↑ Example 47: String Substitution and Splitting

Original: The goose looked at a book and said *ooh*. Substituted first: The geese looked at a book and said *ooh*.

Original: The goose looked at a book and said *ooh*. Substituted all: The geese leeked at a beek and said *ooh*.

Split on ‘looked’ (no expansion)

Before: ‘The goose looked at a book and said *ooh*.’. After: ‘

Split on ‘looked’ (with expansion)

Before: ‘The goose ’. After: ‘ at a book and said *ooh*.’

For more complex substitutions, including replacing commands or command arguments, use $\LaTeX$ 3 regular expressions (see §1.2.1).

2.8.2. Initial Letters

```
\DTLinitials{<text>}
```

This is simply a shortcut that uses `\DTLstoreinitials` to obtain the initials and then displays the result.

```
\xDTLinitials{<text>}
```

Designed for use with placeholder commands, this expands the first token in its argument before passing it to `\DTLinitials`.

```
\DTLstoreinitials{<text>}{<cs>}
```

Splits `<text>` into words and stores the initial of each word in the control sequence `<cs>`. Normal space characters, the hyphen character (–), non-breakable spaces (~) and space commands `\nobreakspace` and `\space` are all considered word boundaries. Words broken by an apostrophe are also detected but by default the apostrophe and following initial are ignored. For example, “O’Brien” will become just “O” (followed by a dot) but it will actually be converted to:

```
\DTLaposinitialpunc{O}{B}{<punc>}
```

Only the first apostrophe in the word is treated in this way. If a word has multiple apostrophes, such as “fo’c’s’le” in the example below, then the word will be split on the first apostrophe (“fo”,

which has the initial “f” and “c’s’le”, which has the initial “c”). An apostrophe at the end of a word is considered trailing punctuation.

Once the supplied *<text>* has been split into words, `\DTLstoreinitials` uses:

```
\DTLStoreInitialGetLetter{<word>}{<cs>}
```

to get the initial letter of each word. The default definition just uses `\DTLGetInitialLetter` which means that the results can vary if localisation support is provided. Ordinarily, this should produce at least one letter, but it’s possible for `\DTLStoreInitialGetLetter` to set the control sequence *<cs>* to expand to nothing. If this occurs, `\DTLstoreinitials` will usually skip the initial and the following punctuation. The exception is where a word is split by an apostrophe.

In the case of *<head>’<tail>*, where *<H>* indicates the initial for *<head>* and *<T>* indicates the initial for *<tail>* (as obtained by `\DTLStoreInitialGetLetter`), then if *<H>* and *<T>* are both empty, the initials and following punctuation are omitted. If *<H>* is empty but *<T>* isn’t then `\DTLinitialpunc{<T>}{<punc>}` is used, otherwise (regardless of whether or not *<T>* is empty) `\DTLaposinitialpunc{<H>}{<T>}{<punc>}` is used.

```
\DTLinitialpunc{<letter>}{<punc>}
```

This command is used to encapsulate each initial (except in the case of a word containing an apostrophe) and the following period/full stop. The first argument *<letter>* is the initial that was obtained by `\DTLStoreInitialGetLetter` and the second argument *<punc>* is the command that may expand to the trailing punctuation character (see below).

```
\DTLaposinitialpunc{<H>}{<T>}{<punc>}
```

This command is used to encapsulate an initial in a word split by an apostrophe in the form *<head>’<tail>*. The first argument *<H>* is the initial for *<head>* and the second argument *<T>* is the initial for *<tail>*. The final argument *<punc>* is as for `\DTLinitialpunc`. By default this just expands to *<H><punc>* (that is, the initial following the apostrophe is ignored).

The following commands are used for the punctuation.

```
\DTLbetweeninitials
```

initial: .

Placed between initials.

```
\DTLafterinitials
```

initial: .

Placed after the initials (that is, at the end after the final initial).



```
\DTLafterinitialbeforehyphen
```

initial: .

Placed after an initial before a hyphen.



```
\DTLinitialhyphen
```

initial: -

Placed where a hyphen occurs. Note that this isn't included in the final argument of `\DTLinitialpunc` or `\DTLaposinitialpunc`.



`\DTLstoreinitials` is designed for text consisting of words with possible leading or trailing punctuation. Aside from apostrophes and hyphens, mid-word punctuation isn't supported. This is demonstrated in Example 48 where the sequence “+12x, y” is skipped.

If you want to remove the dots, you can either redefine `\DTLbetweeninitials`, `\DTLafterinitials` and `\DTLafterinitialbeforehyphen` to do nothing or redefine `\DTLinitialpunc` and `\DTLaposinitialpunc` to ignore the final argument. If you want to remove the hyphen then you need to redefine `\DTLinitialhyphen` to do nothing.



```
\DTLGetInitialLetter{<text>}{<cs>}
```

Obtains the first letter of `<text>` and stores it in the control sequence `<cs>`. This is intended for use in commands that need initials or letter groups and is governed by the `initial-purify` option.

The `initial-purify=early` setting makes `\DTLGetInitialLetter` purify the `<text>` argument (that is, expand and remove functions in `<text>`) before applying the initial letter algorithm. With `initial-purify=late`, the `<text>` argument won't be expanded until content is passed to `\DTLCurrentLocaleGetInitialLetter` (steps 3 or 4). Note that if `initial-purify=early` then step 3 in the algorithm below won't apply since any commands will have already been stripped or replaced.

The algorithm used by `\DTLGetInitialLetter{<text>}{<cs>}` is as follows:

1. if `<text>` is blank (empty or spaces) then `<cs>` is set to empty;
2. if `<text>` starts with a group, the content of the group is assumed to be an initial letter (even if it consists of multiple letter or non-letter characters) and `<cs>` will be set to the purified content of that group;
3. if `<text>` starts with `\<cmd>{<substr>}` then `<cs>` will be set to `\<cmd>{<letter>}` where `<letter>` is obtained from applying `\DTLCurrentLocaleGetInitialLetter` to the `<substr>` argument;

4. otherwise `\DTLCurrentLocaleGetInitialLetter` is used to obtain the first letter of $\langle text \rangle$.

See §2.8.2.2.



If localisation support is provided by a `datatool- $\langle locale \rangle$ .ldf` file, then that should define `\DTLCurrentLocaleGetInitialLetter` as described in §2.3.

2.8.2.1. Initial Letters Example

Example 48 obtains initials from names containing hyphens and apostrophes.

48



```
Marie\space Élise del~Rosario:
\DTLinitials{Marie\space Élise del~Rosario}

Élouise-Mary de Vere: \DTLinitials{Élouise-
Mary de Vere}

Mary-Jane d'Arcy: \DTLinitials{Mary-Jane d'Arcy}

Mary-Jane d'Arcy-Lancaster:
\DTLinitials{Mary-Jane d'Arcy-Lancaster}

Mary-Jane d'Arcy FitzGerald:
\DTLinitials{Mary-Jane d'Arcy FitzGerald}

Niall O'Brien: \DTLinitials{Niall O'Brien}

De'Ondre Andros: \DTLinitials{De'Ondre Andros}

Dickie `Quack' von Duck:
\DTLinitials{Dickie `Quack' von Duck}
```

Aside from apostrophes and hyphens, mid-word punctuation isn't supported. The sequence “+12x, y” in the following will be skipped because it isn't recognised as a word.



```
@aardvark +12x,y fo'c's'le *zebra?:
\DTLinitials{@aardvark +12x,y fo'c's'le *zebra?}
```

`\DTLStoreInitialGetLetter` is then redefined to set the second argument to empty for the given set of words: “d”, “de”, “del” and “von”. This means that they will be omitted from

the list of initials.

```
Skip `d`, `de`, `del`, and `von`:
\renewcommand{\DTLStoreInitialGetLetter}[2]{%
  \DTLifinlist{#1}{d,de,del,von}{\def#2{}}
  {\DTLGetInitialLetter{#1}{#2}}%
}
```

The same names and words are repeated to illustrate the difference.

The default definition of `\DTLStoreInitialGetLetter` means that, for example, “d’Arcy” has the initial “d” but this redefinition will change the initial for “d’Arcy” to “A”. This is a better method than simply redefining `\DTLaposinitialpunc` to expand to the second argument, which would interfere with “O’Brien” and “De’Ondre”.

↑ Example 48: Name or Phrase Initials

```
Marie Élise del Rosario: M.É.d.R.
Élouise-Mary de Vere: É.-M.d.V.
Mary-Jane d’Arcy: M.-J.d.
Mary-Jane d’Arcy-Lancaster: M.-J.d.-L.
Mary-Jane d’Arcy FitzGerald: M.-J.d.F.
Niall O’Brien: N.O.
De’Ondre Andros: D.A.
Dickie ‘Quack’ von Duck: D.Q.v.D.
@aardvark +12x,y fo’c’s’le *zebra?: a.f.z.
Skip `d`, `de`, `del`, and `von`:
Marie Élise del Rosario: M.É.R.
Élouise-Mary de Vere: É.-M.V.
Mary-Jane d’Arcy: M.-J.A.
Mary-Jane d’Arcy-Lancaster: M.-J.A.-L.
Mary-Jane d’Arcy FitzGerald: M.-J.A.F.
Niall O’Brien: N.O.
De’Ondre Andros: D.A.
Dickie ‘Quack’ von Duck: D.Q.D.
@aardvark +12x,y fo’c’s’le *zebra?: a.f.z.
```

2.8.2.2. Initial Letters with UTF-8 Example

Example 49 has words containing UTF-8 characters.

49

```

ábc: \DTLGetInitialLetter{ábc}{\result}
Initial: \result.
{áb}c (áb grouped): \DTLGetInitialLetter{{áb}c}
{\result}
Initial: \result.

``ábc'': \DTLGetInitialLetter{``ábc''}{\result}
Initial: \result.
``{áb}c'' (áb grouped): \DTLGetInitialLetter{``{áb}
c''}{\result}
Initial: \result.

```

Note the difference between {áb}c (which satisfies step 2 of the \DTLGetInitialLetter algorithm) and ``{áb}c'' (which doesn't).

↗ Example 49: Word Initial Letter with UTF-8

```

ábc: Initial: á. ábc (áb grouped): Initial: áb.
“ábc”: Initial: á. “ábc” (áb grouped): Initial: á.

```

2.8.2.3. Initial Letters with Commands Example

Example 50 has words containing containing commands. In the first instance, the command is an accent command, which can expand. The second is a robust formatting command.

50

```

Purify early.\DTLsetup{initial-purify=early}

\ '{a}bc (accent command): \DTLGetInitialLetter{\ '{a}
bc}{\result}
Initial: \result.
\emph{ábc}: \DTLGetInitialLetter{\emph{ábc}}{\result}

Initial: \result.

Purify late.\DTLsetup{initial-purify=late}

```

```
\'{a}bc (accent command): \DTLGetInitialLetter{\'{a}
bc}{\result}
Initial: \result.
\emph{ábc}: \DTLGetInitialLetter{\emph{ábc}}{\result}

Initial: \result.
```

Note that in the last case above, the formatting command `\emph` isn't stripped.

↑ Example 50: Word Initial Commands

Purify early.
`ábc` (accent command): Initial: `á`.
`ábc`: Initial: `á`.
Purify late.
`ábc` (accent command): Initial: `á`.
`ábc`: Initial: `á`.

2.8.3. Advanced Utility Commands

These commands use L^AT_EX3 syntax so you will need `\ExplSyntaxOn` to change the category codes.

```
\datatool_pad_trailing_zeros:Nn <tl-var> {<n>}
```

This command is intended for use with token list variables that store a plain number and will pad `<tl-var>` with 0s to ensure that there are a minimum of `<n>` digits after the decimal point. If the number in `<tl-var>` was originally an integer, it will become a decimal with `<n>` 0s after the decimal point. This command does nothing if `<n>` is not greater than zero.

The `<token list>` should contain a plain number (decimal or integer) before use but there is no check to ensure this. The command simply tests for the presence of a decimal point (`.`) within `<token list>`.

The commands `\datatool_measure_<type>:Nn` are simply shortcuts that use `\settoheight`, `\settodepth` and `\settoheight` with a hook to disable problematic commands. That is, each command is defined to do:

```
\setto<type> <dim> {<hook><text>}
```

2. Base Commands (*datatool-base package*)

The commands `\datatool_measure_ht_plus_dp:Nn` and `\datatool_measure:NNNn` are slightly more complicated, but essentially do something similar.

The hook is a token list variable:

```
\l_datatool_measure_hook_tl
```

The default definition disables `\label`, `\ref` and `\pageref`, makes `\refstepcounter` behave like `\stepcounter`, and `\hypertarget` and `\hyperlink` will simply expand to their second argument.

If you are using a package that redefines any of those commands to include a starred form or optional argument and you are using that syntax within $\langle text \rangle$, then you will need to adjust `\l_datatool_measure_hook_tl` as applicable.

```
\datatool_measure_width:Nn  $\langle dim \rangle$  { $\langle text \rangle$ }
```

Measures the width of the supplied text with problematic commands locally disabled by the hook.

```
\datatool_measure_height:Nn  $\langle dim \rangle$  { $\langle text \rangle$ }
```

Measures the height of the supplied text with problematic commands locally disabled by the hook.

```
\datatool_measure_depth:Nn  $\langle dim \rangle$  { $\langle text \rangle$ }
```

Measures the depth of the supplied text with problematic commands locally disabled by the hook.

```
\datatool_measure_ht_plus_dp:Nn  $\langle dim \rangle$  { $\langle text \rangle$ }
```

Measures the combined height and depth of the supplied text with problematic commands locally disabled by the hook.

```
\datatool_measure:NNNn  $\langle wd-dim \rangle$   $\langle ht-dim \rangle$   $\langle dp-dim \rangle$  { $\langle text \rangle$ }
```

Measures the width, height and depth of the supplied text with problematic commands locally disabled by the hook.

The following commands are intended to work around the problem of UTF-8 characters being represented by multiple tokens with pdf $\LaTeX$, when a single grapheme is required from the start of a token list (for example, when obtaining initials).

These commands are experimental.

```
\datatool_get_first_grapheme:nN {<text>} <tl-var>
```

This obtains the first grapheme by using `\text_map_inline:n` and breaking after the first iteration. Note that this may not be a “letter” but may be a punctuation character. The `<text>` is purified before mapping.

```
\datatool_get_first_letter:nN {<text>} <tl-var>
```

Similar to the previous command but skips leading non-letters. This uses `\datatool_if_letter:nT` to test if the grapheme is a letter.

```
\datatool_if_letter:nTF {<grapheme>} {<true>} {<false>}
```

Tests if `<grapheme>` is a letter. Note that `<grapheme>` is expected to be a single character, which may be a multi-byte character. A grapheme is considered a letter if the first token of `<grapheme>` has a letter category code or if the upper and lowercase versions of `<grapheme>` are different. Note that not all letters have different upper and lowercase versions. If these are likely to be tested in this command, then use $\text{\XeTeX}$ or $\text{\LuaTeX}$ and ensure the character has the letter category code.

If the `<grapheme>` argument doesn’t contain a single character but instead contains a mixture of letters and punctuation, then `\datatool_if_letter:nTF` will do `<true>`. Ensure that `<grapheme>` is a single character stripped of all commands and braces.

For example, with $\text{\XeTeX}$ and $\text{\LuaTeX}$ the character “Á” is considered a single token and has the category code 11 (letter), but with $\text{\pdfTeX}$ and UTF-8 “Á” consists of two tokens where the first token has category code 13. However `\text_lowercase:n` is capable of converting “Á” to “á” and `\text_uppercase:n` is capable of converting “á” to “Á”. So if `<grapheme>` (which should already have been expanded and purified) has different uppercase and lowercase versions, it can be considered a letter.

The regular expression class `[ :alpha: ]` only covers ASCII letters. Recall also from Example 10 that ASCII control codes or non-letter characters with the category code set to “letter” were used to influence sorting. Since `\datatool_if_letter:nTF` was provided to assist with obtaining letter groups from sort values, it needs to take this into account. If this behaviour is inappropriate for your use, then use more appropriate commands provided by $\text{\LaTeX}$, such as the regular expression matching commands.

2.9. Comma-Separated Lists

$\LaTeX$ 3 provides commands for processing CSV lists. See §1.2.2 for an example.

The *datatool-base* package provides some commands that take a CSV list as the argument, such as `\dtladdall` (see §2.5.1), `\DTLaddall` (see §2.5.2) and `\DTLifinlist` (see §2.4.1.2). Unless otherwise stated, the argument may also be a command whose definition is a CSV list, but note that the argument must be exactly one token (the command) with no leading spaces or trailing tokens.

This behaviour is new to v3.0. Older versions would expand the first element in the list for some commands but not others.

Example 51 searches for an element in a list of four elements:

51

```
`duck' in `ant,duck,goose,zebra'?
\DTLifinlist{duck}{ant,duck,goose,zebra}{true}{false}
.
```

The above is equivalent to:

```
\newcommand{\mylist}{ant,duck,goose,zebra}
`duck' in ` \mylist'?
\DTLifinlist{duck}{\mylist}{true}{false}.
```

However, the following searches a list of one element where the sole element consists of two tokens (a space and the command `\mylist`):

```
\newcommand{\mylist}{ant,duck,goose,zebra}
`duck' in ` \mylist'?
\DTLifinlist{duck}{ \mylist}{true}{false}
```

The following searches a list of two elements, where the first element is `\mylist` and the second element is “zebu”:

```
\newcommand{\mylist}{ant,duck,goose,zebra}
`duck' in ` \mylist,zebu'?
\DTLifinlist{duck}{\mylist,zebu}{true}{false}.
```

Example 51: CSV List Argument Expansion

‘duck’ in ‘ant,duck,goose,zebra’? true.
 ‘duck’ in ‘ant,duck,goose,zebra’ (single token)? true.
 ‘duck’ in ‘ ant,duck,goose,zebra’ (one element, two tokens)? false
 ‘duck’ in ‘ant,duck,goose,zebra,zebu’ (two elements)? false.

There are two options, `skip-empty` and `trim`, that determine how to split the elements in the CSV list. These apply to most CSV list arguments, such as for `\DTLaddall` and `\DTLifinlist`, but not for commands described in §2.5.1 like `\dtladdall`. These settings also don’t apply to `<key>=<value>` comma-separated list options. Package options and options provided in `\DTLsetup` are always trimmed and skip empty elements.

The `datatool-base` package automatically loads `etoolbox` so you can use the `etoolbox` commands, such as `\forcsvlist`, to iterate over CSV lists. For examples, see [Iterating Over a Comma-Separated List](#).³

2.9.1. List Settings

The options described in this section govern the CSV list settings. They may be passed in the value of the `lists` option. For example:

```
\DTLsetup{
  lists={
    trim={false},
    skip-empty={false}
  }
}
```

`skip-empty=<boolean>`

initial: **true**

If true, empty elements will be skipped when parsing a CSV list.

`trim=<boolean>`

initial: **true**

If true, leading and trailing spaces will be trimmed from elements when parsing a CSV list.

`sort-reverse=<boolean>`

initial: **false**

If true, `\DTLsortwordlist` and `\dtlsortlist` will perform a reverse sort.

³dickimaw-books.com/latex/admin/html/doccsvlist.shtml


`sort-datum=<boolean>`

initial: **false**

If this conditional is true, then `\DTLsortwordlist` will parse each element when it uses the handler macro at the start, storing each element as a datum item, and will use numerical ordering for numeric data types. Integers and decimals will be numerically compared against each other, but the string comparison will be used if they are compared against another data type (including currency). Currency elements will be numerically compared if they have the same currency symbol, otherwise the string comparison will be used.


`and=<value>`

initial: **word**

Determines how `\DTLlistand` should expand. The *<value>* may be: `word` (expand to `\DTLlandname`) or `symbol` (expand to `\&`).



If there is no language support, `\DTLlandname` will expand to `\&` in which case `and=word` won't produce any noticeable effect.

2.9.2. Formatting Lists


`\DTLformatlist{<list>}`

modifier: *****

Formats the CSV list supplied in the argument *<list>*. The unstarred version adds grouping, the starred version doesn't. Each item in the list is formatted with:


`\DTLlistformatitem{<item>}`

This simply expands to *<item>* by default. Each pair of elements, except for the final pair are separated with:


`\DTLlistformatsep{<item>}`

initial: **,**

This expands to **,** (comma followed by a space) by default. The final two elements are separated with:


`\DTLlistformatlastsep`

This expands to `\DTLlistand\space` by default.

`\DTLlistformatoxford`*initial: empty*

If there are more than two items in the list, `\DTLlistformatlastsep` will be preceded by `\DTLlistformatoxford` which does nothing by default. If you want an Oxford comma then redefine `\DTLlistformatoxford` to a comma:

```
\renewcommand{\DTLlistformatoxford}{, }
```

`\DTLlistand`

Expands either to `\DTLandname` or to `\&`, depending on the `and` option in the `lists` setting.

`\DTLandname`
*initial: varies
language-sensitive*

This is initially defined to expand to `\andname` if that command was defined when `datatool-base` was loaded otherwise to `\&`. However `\DTLandname` is redefined by localisation support to expand to the appropriate word.

Example 52 redefines `\DTLlistformatitem` to render each item in italic:

52

```
\renewcommand{\DTLlistformatitem}[1]{\emph{#1}}
One: \DTLformatlist{elephant}.

Two: \DTLformatlist{elephant,ant}.

Three: \DTLformatlist{elephant,ant,zebra}.

Four: \DTLformatlist{elephant,ant,zebra,duck}.

\renewcommand{\DTLlistformatoxford}{, }
Oxford comma:
\DTLformatlist{elephant,ant,zebra,duck}.

Omit empty elements and leading/trailing spaces:
\DTLformatlist{elephant , ant, , duck}.

\DTLsetup{
  lists={
```

```

    trim=false,
    skip-empty=false
  }
}

```

Retain empty elements and leading/trailing spaces:
`\DTLformatlist{elephant , ant,,duck}.`

Example 52: Formatting CSV Lists

One: *elephant*.
 Two: *elephant & ant*.
 Three: *elephant, ant & zebra*.
 Four: *elephant, ant, zebra & duck*.
 Oxford comma: *elephant, ant, zebra, & duck*.
 Omit empty elements and leading/trailing spaces: *elephant, ant, & duck*.
 Retain empty elements and leading/trailing spaces: *elephant , ant, , & duck*.

2.9.3. List Elements

```
\DTLlistelement{<list>}{<idx>}
```

Does the *<idx>*th element in the list, where indexing starts with 1 for the first element.

```
\DTLfetchlistelement{<list>}{<idx>}{<cs>}
```

Fetches the *<idx>*th element in the list and defines the command *<cs>* to that element value.

```
\DTLnumitemsinlist{<list>}{<cs>}
```

Counts the number of elements in the list and defines the command *<cs>* to that number.

Example 53 uses the above commands for a three-element list, where the second element contains a comma.

53

```

\newcommand{\mylist}{ant,{bee, wasp and hornet},fly}
List: \DTLformatlist{\mylist}.

```

```
Number of elements: \DTLnumitemsinlist{\mylist}
```

```
{\total}\total.
```

```
Second element: \DTLlistelement{\mylist}{2}.
```

```
Fetch third element:
```

```
\DTLfetchlistelement{\mylist}{3}{\myelem}\myelem.
```

For comparison, the closest $\text{\LaTeX3}$ code is:

```
\LaTeX3 List:
```

```
\ExplSyntaxOn
```

```
\clist_set:NV \l_tmpa_clist \mylist
```

```
\clist_use:Nnnn \l_tmpa_clist { ~ and ~ } { , ~ }  
{ , ~ and ~ }
```

```
\ExplSyntaxOff
```

```
Number of elements:
```

```
\ExplSyntaxOn
```

```
\clist_count:N \l_tmpa_clist .
```

```
\ExplSyntaxOff
```

```
Second element:
```

```
\ExplSyntaxOn
```

```
\clist_item:Nn \l_tmpa_clist { 2 } .
```

```
\ExplSyntaxOff
```

```
Fetch third element:
```

```
\ExplSyntaxOn
```

```
\tl_set:Ne \l_tmpa_tl
```

```
{ \clist_item:Nn \l_tmpa_clist { 3 } }
```

```
\l_tmpa_tl .
```

```
\ExplSyntaxOff
```

 Example 53: Elements of a CSV List


List: ant, bee, wasp and hornet & fly.
 Number of elements: 3.
 Second element: bee, wasp and hornet.
 Fetch third element: fly.
 $\LaTeX$ List: ant, bee, wasp and hornet, and fly
 Number of elements: 3.
 Second element: bee, wasp and hornet.
 Fetch third element: fly.

2.9.4. Adding to Lists

The *datatool-base* package automatically loads *etoolbox* so you can use commands provided by that package to prepend or append to a command definition, such as `\preto` and `\appto`. Remember to include the comma separator.



If you have more complex requirements, consider using $\LaTeX$ commands (see §1.2.2 for an example).



```
\dtlinsertinto{<element>}{<sorted-list cs>}{<criteria cs>}
```

Globally inserts `<element>` into a sorted list (provided in the definition of the command `<sorted-list cs>`) according to the comparison macro `<criteria cs>`, which should have the same syntax as that for `\dtlsortlist`. Predefined comparison commands are described in §2.9.5.1.

Example 54 constructs a list, typesetting the contents with `\DTLformatlist` after each modification:

54



```
\newcommand{\mylist}{ant,bee}
Original list: \DTLformatlist{\mylist}.

\appto\mylist{,zebra}
Item appended: \DTLformatlist{\mylist}.

\preto\mylist{aardvark,}
Item prepended: \DTLformatlist{\mylist}.
```

```
\dtlinsertinto{duck}{\mylist}{\dtlcompare}
Item inserted: \DTLformatlist{\mylist}.
```

↑ Example 54: Appending, Prepending and Inserting List Elements



Original list: ant & bee.
 Item appended: ant, bee & zebra.
 Item prepended: aardvark, ant, bee & zebra.
 Item inserted: aardvark, ant, bee, duck & zebra.

2.9.5. Sorting Lists

There are two commands provided by *datatool-base* for sorting a CSV list. Both take a command as the first argument whose definition should be the CSV list. On completion this command will be (locally) redefined to expand to the ordered list. The second argument is also a command but its behaviour and syntax is different.

The first (and older) command is `\dtlsortlist`, which requires a comparison macro. This macro is repeatedly used to compare pairs of elements of the list throughout the sorting process.

The second command is `\DTLsortwordlist`, which requires a handler macro that is used to convert each element of the list into a byte sequence before sorting. The byte sequences are then compared throughout the sorting process using a simple character code comparison. `\DTLsortwordlist` is therefore more efficient, particularly if any localisation support is provided.

```
\dtlsortlist{<list-cs>}{<criteria cs>}
```

Sorts a CSV list according to the comparison command `<criteria cs>`. Note that `<list-cs>` must be a command not an explicit list. After the function has completed, `<list-cs>` will expand to the sorted list. The comparison command compares two list elements `<A>` and `<B>` and must have the syntax:

```
<criteria cs>{<reg>}{<A>}{<B>}
```

where `<reg>` is a count register. If `<A>` is deemed to be less than `<B>` then `<criteria cs>` should set `<reg>` to `-1`, if `<A>` is deemed to be greater than `<B>` then `<reg>` should be set to `+1` and if `<A>` and `<B>` are deemed equal then `<reg>` should be set to `0`. Predefined comparison commands are described in §2.9.5.1.

```
\DTLsortwordlist{<list-cs>}{<handler-cs>}
```


2. Base Commands (*datatool-base package*)

This command is an alternative to `\dtlsortlist` that has a handler macro for converting the original string value of each list element into a byte sequence. The handler macro should have the syntax:

```
<handler-cs> { <actual> } { <tl> }
```

where `<actual>` is the original value and `<tl>` is a token list control sequence in which to store the byte sequence. Predefined handlers are listed in §2.9.5.2.

The advantage with `\DTLsortwordlist` over `\dtlsortlist` is that with `\DTLsortwordlist` all the sort values are processed once at the start whereas with `\dtlsortlist` the values are repeatedly processed every time the comparison function is used. The `lists` option `sort-datum` only has an effect with `\DTLsortwordlist`. The `sort-reverse` option, which reverses the sort, governs both `\DTLsortwordlist` and `\dtlsortlist`.



Both `\dtlsortlist` and `\DTLsortwordlist` change the definition of `<list-cs>` so that it expands to the sorted list on completion. However, with `\dtlsortlist` the resulting `<list-cs>` definition is just a comma-separated list of ordered items from the original CSV list, but with `\DTLsortwordlist` the resulting `<list-cs>` definition is a comma-separated list of *sorted elements*.

A sorted element is in the form:

```
<sorted-marker-cs> { <actual> } { <sort value> } { <letter group> }
```

where `<actual>` is the original item or the datum item if the `sort-datum` option is true, `<sort value>` is the sort value obtained by the handler function (regardless of the data type), and `<letter group>` is the letter group identifier obtained from the `<sort value>` via:



```
\DTLassignlettergroup { <actual> } { <sort value> } { <cs> }
```

The letter group is assigned as follows:

1. If the `<actual>` part is a datum item then the letter group is assigned according to the data type as follows:

Integer or Decimal The letter group is considered a numeric group so `<cs>` is defined as `\dtlnumbergroup { <num> }` where `<num>` is the numeric value. Before encapsulating with `\dtlnumbergroup` a hook may be used to alter the `<num>` argument.



```
\DTLPreProcessIntegerGroup { <cs> } { <actual> }
```

2. Base Commands (*datatool-base package*)

This is used when $\langle num \rangle$ has been identified as an integer.

```
\DTLPProcessDecimalGroup{ $\langle cs \rangle$ }{ $\langle actual \rangle$ }
```

This is used when $\langle num \rangle$ has been identified as a decimal. In both cases, the $\langle cs \rangle$ argument is a command that expands to the numeric value and $\langle actual \rangle$ is the original value passed to `\DTLassignlettergroup`. These hooks do nothing by default.

Currency The letter group is considered a currency group so $\langle cs \rangle$ is defined as `\dtlcurrencygroup{ $\langle sym \rangle$ }{ $\langle num \rangle$ }` where $\langle sym \rangle$ is the currency symbol and $\langle num \rangle$ is the numeric value. Before encapsulating with `\dtlcurrencygroup` a hook may be used to alter the $\langle sym \rangle$ and $\langle num \rangle$ arguments.

```
\DTLPProcessCurrencyGroup{ $\langle sym-cs \rangle$ }{ $\langle num-cs \rangle$ }{ $\langle actual \rangle$ }
```

The $\langle sym-cs \rangle$ is the command that expands to the currency symbol, $\langle num-cs \rangle$ is the command that expands to the numeric value, and $\langle actual \rangle$ is the original value passed to `\DTLassignlettergroup`. This hook does nothing by default.

String If the $\langle sort value \rangle$ starts with a letter, $\langle cs \rangle$ is set to `\dtllettergroup{ $\langle initial \rangle$ }` where $\langle initial \rangle$ is obtained using `\DTLCurrentLocaleGetInitialLetter` otherwise $\langle cs \rangle$ is defined as `\dtlnonlettergroup{ $\langle grph \rangle$ }` where $\langle grph \rangle$ is the first grapheme in the $\langle sort value \rangle$. Before encapsulating with `\dtllettergroup` or `\dtlnonlettergroup`, hooks are available to alter the $\langle grph \rangle$.

```
\DTLPProcessLetterGroup{ $\langle cs \rangle$ }
```

This hook is used for a letter. This is defined to use `\DTLCurrentLocaleWordHandler`.

```
\DTLPProcessNonLetterGroup{ $\langle cs \rangle$ }
```

This hook is used for a non-letter and does nothing by default. In both cases, the $\langle cs \rangle$ is a command that expands to $\langle grph \rangle$.

2. Otherwise the letter group is assigned as for the string data type above.

2. Base Commands (*datatool-base package*)

If you want to iterate over a list that uses a handler function on each list element (such as `etoolbox's \forcsvlist`), then you can use the following commands on the element:

`\DTLsortedactual{<sorted element>}`

Expands to the actual element. You can also simply use `<sorted element>` directly. The difference is how expansion is applied. If the `sort-datum` option was true then the actual element will be a datum item. (That is, the format used in the expansion text of a datum control sequence.)

`\DTLsortedvalue{<sorted element>}`

Expands to the string sort value. For numeric data types (if parsing is on), this value would only have been used for comparing across different data types.

`\DTLsortedletter{<sorted element>}`

Expands to the letter group.

The above commands all expect a sorted element with its special markup as the argument. If you are using `\@for` to iterate over the sorted list, you will need to expand the loop control sequence first.

The following command is used by `\DTLsortwordlist` but not by `\dtlsortlist`.

`\dtlfallbackaction{<val1>}{<val2>}{<swap code>}{<no swap code>}`

If two sort values are identical, this command is used to determine whether or not the sort function should swap the elements. The arguments `<val1>` and `<val2>` are the original values or the datum items (not the identical byte sequences or numerical values). This command should expand to `<swap code>` if the elements should be swapped, otherwise it should expand to `<no swap code>`. The default definition uses the case-sensitive `\DTLifstringgt` to compare the original values.

For example, the following sorts a list and shows the result (with `\show`) in the transcript:

```
\newcommand{\mylist}{\pounds2, zebu, -  
.25, bee, \$5, ant, duck,  
+10, 4.56, \$23.10, 123}  
\dtlsortlist{\mylist}{\dtlcompare}  
\show\mylist
```

The transcript shows the following:

```
> \mylist=macro:
->\pounds
2, \$23.10, \$5, +10, -.25, 123, 4.56, ant, bee, duck, zebu.
```

This is a simple character code sort that produces a simple comma-separated list as the result. Compare the above with:

```
\newcommand{\mylist}{\pounds2, zebu, -
.25, bee, \$5, ant, duck,
+10, 4.56, \$23.10, 123}
\DTLsortwordlist{\mylist}{\DTLsortwordcasehandler}
\show\mylist
```

The result is now more complex. I've added line breaks for clarity and replaced the private command for the sort element markup with `<S-cs>` for compactness:

```
> \mylist=macro:
-><S-cs>{\$23.10}{\$23.10}{\dtlnonlettergroup{}} ,
<S-cs>{\$5}{\$5}{\dtlnonlettergroup{}} ,
<S-cs>{+10}{+10}{\dtlnonlettergroup{+}} ,
<S-cs>{-.25}{-.25}{\dtlnonlettergroup{-}} ,
<S-cs>{123}{123}{\dtlnonlettergroup{1}} ,
<S-cs>{\pounds 2}{2}{\dtlnonlettergroup{2}} ,
<S-cs>{4.56}{4.56}{\dtlnonlettergroup{4}} ,
<S-cs>{ant}{ant}{\dtllettergroup{a}} ,
<S-cs>{bee}{bee}{\dtllettergroup{b}} ,
<S-cs>{duck}{duck}{\dtllettergroup{d}} ,
<S-cs>{zebu}{zebu}{\dtllettergroup{z}} .
```

This produces a slightly different order: \$23.10, \$5, +10, -.25, 123, £2, 4.56, ant, bee, duck, zebu. The `\pounds` command is stripped by the handler as it is unable to expand to just text. Whereas the `\$` command is converted to the detokenized `$` by the sort hook (see §2.9.5.3). Note that the currency and numbers have all been assigned to the non-letter group.



A different result can occur if localisation support is provided (see §2.3).

The closest match to the case-sensitive `\dtlcompare` is `\DTLsortwordcasehandler` (used above). The closest match to the case-insensitive `\dtlcompare` is `\DTLsortwordhandler`. If the above example was switched to `\DTLsortlettercasehandler` or `\DTLsortletterhandler`, the hyphen/minus character `-` would be stripped from `-.25`, resulting in a sort value of `.25`.

2. Base Commands (datatool-base package)

When using lexicographical ordering, there is a distinct difference between `-.25` (a string containing four characters) and `-0.25` (a string containing five characters) or between `+10` and `10`. A simple character code comparison will place `+10` before `-.25` (since the plus character “+” has a lower codepoint value than the hyphen/minus character “-”).

Switching the `sort-datum` option to true adds an extra level of complexity in the result, but creates a different order because the numbers can now be compared numerically:

```
\DTLsetup{lists={sort-datum={true}}}
\newcommand{\mylist}{\pounds2, zebu, -
.25, bee, \$5, ant, duck,
+10, 4.56, \$23.10, 123}
\DTLsortwordlist{\mylist}{\DTLsortwordcasehandler}
\show\mylist
```

I’ve added line breaks for clarity, replaced constants with their actual numeric values, and replaced the private commands for the sort element markup with `<S-cs>` and datum markup with `<D-cs>` for compactness:

```
> \mylist=macro:
-><S-cs>{<D-cs>{\$5}{5}{\$}{3}}{\$5}{\dtlcurrencygroup{\$}
{5}},
<S-cs>{<D-cs>{\$23.10}{23.10}{\$}{3}}{\$23.10}{\dtl-
currencygroup{\$}{23.10}},
<S-cs>{<D-cs>{-.25}{-0.25}}{-0.25}{\dtlnumbergroup{-
0.25}},
<S-cs>{<D-cs>{4.56}{4.56}}{4.56}{\dtlnumbergroup
{4.56}},
<S-cs>{<D-cs>{+10}{10}}{+10}{\dtlnumbergroup{10}},
<S-cs>{<D-cs>{123}{123}}{123}{\dtlnumbergroup{123}},
<S-cs>{<D-cs>{\pounds 2}{2}{\pounds}{3}}{2}{\dtlcurrency-
group{\pounds}{2}},
<S-cs>{<D-cs>{ant}}{ant}{\dtllettergroup{a}},
<S-cs>{<D-cs>{bee}}{bee}{\dtllettergroup{b}},
<S-cs>{<D-cs>{duck}}{duck}{\dtllettergroup{d}},
<S-cs>{<D-cs>{zebu}}{zebu}{\dtllettergroup{z}}.
```

This expands to a different order: \$5, \$23.10, -.25, 4.56, +10, 123, £2, ant, bee, duck, zebu. Note that the numeric values have been split into different sub-groups: currency and number. The dollar \$ currency is placed before the numbers because a string comparison is used between a currency numeric value and non-currency number. For example, 4.56 and 123 are compared numerically, so 4.56 is placed before 123, but \$23.10 and +10 are compared lexicographically.

The `\pounds2` item has been correctly parsed as currency, but the string sort value ends up as just 2 as a result of stripping `\pounds`. Since 123 isn’t currency but `\pounds2` is, the values are compared lexicographically instead of numerically, which means comparing the string

“123” with the string “2”. The best solution is to provide a local redefinition of `\pounds`:

```
\dtlSortWordCommands{\def\pounds{\pounds}}
```

This is done automatically by `datatool-GB.ldf` and other localisation files that support pound sterling currency (see §2.3.5). Adding localisation support, for example:

```
\usepackage[locales=en-GB]{datatool-base}
```

results in a different order: -.25, +10, \$5, \$23.10, £2, 4.56, 123, ant, bee, duck, zebu.

It’s important to remember the item markup if you need to extract information within a list loop.

If you have $\text{\LaTeX}$ 3 syntax enabled you can apply the changes made in the internal hook with:

```
\datatool_sort_preprocess:Nn <tl-var> {\<text>}
```

where `<tl-var>` is the token list variable in which to store the result. Note that this expands the `<text>` in the same way that the sort handlers do and may also convert to lowercase, depending on the current settings. This command may be used in a locale’s definition of `\DTLCurrentLocaleGetGroupString` if using the “actual” argument.

If you prefer to ignore the current convert to lowercase setting, you can instead use:

```
\datatool_sort_preprocess:NnN <tl-var> {\<text>} <bool-var>
```

The final argument should be a boolean (`\c_true_bool` or `\c_false_bool`) to indicate whether or not the token list should be converted to lowercase.

2.9.5.1. Comparison Commands

These comparison commands may be used with `\dtlsortlist`, `\dtlinsertinto`, and `\dtlsort`. For `\DTLsortwordlist` and `\DTLsortdata` handler functions, see §2.9.5.2. In each case, the syntax is:

```
<cs>{\<count-reg>}\<arg1>}\<arg2>}
```

where `<count-reg>` is a count register (integer variable). If `<arg1>` is deemed to be less than (comes before) `<arg2>` then `<cs>` will set `<count-reg>` to `-1`, if `<arg1>` is deemed to be greater than (comes after) `<arg2>` then `<count-reg>` is set to `+1` and if `<arg1>` and `<arg1>` are deemed equal then `<count-reg>` is set to `0`. Note that the handler’s notion of equality doesn’t necessarily mean the

two arguments are identical. For example, a case-insensitive comparison will consider “word” and “Word” as equal, and `\DTLnumcompare` will consider `\$1,234.50` and `1234.5` to be equal, since only the numerical values are compared.

`\dtlletterindexcompare{<count-reg>}{<string1>}{<string2>}`

Designed for case-insensitive letter order comparison, but a hook (see §2.9.5.3) is used to locally redefine certain commands to allow for adjustments in the compared strings. Spaces are stripped from the strings so, for example, “sea lion” will come after “sealant”. (Note that this isn’t quite analogous to `\DTLsortletterhandler` as that also discards hyphens.)

`\DTLsortwordhandler` is used to convert both strings to byte sequences, which are then compared. It’s therefore more efficient to use `\DTLsortwordlist` to avoid repeatedly converting the same strings to byte sequences.

`\dtlwordindexcompare{<count-reg>}{<string1>}{<string2>}`

As `\dtlletterindexcompare` but spaces aren’t stripped from the strings so, for example, “sea lion” will come before “sealant”.

Since both `\dtlletterindexcompare` and `\dtlwordindexcompare` use `\DTLsortwordhandler`, they are sensitive to the current language *provided that* a suitable language module has been installed (see §2.3 for more details and Example 61 for an example). This does not apply to the simple character code commands `\dtlcompare` and `\dtlicompare`.

`\dtlcompare{<count-reg>}{<string1>}{<string2>}`

This command is used internally by the unstarred `\DTLifstringlt`, `\DTLifstringeq` and `\DTLifstringgt` for a case-sensitive comparison. If you are using `\DTLsortwordlist`, the closest matching handler is `\DTLsortwordcasehandler`. However `\dtlcompare` has no localisation support and just performs a character code comparison.

`\dtlicompare{<count-reg>}{<string1>}{<string2>}`

This command is used internally by the starred `\DTLifstringlt*`, `\DTLifstringeq*` and `\DTLifstringgt*` for a case-insensitive comparison. If you are using `\DTLsortwordlist`, the closest matching handler is `\DTLsortwordhandler`. However `\dtlicompare` has no localisation support and just performs a character code comparison (after converting the strings to lowercase).



```
\DTLnumcompare{⟨count-reg⟩}{⟨num1⟩}{⟨num2⟩}
```

Compares $\langle num1 \rangle$ and $\langle num2 \rangle$ numerically, where the arguments are formatted numbers or datum control sequences. Unlike the numerical comparison commands in §2.4.1.3, this command ignores the `math` setting, and will use `!3int` or `!3fp` comparisons, depending on the data types. Any currency symbol (if present) will be ignored. If either $\langle num1 \rangle$ or $\langle num2 \rangle$ are not recognised as numerical values, the value will be treated as zero.

Options that govern comparison commands can be set within the `compare` setting value. For example:



```
\DTLsetup{ compare={expand-cs=true} }
```



```
expand-cs=⟨boolean⟩
```

initial: **false**

Both `\dtlcompare` and `\dtlicompare` (but not the other comparison commands) are governed by the `expand-cs` boolean option. When used with `\dtlcompare` or `\dtlicompare`: if true, $\langle string1 \rangle$ and $\langle string2 \rangle$ will be fully expanded and purified before comparison. If false, the following boolean option takes effect:



```
skip-cs=⟨boolean⟩
```

initial: **false**

When used with `\dtlcompare` or `\dtlicompare` where `expand-cs=false`: if `skip-cs=true`, any commands found in $\langle string1 \rangle$ or $\langle string2 \rangle$ will be replaced with the control code `0x0A` (newline). This means that a command is considered lexicographically smaller than punctuation, digits and letters. If false, all commands will be removed. This conditional has no effect if `expand-cs=true`.

2.9.5.2. \DTLsortwordlist Handlers

The following handler macros are provided for use with `\DTLsortwordlist` and may also be used as the `function` in `\DTLsortdata`. In each case, $\langle original \rangle$ is the original string and $\langle cs \rangle$ is a control sequence that will be defined to the resulting sort value (which will then be treated as a byte sequence).



```
\DTLsortwordhandler{⟨original⟩}{⟨cs⟩}
```

A case-insensitive word order handler. This expands $\langle original \rangle$ and converts it to lowercase, then applies `\DTLDefaultLocaleWordHandler` and purifies the result. This means that it's sensitive to the current language *provided that* a suitable language module has been installed (see §2.3 for more details and Example 61 for an example).



```
\DTLsortwordcasehandler{⟨original⟩}{⟨cs⟩}
```

A case-sensitive word order handler. This expands *⟨original⟩*, then applies `\DTLDefaultLocaleWordHandler` and purifies the result.



```
\DTLsortletterhandler{⟨original⟩}{⟨cs⟩}
```

A case-insensitive letter order handler. Similar to `\DTLsortwordhandler` but discards hyphens and spaces.



```
\DTLsortlettercasehandler{⟨original⟩}{⟨cs⟩}
```

A case-sensitive letter order handler. Similar to `\DTLsortwordcasehandler` but discards hyphens and spaces.

The above handler macros are simple wrapper functions that ensure the value is expanded and pre-processed (case conversion or stripping hyphens and spaces) and stored in *⟨cs⟩* before using the default word handler:



```
\DTLDefaultLocaleWordHandler{⟨cs⟩}
```

This uses `\DTLCurrentLocaleWordHandler{⟨cs⟩}` to convert *⟨cs⟩* to a byte sequence that ensures the locale's alphabetic ordering, and then appends `\datatoolctrlboundary` to *⟨cs⟩*. If you don't require the boundary marker, you can redefine this command to just use the current locale handler:



```
\renewcommand{\DTLDefaultLocaleWordHandler}[1]{%
  \DTLCurrentLocaleWordHandler{#1}%
}
```

2.9.5.3. Word Sort Hook

The hook used by `\dtlwordindexcompare` and `\dtlletterindexcompare` is also used at the start of `\DTLsortwordlist`. This means that the hook is applied only once with an instance of `\DTLsortwordlist`, but with `\dtlsortlist` the hook is applied each time a comparison is required by `\dtlwordindexcompare` or `\dtlletterindexcompare`. Therefore, if you want word or letter sorting, it's better to use the newer `\DTLsortwordlist` with `\DTLsortwordhandler` or `\DTLsortletterhandler`.

The commands changed by this hook are listed below.

`\dtltxorsort{\TeX}\{sort\}`

This normally expands to just its first argument but inside `\DTLsortwordlist` it expands to its second argument instead. You may recall from §2.6 that `\DTLdefcurrency` defines `\DTLcurr{ISO}` to use `\dtltxorsort` in its argument. This allows the currency symbol to expand to a string representation within `\DTLsortwordlist`, `\dtlwordindexcompare` or `\dtlletterindexcompare`.

`\datatoolasciistart`

Expands to nothing normally. The hook redefines this command to expand to the null character, which means that it will come before all other characters.

`\datatoolasciend`

Expands to nothing normally. The hook redefines this command to expand to the delete character (0x7F, the highest ASCII character).

`\datatoolctrlboundary`

Expands to nothing normally. The hook redefines this command to expand to the character 0x1F (the last control character before the space character). This command is automatically appended to the sort value by `\DTLDefaultLocaleWordHandler`.

`\datatoolpersoncomma`

Designed to indicate word inversion for a person, this command expands to `, \space` normally. The hook redefines this command to the character 0x1C. For example,

`Kunth\datatoolpersoncomma Donald E.`

`\datatoolplacecomma`

Designed to indicate a comma to clarify a place, this command expands to `, \space` normally. The hook redefines this command to the character 0x1D. For example:

`Paris\datatoolplacecomma Texas`

`\datatoolsubjectcomma`

Designed to indicate heading inversion (subjects, concepts and objects), this command expands to `, \space` normally. The hook redefines this command to the character 0x1E. For example:

New York\datatoolsubjectcomma population

There are two ways of dealing with parenthetical content.

`\datatoolparen{<text>}`

This command normally expands to `\space (<text>)`. The hook redefines `\datatoolparen` to expand to `\datatoolctrlboundary`, ignoring the argument. For example:

duck\datatoolparen{wildfowl}

This means that the parenthetical content is omitted in the comparison. Note that this will cause the following terms to be considered identical:

duck\datatoolparen{wildfowl}
duck\datatoolparen{cricket}

With `\DTLsortwordlist`, this is addressed by the default definition of `\dtlfall-backaction` which will compare the duplicate values using the original definition of `\datatoolparen`. Note that since `\DTLDefaultLocaleWordHandler` appends `\datatoolctrlboundary` to all values, “duck” will become “duck” followed by 0x1F and `duck\datatoolparen{<text>}` will become “duck” followed by 0x1F 0x1F. The first control character is inserted by `\datatoolparen` and the second by `\DTLDefaultLocaleWordHandler`. This means that the parenthetical entries will come after the word on its own, and the word on its own will come after the word followed by one of the comma markers.

An alternative method that can be used with `\dtlsortlist` is to mark where the parenthetical material starts:

`\datatoolparenstart`

Designed to indicate the start of parenthetical content, this command expands to `\space` normally. The parenthesis characters need to be added explicitly. The hook redefines this command to expand to the character 0x1F. For example:

duck\datatoolparenstart (cricket)

In this case, the parenthetical content is included in the comparison.



The commands `\datatoolctrlboundary` and `\datatoolparenstart` have the same definition within the hook but expand differently in normal text.

The use of control codes affects ordering. The normal space character has character code 0x20 and a comma has the character code 0x2C. This means that a comma would ordinarily be placed after a space character, whereas the low-end control codes come before space.



The control codes used in the above commands are all assigned the “other” category code (12) within the definition used by the hook. They are only intended for use in sorting, not for typesetting.

Following the guidelines of the Oxford Style Manual, when sorting terms that have identical pre-inversion parts, the following ordering is applied: people, places, subjects, no inversions, and parenthetical. This is achieved through the marker commands (see examples 57 & 58).



```
\datatoolSetCurrencySort
```

Locally redefines common currency commands to their string alternatives. For example, `\pounds` is set to `\l_datatool_pound_str`.

The hook also redefines `\nobreakspace` and `\_` to `\space`, `\TeX` and `\LaTeX` are locally redefined to simply expand to “TeX” and “LaTeX”, respectively, and the following commands are locally redefined to expand to the corresponding detokenized character: `\$` (\$), `\_` (_), `\#` (#), `\%` (%), and `\&` (&).

Additional code can be added to the hook with:



```
\dtlSortWordCommands{<code>}
```

This globally adds `<code>` to the hook. Within `<code>` the `@` character has the letter category code, which means that you can use internal `@`-commands in `<code>` (see Example 62). If you need to use control codes, make sure that you first (locally) change the category code to “other” before using them in the hook.

2.9.5.4. CSV List Sorting Examples

The following examples demonstrate the different sorting methods.

2.9.5.4.1. Sorting with `\dtlsortlist`

Examples 55 & 56 sort the list: sea, sea lion, Sealyham, seal, sealant, sealing wax, which is defined as:

```
\newcommand{\mylist}{sea, sea lion, Sealyham, seal,
sealant, sealing wax}
```

The sorted order varies depending on whether or not the sort method is case-sensitive or strips spaces. (See Example 59 for a UTF-8 example.)

Example 55 uses `\dtlsortlist` with `\dtlcompare` (case-sensitive) and then with `\dtlicompare` (case-insensitive). Note that case-sensitive sorting places the uppercase characters before the lowercase characters (so Sealyham comes first with `\dtlcompare` and last with `\dtlicompare`). In both cases, “sea lion” is placed before “sealing wax” (that is, the space is significant).

55

```
\dtlsortlist{\mylist}{\dtlcompare}
Case-sensitive: \DTLformatlist{\mylist}.

\dtlsortlist{\mylist}{\dtlicompare}
Case-insensitive: \DTLformatlist{\mylist}.
```

↑ Example 55: Sorting Lists with `\dtlsortlist` (Case vs No Case)



Case-sensitive: Sealyham, sea, sea lion, seal, sealant & sealing wax.

Case-insensitive: sea, sea lion, seal, sealant, sealing wax & Sealyham.

Example 56 uses `\dtlletterindexcompare` (letter sort) and `\dtlwordindexcompare` (word sort) instead. The order for both is case-insensitive (so Sealyham comes last), but the letter order method puts “sea lion” after “sealing wax” (spaces are discarded, “o” comes after “n”) whereas the word order method puts “sea lion” before “seal” (the space is significant, “l” comes after space).

56

↑ Example 56: Sorting Lists with `\dtlsortlist` (Letter vs Word)



Letter: sea, seal, sealant, sealing wax, sea lion & Sealyham.

Word: sea, sea lion, seal, sealant, sealing wax & Sealyham.

2.9.5.4.2. Sort Markers

Example 57 demonstrates the use of the comma and parenthetical markers. The simple character code commands `\dtlcompare` and `\dtlicompare` are governed by the `skip-cs` and `expand-cs` options within the `compare` setting. If both options are false, any commands found in the sort value, including the marker commands, are replaced with the character code 0x0A. If `skip-cs=true` (and `expand-cs=false`) then the marker commands will

57

be stripped. If `expand-cs=true`, then the `skip-cs` setting is ignored and the marker commands will expand to their usual meaning (not the meaning provided by the word sort hook). I've used `expand-cs=true` to ensure the marker commands are expanded.

The example doesn't use `\dtlcompare`, which would put the elements starting with "Duck" at the beginning, but would otherwise use the same relative order as `\dtlcompare`.

The word and letter functions use the word sort hook (see §2.9.5.3), which means that the special marker functions expand to low ASCII control characters, which means that they are placed before ordinary punctuation and letters. The example list is defined as:

```
\newcommand{\mylist}{duckling,
  Duck\datatoolplacecomma Mallard County,
  Duck\datatoolpersoncomma Robbie,
  Duck\datatoolsubjectcomma Anatomy of a,
  duck\datatoolparenstart (cricket),
  duck\datatoolparen{verb},
  {Duck, Duck, Goose},
  duck soup, duck, duck and dive
}
```

Note that there is one element that has normal commas ("Duck, Duck, Goose"). This requires braces to hide the commas from the list parser. The other commas are hidden in the marker commands: `\datatoolpersoncomma` (to signify *<surname>*, *<forename>*), `\datatoolsubjectcomma` (heading inversion), and `\datatoolplacecomma` (to signify a place). The above example uses two different ways of marking parenthetical material, `\datatoolparenstart` and `\datatoolparen`. This affects sorting.

Since the lists have long elements and elements with commas, I've used the `multicol` package to arrange them in columns and changed the separators used by `\DTLformatlist` to insert line breaks. This means that the only commas shown are those within the elements. I've used grouping to localise the changes, which ensures that each sort starts from the same list.

```
\renewcommand{\DTLlistformatsep}{\newline}
\renewcommand{\DTLlistformatlastsep}{\newline}
\DTLsetup{compare={expand-cs=true}}
\begin{multicols}{3}
{\dtlsortlist{\mylist}{\dtlcompare}
Case-insensitive:\newline
\DTLformatlist{\mylist}.}

{\dtlsortlist{\mylist}{\dtlwordindexcompare}
Word sort:\newline
\DTLformatlist{\mylist}.}
```

```
\dtlsortlist{\mylist}{\dtlletterindexcompare}
Letter sort:\newline
\DTLformatlist{\mylist}.
\end{multicols}
```

The simple case-insensitive comparison (`\dtlcompare`) doesn't recognise any difference between explicit commas and the commas within the marker commands, so “Duck, Duck, Goose” is placed between “Duck, Anatomy of a” and “Duck, Mallard County”. Sorting by character code orders the space character 0x20 before the comma character 0x2C, so “duck soup” is placed before “Duck, Anatomy of a”.

Note the difference between using `\datatoolparen` (which inserts 0x1F and discards its argument) and `\datatoolparenstart` (which inserts 0x1F). This means that “(verb)” is omitted from the comparison but “(cricket)” isn't, so “duck (verb)” ends up before “duck (cricket)”.

↑ Example 57: Sorting Lists with `\dtlsortlist` (comma and parenthetical markers)

Case-insensitive:	Word sort:	Letter sort:
duck	duck	duck
duck (cricket)	Duck, Robbie	Duck, Robbie
duck (verb)	Duck, Mallard County	Duck, Mallard County
duck and dive	Duck, Anatomy of a	Duck, Anatomy of a
duck soup	duck (verb)	duck (verb)
Duck, Anatomy of a	duck (cricket)	duck (cricket)
Duck, Duck, Goose	duck and dive	Duck, Duck, Goose
Duck, Mallard County	duck soup	duck and dive
Duck, Robbie	Duck, Duck, Goose	duckling
duckling.	duckling.	duck soup.

Example 58 adapts Example 57 to use `\DTLsortwordlist`, but there's no equivalent to the `\dtlcompare` handler:

```
{\DTLsortwordlist{\mylist}{\DTLsortwordhandler}
Word sort:\newline
\DTLformatlist{\mylist}.}

{\DTLsortwordlist{\mylist}{\DTLsortletterhandler}
Letter sort:\newline
\DTLformatlist{\mylist}.}
```

I've also supplied my own custom handler that first strips explicit commas and then behaves like `\DTLsortletterhandler`:

```

\ExplSyntaxOn
\NewDocumentCommand \mycustomhandler { m m }
{
  \tl_set:Nn #2 { #1 }
  \regex_replace_all:nnN { , } { } #2
  \DTLsortletterhandler { #2 } #2
}
\ExplSyntaxOff

\DTLsortwordlist{\mylist}{\mycustomhandler}
Custom sort:\newline
\DTLformatlist{\mylist}.

```

Note that the position of “duck” has changed. This is because of the boundary marker that’s appended to all the values. The boundary marker control code is now compared with the comma and parentheses marker control codes.

↑ Example 58: Sorting Lists with `\DTLsortwordlist` (comma and parenthetical markers)

Word sort:	Letter sort:	Custom sort:
Duck, Robbie	Duck, Robbie	Duck, Robbie
Duck, Mallard County	Duck, Mallard County	Duck, Mallard County
Duck, Anatomy of a	Duck, Anatomy of a	Duck, Anatomy of a
duck	duck	duck
duck (verb)	duck (verb)	duck (verb)
duck (cricket)	duck (cricket)	duck (cricket)
duck and dive	Duck, Duck, Goose	duck and dive
duck soup	duck and dive	Duck, Duck, Goose
Duck, Duck, Goose	duckling	duckling
duckling.	duck soup.	duck soup.

2.9.5.4.3. UTF-8

Results with `\dtlcompare` or `\dtlicompare` for UTF-8 values are less satisfactory. The list for examples 59, 60 & 61 is defined as:

```

\newcommand{\mylist}
{elk, Æthelstan, Arnulf, elf, résumé,
 Óslác, élan, Aeolus, resume, elephant, zygote, élite,
 Valkyrie, Zulu, elbow, Adelolf, rose}

```


X_YLaTeX and LuaLaTeX both natively support UTF-8. Modern pdfLaTeX now defaults to UTF-8 but if you want to use inputenc remember to load it before *datatool-base*:

```
\usepackage[utf8]{inputenc}
\usepackage{datatool-base}
```

Example 59 sorts the lists with `\dtlsortlist` as for the earlier Example 55:

59

```
\dtlsortlist{\mylist}{\dtlcompare}
Case-sensitive: \DTLformatlist{\mylist}.

\dtlsortlist{\mylist}{\dtlicompare}
Case-insensitive: \DTLformatlist{\mylist}.
```

Note that the UTF-8 characters are all placed after the Basic Latin characters, so “Æthelstan” is placed after “zygote” for both the case-sensitive and case-insensitive comparators. However, “Æthelstan” and “Óslác” come before “élan” and “élite” for the case-sensitive sort. Whereas for the case-insensitive sort, “Óslác” comes after “élite”.

↑ Example 59: Sorting Lists with `\dtlsortlist` and UTF-8



Case-sensitive: Adelolf, Aeolus, Arnulf, Valkyrie, Zulu, elbow, elephant, elf, elk, resume, rose, résumé, zygote, Æthelstan, Óslác, élan & élite.
Case-insensitive: Adelolf, Aeolus, Arnulf, elbow, elephant, elf, elk, resume, rose, résumé, Valkyrie, Zulu, zygote, Æthelstan, élan, élite & Óslác.

Note that because `\dtlletterindexcompare` and `\dtlwordindexcompare` both internally use `\DTLsortwordhandler`, they are able to handle UTF-8 in the same way as using `\DTLsortwordlist` with the handler macros described in §2.9.5.2. However, it's more efficient to use `\DTLsortwordlist` to avoid repeatedly pre-processing the values.

Example 60 adapts Example 59 to use `\DTLsortwordlist` instead of `\dtlsortlist`.

60

```
\DTLsortwordlist{\mylist}{\DTLsortlettercasehandler}
Case-sensitive sort: \DTLformatlist{\mylist}.

\DTLsortwordlist{\mylist}{\DTLsortletterhandler}
Case-insensitive sort: \DTLformatlist{\mylist}.
```

Note that the UTF-8 characters are still listed after the Basic Latin characters when there is no

localisation support. So the result is the same as before.

↑ Example 60: Sorting Lists with `\DTLsortwordlist` and UTF-8 and No Localisation Support

Case-sensitive: Adelolf, Aeolus, Arnulf, Valkyrie, Zulu, elbow, elephant, elf, elk, resume, rose, résumé, zygote, Æthelstan, Óslác, élan & élite.

Case-insensitive: Adelolf, Aeolus, Arnulf, elbow, elephant, elf, elk, resume, rose, résumé, Valkyrie, Zulu, zygote, Æthelstan, élan, élite & Óslác.

Example 61 is a minor modification of Example 59, but it requires `datatool-english` to be installed. This can then be loaded via the `tracklang` interface (for example, with the `locales` option) which ensures that the localisation support provided by `datatool-english` is used. Remember to include the region if currency support is also required. For example:

```
\documentclass[en-GB]{article}
```

Or:

```
\usepackage[locales=en-GB]{datatool-base}
```

This produces a better order because the `datatool-english-utf8.ldf` file substitutes common UTF-8 characters for the closest ASCII equivalent (“Æ” for “AE”, “Ó” for “O”, “á” for “a”, and “é” for “e”). This means that “Æthelstan” is now at the start before “Adelolf” (because “AE” comes before “Ad”) for the case-sensitive sort but between “Aeolus” and “Arnulf” for the case-insensitive sort.

↑ Example 61: Sorting Lists with `\DTLsortwordlist` and UTF-8 and Localisation Support

Case-sensitive: Æthelstan, Adelolf, Aeolus, Arnulf, Óslác, Valkyrie, Zulu, élan, elbow, elephant, elf, élite, elk, resume, résumé, rose and zygote.

Case-insensitive: Adelolf, Aeolus, Æthelstan, Arnulf, élan, elbow, elephant, elf, élite, elk, Óslác, resume, résumé, rose, Valkyrie, Zulu and zygote.

Note that with `datatool-english-utf8.ldf`, “résumé” is converted into “resume”, which means “résumé” has an identical sort value to “resume”. With `\DTLsortwordlist`, the relative ordering of these duplicates is then determined by `\dtlfallbackaction`, which by default compares the original values with `\DTLifstringgt`. Since the unstarred `\DTLifstringgt` internally uses `\dtlcompare` without localisation, this places “resume” before “résumé”.

Remember that `\dtlsortlist` doesn't use `\dtlfallbackaction` so “résumé” and “resume” are deemed equivalent with `\dtlwordindexcompare` `\dtlletterindexcompare` so the result with `\dtlsortlist` would retain their original relative order.

2.9.5.4.4. Roman Numerals

Example 62 has a list of names with Roman numerals, such as is used in the names of monarchs. In the first case, the numerals are explicitly included in the list. In the second case a command is provided that has a different definition in the hook.

62

```
\newcommand{\mylist}{John XVI, John VI,
John XIX, John IX, John IV, John VII, John V}
\DTLsortwordlist{\mylist}{\DTLsortwordhandler}
\DTLformatlist\mylist.

\newcommand{\Ord}[1]{\MakeUppercase{\romannumeral #1}}
\dtlSortWordCommands{\renewcommand\Ord[1]{\two@digits{#1}}
}
\renewcommand\mylist{John \Ord{16}, John \Ord{6},
John \Ord{19}, John \Ord{9}, John \Ord{4}, John \Ord{7},
John \Ord{5}}
\DTLsortwordlist{\mylist}{\DTLsortwordhandler}
\DTLformatlist\mylist.
```

Ordinarily this custom `\Ord` command will convert its numeric argument into an uppercase Roman numeral (for example, `\Ord{16}` would expand to “XVI”), but when sorting it expands to a number instead (for example, `\Ord{16}` would expand to “16”).

Note the use of `\two@digits` that zero-pads the number to ensure that it has at least two digits (for example, `\Ord{6}` would expand to “06”). This is because lexicographic ordering rather than numeric ordering is used (otherwise “16” would come before “6”). If large ordinals are expected then extra padding would be required (which can be obtained with `\dtlpad-leadingzeros`).

↑ Example 62: Sort Word Hook (Roman Numerals)



John IV, John IX, John V, John VI, John VII, John XIX & John XVI.
John IV, John V, John VI, John VII, John IX, John XVI & John XIX.

3. Databases (datatool package)

The datatool package provides a means of creating and loading databases. Once a database has been created (either with supplied document commands or by parsing an external file), it is possible to iterate through each row of data, to make it easier to perform repetitive actions, such as mail merging.



Whilst $\text{\TeX}$ is an excellent typesetting language, it is not designed as a database management system, and attempting to use it as such is like trying to fasten a screw with a knife instead of a screwdriver: it can be done, but requires great care and is more time consuming. Version 2.0 of the datatool package switched to a completely different method of storing the data to previous versions.^a Version 3.0 has switched to using $\text{\LaTeX}$ 3 commands internally for some of the database functions. As a result, the code is much more efficient. However, large databases and complex operations will still slow the time taken to process your document. Therefore, if you can, it is better to do the complex operations using whatever system created the data in the first place.

^aMany thanks to Morten Høgholm for providing the new code.

Some advanced commands for accessing database information are described in §3.16, but using $\text{\TeX}$ is nowhere near as efficient as, say, using a SQL database, so don't expect too much from this package.

I've written a Java helper application to accompany datatool called `datatooltk`. The installer `datatooltk-installer.jar`¹ is available on CTAN.² The application will allow you to edit DTLTEX (see §3.15.1.2) and DBTEX (see §3.15.1.3) files saved using `\DTLwrite` in a graphical interface or import data from a SQL database, a CSV file or a probsoln dataset.



```
\usepackage[\langle options \rangle]{datatool}
```

The datatool package automatically loads the `datatool-base` package, so all commands provided by `datatool-base` are available. The commands provided by `datatool` relate to databases.

The supplementary packages `dataplot`, `datapie`, `databar`, `databib` and `datagidx` automatically load `datatool` and provide additional commands that act on databases. In the case of `databib` and `datagidx`, the databases have a specific structure. The `dataplot`, `datapie` and `databar` packages provide commands to visually represent numeric data stored in a database.

¹mirrors.ctan.org/support/datatooltk/datatooltk-installer.jar

²ctan.org/pkg/datatooltk

3.1. Options

All options provided by `datatool-base` (see §2) may also be passed to `datatool`. Additionally, the following options are also available, some of which can only be passed as a package option, some of which can only be used in `\DTLsetup`, and some may be used in either context.

default-name=*<db-name>*

initial: **untitled**

May be used either as a package option or in `\DTLsetup`, this sets the default database name for commands where the name is optional (such as `\DTLaction` and `\DTLwrite`). Note that the argument is expanded when the option is set. For example:

```
\newcommand{\mydatacmd}{mydata}
\DTLsetup{default-name=\mydatacmd}
\renewcommand{\mydatacmd}{otherdata}
```

In the above, the default database name remains “mydata” after `\mydatacmd` is redefined.

delimiter=*<char>*

initial: **"**

This option may only be used as a package option and sets the delimiter used in CSV and TSV files. The value *<char>* must be a single token, and is used in the file to delimit a value which may contain the separator character to hide the separator from the parser.

After the package has loaded, you can use `\DTLsetdelimiter` to set the delimiter. Alternatively, you can use the `delimiter` setting within the optional argument of `\DTLread` or `\DTLwrite` or within the value of the `io` option in `\DTLsetup`. For example:

```
% Package option:
\usepackage[delimiter='{'}]{datatool}
% Change default:
\DTLsetdelimiter{|}
% Or:
\DTLsetup{io={delimiter={|}}}}
% Override the default just for this file:
\DTLread[format=csv,delimiter="{"}]{myfile}
```

new-value-expand=*<boolean>*

default: **true**; initial: **false**

This boolean option determines whether or not new values should be expanded before they are added to a database. This also includes data read from CSV and TSV files (see §3.15.1.1), and

DTLTEX files (see §3.15.1.2), but not DBTEX files (see §3.15.1.3).



With `new-value-expand=false`, you can still expand individual values using the `expand-value` or `expand-once-value` when adding an entry to a database with the `new entry` action.

If `new-value-expand=true`, protected expansion is applied to the value, otherwise no expansion is performed. For example:



```
\newcommand{\qt}[1]{` `#1'}
\DTLsetup{new-value-expand=true}
\DTLnewdbentry{mydata}{Name}{Zoë \qt{Stripes} Zebra}
```

In the above, the entry will be added to the database as Zoë ` `Stripes' ' Zebra. This means that if the definition of `\qt` is later changed, it won't affect this entry. In this particular case, it's better to have the default `new-value-expand=false` setting to prevent expansion (or use robust commands).

The `new-value-expand=true` option is useful if you are programmatically creating entries with placeholder commands, which need to be expanded. Example 63 demonstrates the difference:

63



```
\makeatletter
\DTLsetup{new-value-expand=false}
\DTLnewdb{test1}
\DTLaddcolumnwithheader{test1}{entry}
{Entry (Not Expanded)}
\@for\myentry:=ant,bee,duck,zebra\do{
  \DTLnewrow{test1}
  \DTLnewdbentry{test1}{entry}{\myentry}
}
\DTLsetup{new-value-expand=true}
\DTLnewdb{test2}
\DTLaddcolumnwithheader{test2}{entry}
{Entry (Expanded)}
\@for\myentry:=ant,bee,duck,zebra\do{
  \DTLnewrow{test2}
  \DTLnewdbentry{test2}{entry}{\myentry}
}
\makeatother
\renewcommand{\myentry}{Unknown!}
```

3. Databases (datatool package)

```
\DTLdisplaydb{test1}  
\DTLdisplaydb{test2}
```

In the first case, the placeholder command ends up in the database entry, which means it's susceptible to the changing definition of that command. This means that every entry ends up with the same value. (If I hadn't redefined `\myentry` after the `\@for` loop it would have resulted in the "Undefined control sequence" error as at the end of the loop `\myentry` is `\@nil`, which is an undefined marker).

In the second case, the placeholder command `\myentry` is expanded before the entry is added to the database.

↑ Example 63: New Value Expansion

Entry (Not Expanded)	Entry (Expanded)
Unknown!	ant
Unknown!	bee
Unknown!	duck
Unknown!	zebra

This setting may also be switched on with:

```
\dtlexpandnewvalue
```

and switched off with:

```
\dtlnoexpandnewvalue
```

Note that the I/O `expand` option affects this setting. For example:

```
\DTLsetup{ io={expand=protected} }
```

This is equivalent to:

```
\DTLsetup{  
  io={expand=protected},  
  new-value-expand=true  
}
```

This means that:

```
\DTLread[expand=protected]{filename}
```

is a convenient shortcut for:

```
{\DTLsetup{new-value-expand=true}% local change
 \DTLread{filename}
 % where the file contains LATEX commands
}
```

new-value-trim=*<boolean>*

default: **true**; initial: **true**

This boolean option determines whether or not new values should have leading and trailing spaces trimmed before they are added to a database. Note that this option is independent of the `trim` option provided by the base package.

Ungrouped values added with the `new entry` action will always be trimmed due to the automated trimming of the `<key>=<value>` interface. However, if you specifically want leading or trailing spaces, you will need both `\DTLsetup{new-value-trim=false}` and `\DTLaction[value={ <value> },...]{new entry}` (that is, group the value and put the spaces inside the group).

Example 64 has the following:

```
\DTLnewdb{mydata}
\DTLnewrow{mydata}
\DTLsetup{new-value-trim=true}
\DTLnewdbentry{mydata}{Column1}{ value1 }
\DTLnewrow{mydata}
\DTLsetup{new-value-trim=false}
\DTLnewdbentry{mydata}{Column1}{ value2 }
% compare with \DTLaction:
\DTLsetup{default-name=mydata}
\DTLaction{new row}
\DTLaction[column=1,value= value3 ]{new entry}
\DTLaction{new row}
\DTLaction[column=1,value={ value4 }]{new entry}
```

The spaces can be shown by modifying the string formatting command to enclose the value in double-quotes:

64


```
\renewcommand{\dtlstringformat}[1]{``#1''}
\DTLdisplaydb{mydata}
```

↑ Example 64: Trimming New Values

Column1

“value1”

“ value2 ”

“value3”

“ value4 ”

io= { *⟨key=value list⟩* }

This option can't be used as a package option. The value is a *⟨key⟩=⟨value⟩* list of I/O settings for use with `\DTLread` and `\DTLwrite`, which are described in §3.15.2.

separator=*⟨char⟩*

initial: ,

This option may only be used as a package option and sets the separator used in CSV files. The value *⟨char⟩* must be a single token, and is used in a CSV file to separate columns within each row. After the package has loaded, you can use `\DTLsetseparator` to set the separator. Alternatively, you can use the `separator` setting within the optional argument of `\DTLread` or `\DTLwrite` or within the value of `io` option in `\DTLsetup`.

Note that the tab character is normally treated as a space by $\text{\LaTeX}$. For TSV files, the tab character will need to have its category code changed to distinguish it from a space. This can be done with `format=tsv` (in the `io` option) or with `\DTLsettabseparator`.

Examples:

```
% Set the default separator to ;
\usepackage[separator=;]{datatool}
% Set the default separator to |
\DTLsetup{ io={separator=|} }
% Load a CSV file with the separator :
\DTLread[ format=csv, separator={:} ]{file-1}
% Load a TSV file with the tab separator
\DTLread[ format=tsv ]{file-2}
```

store-datum=*<boolean>*default: **true**; initial: **false**

If true, new values will be stored as a datum item in the database (see §2.2). This means that each element doesn't need to be repeatedly parsed to determine whether it is numeric or what its numeric value is. If you want to save a database with the datum markup retained, you will need to use `format=dbtex-3`, as that's the only format to support datum items.

The `store-datum` option is useful if your document requires a lot of numeric computations (for example, aggregating data or plotting charts). However, it makes looking up rows by unique labels harder, so the setting is best left off with `datagidx` and `databib`.

3.2. Example Databases

There are a number of examples in this user guide that illustrate database commands on sample data. This section describes the sample data to provide a convenient point of reference for each example. Some databases are constructed within the example document preamble using `\DTLaction`. Some databases are loaded from a CSV file, which is the more common way of loading data, but the self-contained example documents need to create the required CSV file. This is done using the `filecontents` environment, except for the examples with a TSV file, which needs to have the tab character preserved.

3.2.1. Student Marks (CSV)

The “marks” database consists of columns with the labels: `Surname`, `Forename`, `Student-No` (a unique identifier, which disambiguates between the two students with the same name), and columns with the marks for each assignment.

This sample database is used in the following examples:

- 67. Select row action;
- 68. Row aggregate actions;
- 76. Referencing Rows from Displayed Data;
- 77. Inserting a Column at the Start of Displayed Data;
- 82. Display Data in a Stripy Table;
- 84. Display Two Fields in One Column;

3. Databases (datatool package)

- 85. Displaying Data with Calculations, Filtering and Row Highlighting;
- 87. Iterating Over Rows with `\DTLmapdata` to Append a Column;
- 89. Using `\DTLforeach` to Display a Stripy Table;
- 90. Displaying Data with Row Numbers Using `\DTLforeach`;
- 92. Editing a Database with `\DTLforeach`;
- 93. Loops and Alignment;
- 105. Sorting Data Using `\DTLsortdata` by Descending Numeric and Ascending String Values;
- 128. Multi Bar Chart;
- 129. Multi Bar Chart (Action ‘multibar chart’);
- 138. Multi Bar Chart With Group Labels;
- 148. Multi Bar Chart With a Legend.

The “marks” database is read in from the file `studentmarks.csv`, which contains the following content:

```
Surname,Forename,StudentNo,Assign1,Assign2,Assign3
"Smith, Jr",John,102689,68,57,72
"Brown",Jane,102647,75,84,80
"Brown",Jane,102646,64,92,79
"Brown",Andy,103569,42,52,54
"Adams",Zoë,105987,52,48,57
"Brady",Roger,106872,68,60,62
"Verdon",Clare,104356,45,50,48
```

The data can be loaded with:

```
\DTLread[name=marks]{studentmarks.csv}
```

Alternatively, you can setup the default database name first, to avoid having to repeatedly specify it. Since the data contains numeric values that may need to be parsed, it’s also useful to switch on the `store-datum` option to reduce parsing:

```
\DTLsetup{store-datum,default-name=marks}  
\DTLread{studentmarks.csv}
```

Note that this assumes the default settings for `\DTLread`:

```
\DTLread[format=csv, csv-content=literal]  
{studentmarks.csv}
```

This is only a short database for compactness. A similar, but longer database, is the students scores database.

3.2.2. Student Scores

The “scores” database consists of the columns: `forename` (with the title “First Name”), `surname` (with the title “Surname”), `regnum` (with the title “Student Number”), `gender` (which may be a recognised gender label, see §9.4), `parent` (the student’s guardian, parent or parents), `score` and `award`. The `award` column contains currency values, and the `score` column contains decimal values. The `regnum` column consists of a unique identifier. This happens to be numeric for this data, but may not necessarily be numeric for a real-world database. It’s included in the example data to demonstrate querying by a unique value and the data type isn’t relevant for that.

This sample database is used in the following examples:

- 80. Display Two Database Rows Per Tabular Row;
- 81. Display Two Database Rows Per Tabular Row (Top to Bottom);
- 83. Display Stripy Two Database Rows Per Tabular Row;
- 86. Iterating Over Rows with `\DTLmapdata` and `DTLenvmapdata`;
- 193. Mail Merging.

Since the data contains numeric values that may need to be parsed, it’s useful to switch on the `store-datum` option to reduce parsing. The database is constructed in the preamble of example documents as follows:

```
\DTLsetup{store-datum,default-name=scores}  
% define database:
```

3. Databases (datatool package)

```
\DTLaction{new}
% add columns in desired order:
\DTLaction[key=forename,value={First Name}]{add column}
\DTLaction[key=surname,value={Surname}]{add column}
\DTLaction[key=regnum,value={Student Number}]{add column}
\DTLaction[key=gender]{add column}
\DTLaction[key=parent]{add column}
\DTLaction[key=score,value={Score (\%)}]{add column}
\DTLaction[key=award]{add column}
% 1st row:
\DTLaction[
  assign={forename = Jane, surname = Brown,
    regnum = 102647, score = 75, award = {\$1,830},
    gender = F, parent = {Ms Brown}
  }
]{new row}
% 2nd row:
\DTLaction[
  assign={forename = John, surname = {Smith, Jr},
    regnum = 102689, score = 68, award = {\$1,560},
    gender = M, parent = {Mr and Mrs Smith}
  }
]{new row}
% 3rd row:
\DTLaction[
  assign={forename = Quinn, surname = Ó Coinn,
    regnum = 103294, score = 91, award = {\$3,280},
    parent = {Mr and Mrs Ó Coinn}
  }
]{new row}
% 4th row:
\DTLaction[
  assign={forename = Evelyn, surname = O'Leary,
    regnum = 107569, score = 81.5, award = {\$2,460},
    gender = n, parent = {Prof O'Leary}
  }
]{new row}
% 5th row:
\DTLaction[
  assign={forename = Zoë, surname = Adams,
    regnum = 105987, score = 52, award = {\$1,250},
    gender = f, parent = {Mr and Mrs Adams}
  }
]{new row}
% 6th row:
```

3. Databases (datatool package)

```
\DTLaction[
  assign={forename = Clare, surname = Vernon,
    regnum = 104356, score = 45, award = {\$500},
    gender = Female, parent = {Mr Vernon}
  }
]{new row}
% 7th row:
\DTLaction[
  assign={forename = Roger, surname = Brady,
    regnum = 106872, score = 58, award = {\$1,350},
    gender = m, parent = {Dr Brady and Dr Mady}
  }
]{new row}
% 8th row:
\DTLaction[
  assign={
    forename = Andy, surname = Brown, regnum = 103569,
    score = 42, award = {\$980},
    gender = male, parent = {Mr Brown and Prof Sepia}
  }
]{new row}
```

If you prefer a CSV file, the nearest equivalent would be:

```
forename,surname,regnum,gender,parent,score,award
Jane,Brown,102647,F,Ms Brown,75,"$1,830"
John,"Smith, Jr",102689,M,Mr and Mrs Smith,68,"$1,560"
Quinn,Ó Coinn,103294,,Mrs and Mrs Ó Coinn,91,"$3,280"
Evelyn,"O'Leary",n,Prof O'Leary,107569,81.5,"$2,460"
Zoë,Adams,105987,f,Mr and Mrs Adams,52,"$1,250"
Clare,Vernon,104356,f,Mr Vernon,45,"$500"
Roger,Brady,106872,m,Dr Brady and Dr Mady,58,"$1,350"
Andy,Brown,103569,m,Mr Brown and Prof Sepia,42,"$980"
```

However, the CSV format doesn't support missing mid-row values so the missing gender field for Quinn is now empty. This will make a difference if you display the data or test for null but not empty (see §3.10).

If the CSV file is called `studentscores.csv`, then it can be loaded with:

```
\DTLsetup{store-datum,default-name=scores}
\DTLread[ format=csv, csv-content=literal,
  headers={ First Name, Surname, Student Number,
    gender, parent, Score (\%), award }
]
```

```
] {studentscores.csv}
```

Note that if the dollar symbols (\$) in the file are replaced with L^AT_EX markup (\\$), then you will need `csv-content=tex` instead of `csv-content=literal`.

3.2.3. Customers

The “customers” database consists of columns with the labels: `Id` (a unique integer identifier, which happens to match the data row number but this isn’t guaranteed), `Organisation`, `Surname`, `Forename`, `Email`, and `Age` (another numeric column, which could potentially be decimal but only has integer numbers or missing items). There are some empty entries in the `Organisation`, `Email` and `Age` columns.



This sample database is used in the following examples:

- 94. CSV Data Containing Empty Cells and Missing Final Cells;
- 95. Constructed Data With Missing (Null) Values;
- 96. Display Data With Missing (Null) Values Shown as a Dash;
- 97. Iterating Through Data with Empty or Missing Values;
- 98. Editing a Row of Data;
- 99. Sorting CSV Data Using `\DTLsortdata` by `Organisation`, `Surname` and `Forename` With No Replacements;
- 100. Sorting CSV Data Using `\DTLsortdata` by `Organisation`, `Surname` and `Forename` With Replacements;
- 101. Sorting Data Using `\DTLsortdata` With Replacements (Null vs Empty);
- 102. Sorting CSV Data Using `\DTLsortdata` With Language Support;
- 103. Sorting Data Using `\DTLsortdata` on `Age` then `Surname` (Empty or Null Values);
- 104. Sorting Data Using `\DTLsortdata` on `Age` then `Surname` (No Empty Sort Values);
- 106. Sorting CSV Data Using `\dtlsort` by `Organisation`, `Surname` and `Forename` With Replacements;
- 111. Loading and Saving Data (Be Careful of Category Codes).

3. Databases (datatool package)

The customers database can be read in from the file `customers.csv`, which contains the following content:

```
Id,Organisation,Surname,Forename,Email,Age
1,,Parrot,Polly,pp@example.com,42
2,University of Somewhere,Canary,Mabel,mc@example.com
3,University of Somewhere,Zebra,Zoë,zz@example.com,21
4,Zinnia Florestry,Arara,José,ja@example.com,42
5,,Duck,Dickie,dd@example.com,
6,Newt Fellowship,Axolotl,Lizzie,la@example.com
7,Avian Emporium,Canary,Fred,fc@example.com,19
8,Newt Fellowship,,Molgina,m@example.com
9,,Mander,Sally
10,Élite Emporium,Fant,Eli,ef@example.com,101
```

The data can be loaded with:

```
\DTLread[name=customers]{customers.csv}
```

Alternatively, you can setup the default database name first, to avoid having to repeatedly specify it. Since the data contains numeric values that may need to be parsed, it's also useful to switch on the `store-datum` option to reduce parsing.

```
\DTLsetup{store-datum,default-name=customers}
\DTLread{customers.csv}
```

Note that this assumes the default settings for `\DTLread`:

```
\DTLread[format=csv, csv-content=literal]
{customers.csv}
```

Null values can only occur with data loaded from a CSV file when final columns are missing. In this case, the Age column is the last column and is not set in some rows. For example, there's no comma following Lizzie so Lizzie's age will be null. Compare this with the previous row where Dickie Duck has no age but there is a trailing comma. This will set Dickie Duck's age to empty. In the case of Sally Mander, both the Email and Age columns are missing. Since they are final columns both the email and age are null.

This data may also be defined within the document. Note that there is a slight difference here as most of the missing values are now entirely omitted from the database, so any reference to them will result in a null value rather than an empty value. However, there is one case where the Organisation column has been set to empty rather than being omitted, so a reference to that element will result in an empty value not a null value.



```

\DTLsetup{default-name=customers}
% define database:
\DTLaction{new}
% add columns in desired order:
\DTLaction[key=Id]{add column}
\DTLaction[key=Organisation]{add column}
\DTLaction[key=Surname]{add column}
\DTLaction[key=Forename]{add column}
\DTLaction[key=Email]{add column}
\DTLaction[key=Age]{add column}
% 1st row:
\DTLaction[
  assign={
    % Organisation not set
    Id = 1, Email = pp@example.com,
    Surname = {Parrot}, Forename = {Polly}, Age = 42
  }
]{new row}
% 2nd row:
\DTLaction[
  assign={
    % Age not set
    Id = 2, Organisation = {University of Somewhere},
    Email = mc@example.com, Surname = {Canary},
    Forename = {Mabel}
  }
]{new row}
% 3rd row:
\DTLaction[
  assign={
    Id = 3, Organisation = {University of Somewhere},
    Age = 21, Email = zz@example.com, Surname = {Zebra},
    Forename = {Zoë}
  }
]{new row}
% 4th row:
\DTLaction[
  assign={
    Id = 4, Organisation = {Zinnia Florestry}, Age = 42,
    Email = ja@example.com, Surname = {Arara},
    Forename = {José}
  }
]{new row}
% 5th row:

```

3. Databases (datatool package)

```
\DTLaction[
  assign={
    % Organisation and Age not set
    Id = 5, Surname = {Duck}, Forename = {Dickie},
    Email = dd@example.com
  }
]{new row}
% 6th row:
\DTLaction[
  assign={
    % Age not set
    Id = 6, Organisation = {Newt Fellowship},
    Email = la@example.com, Surname = {Axolotl},
    Forename = {Lizzie}
  }
]{new row}
% 7th row:
\DTLaction[
  assign={
    Id = 7, Organisation = {Avian Emporium}, Age = 19,
    Email = fc@example.com, Surname = {Canary},
    Forename = {Fred}
  }
]{new row}
% 8th row:
\DTLaction[
  assign={
    % Age and Surname not set
    Id = 8, Organisation = {Newt Fellowship},
    Email = m@example.com, Forename = {Molgina}
  }
]{new row}
% 9th row:
\DTLaction[
  assign={
    % Organisation empty and Age and Email not set
    Id = 9, Organisation = {},
    Surname = {Mander}, Forename = {Sally}
  }
]{new row}
% 10th row:
\DTLaction[
  assign={
    Id = 10, Organisation = {Élite Emporium}, Age = 101,
    Email = ef@example.com, Surname = {Fant},
```

```
Forename = {Eli}
}
]{new row}
```

3.2.4. Product List

The “product” database consists of the columns: Title, Author, Format (hardback, paperback or ebook), Quantity (integer), Price (decimal), and Notes (which is null in some rows and is only created by the first row to add an item to it).

This sample database is used in the following examples:

- 72. Display Data with Custom Alignment;
- 73. Display Data in a Table Omitting Columns;
- 74. Display Data in a Table with Named Columns;
- 75. Display Data in a Table with Filtered Rows;
- 78. Display Data in a Table with an Extra Column;
- 88. Display Data in a Table with `\DTLforeach`;
- 91. Using `\DTLforeach` to Display Data in a Table with a Running Total Column.

Since the data contains numeric values that may need to be parsed, it’s also useful to switch on the `store-datum` option to reduce parsing. The database is constructed in the preamble of example documents as follows:

```
\DTLsetup{store-datum,default-name=products}
% define database:
\DTLaction{new}
% add columns in desired order:
\DTLaction[key=Title]{add column}
\DTLaction[key=Author]{add column}
\DTLaction[key=Format]{add column}
\DTLaction[key=Quantity]{add column}
\DTLaction[key=Price,value={Price (\$)}]{add column}
% 1st row:
\DTLaction[
```

3. Databases (*datatool* package)

```
assign={
  Title = {The Adventures of Duck and Goose},
  Author = {Sir Quackalot},
  Format = paperback,
  Quantity = 3, Price = {10.99}
}
]{new row}
% 2nd row:
\DTLaction[
  assign={
    Title = {The Return of Duck and Goose},
    Author = {Sir Quackalot},
    Format = paperback,
    Quantity = 5, Price = {19.99}
  }
]{new row}
% 3rd row:
\DTLaction[
  assign={
    Title = {More Fun with Duck and Goose},
    Author = {Sir Quackalot},
    Format = paperback,
    Quantity = 1, Price = {12.99}
  }
]{new row}
% 4th row:
\DTLaction[
  assign={
    Title = {Duck and Goose on Holiday},
    Author = {Sir Quackalot},
    Format = paperback,
    Quantity = 3, Price = {11.99}
  }
]{new row}
% 5th row:
\DTLaction[
  assign={
    Title = {The Return of Duck and Goose},
    Author = {Sir Quackalot},
    Format = hardback,
    Quantity = 3, Price = {19.99}
  }
]{new row}
% 6th row:
\DTLaction[
```

3. Databases (*datatool* package)

```
assign={
  Title = {The Adventures of Duck and Goose},
  Author = {Sir Quackalot},
  Format = hardback,
  Quantity = 9, Price = {18.99}
}
]{new row}
% 7th row:
\DTLaction[
  assign={
    Title = {My Friend is a Duck},
    Author = {A. Parrot},
    Format = paperback,
    Quantity = 20, Price = {14.99}
  }
]{new row}
% 8th row:
\DTLaction[
  assign={
    Title = {Annotated Notes on the 'Duck and Goose'
chronicles},
    Author = {Prof Macaw},
    Format = ebook,
    Quantity = 10, Price = {8.99}
  }
]{new row}
% 9th row:
\DTLaction[
  assign={
    Title = {'Duck and Goose' Cheat Sheet for Students},
    Author = {Polly Parrot},
    Format = ebook,
    Quantity = 50, Price = {5.99}
  }
]{new row}
% 10th row:
\DTLaction[
  assign={
    Title = {'Duck and Goose': an allegory for modern
times?},
    Author = {Bor Ing},
    Format = hardback,
    Quantity = 0, Price = {59.99}
  }
]{new row}
```

```
% 11th row:
\DTLaction[
  assign={
    Title = {Oh No! The Chickens have Escaped!},
    Author = {Dickie Duck},
    Format = ebook,
    Quantity = 11, Price = {2.0}
  }
]{new row}
```

3.2.5. Price List

The “pricelist” database has the columns: Product, Quantity (integer), Price (currency), and Notes (which is null in some rows). Note that, unlike the larger products database above, the price column includes the currency symbol.



This sample database is used in examples 65 (Creating and Displaying a Database with \DTLaction) & 69 (Automatically Formatting Values Calculated by Actions).

Since the data contains numeric values that may need to be parsed, it’s useful to switch on the `store-datum` option to reduce parsing. The database is constructed in the preamble of example documents as follows:



```
% custom expandable command:
\newcommand{\limiteded}{limited edition}
% define a database with the name 'pricelist':
\DTLsetup{store-datum,default-name=pricelist}
\DTLaction{new}% create the default database
% 1st row:
\DTLaction[
  assign={
    Product = {The Adventures of Duck and Goose},
    Quantity = {1,452}, Price = {\$1.99}
  }
]{new row}
% 2nd row:
\DTLaction[
  assign={
    Product = {Duck and Goose on Holiday},
    Quantity = {94}, Price = {\$2.99}
```

```

    }
  ]{new row}
  % the next value needs to be expanded:
  \DTLaction[
    key={Notes}, expand-value={\limiteded} ]{new entry}
  % 3rd row:
  \DTLaction[
    assign={
      Product = {The Return of Sir Quackalot},
      Quantity = {3}, Price = {\$4.99}
    }
  ]{new row}

```

3.2.6. Balance Sheet (CSV)

The “balance” database consists of columns with the labels: Description, In, Out, and Balance. The last three columns are all numeric.



This sample database is used in Example 79 (Adjusting the Item Hook to Calculate Totals and Show Negative Numbers in Red).

The “balance” database is read in from the file `balance.csv`, which contains the following content:

```

Description,In,Out,Balance
Travel expenses,,230,-230
Conference fees,,400,-630
Grant,700,,70
Train fare,,70,0

```

The data can be loaded with:



```

\DTLread[
  name=balance, format=csv,
  headers={ Description, in (\pounds),
            Out (\pounds), Balance (\pounds) }
]{balance.csv}

```

The database name can be set as the default, if preferred. The `format=csv` setting is the default and so may be omitted. Since the data contains numeric values that may need to be parsed, it’s also useful to switch on the `store-datum` option to reduce parsing.

```
\DTLsetup{store-datum,default-name=balance}
\DTLread[
  headers={ Description, in (\pounds),
            Out (\pounds), Balance (\pounds) }
]{balance.csv}
```

3.2.7. Fruit (CSV)

The “fruit” database consists of columns with the labels: Name (string) and Quantity (numeric). The quantity includes decimal values, so evidently some fruit has been cut in half.

This sample database is used in the following examples:

- 114. Pie Chart;
- 115. Pie Chart (Action ‘pie chart’);
- 116. Pie Chart (Filtering);
- 117. Separating Segments from a Pie Chart;
- 118. Separating a Range of Segments from a Pie Chart;
- 119. Separating Individual Consecutive Segments from a Pie Chart;
- 120. Pie Chart (Inner and Outer Labels);
- 121. Pie Chart (Labels Rotated);
- 122. Pie Chart (Percentage Rounding);
- 123. Pie Chart (Changing the Label Format);
- 124. Pie Chart (Changing and Referencing the Segment Colours);
- 125. Vertical Bar Chart;
- 126. Vertical Bar Chart (Action ‘bar chart’);
- 130. Bar Chart With Labels;
- 131. Bar Chart With Labels (Action ‘bar chart’);
- 132. Bar Chart (Filtering);
- 141. Bar Chart With a Limited Set of Custom Colours;

- 142. Bar Chart Cycling through the Colour Set;
- 146. Every Bar Hook (Filtering);
- 147. Bar Chart With a Legend.

The “fruit” database is read in from the file `fruit.csv`, which contains the following content:

```
Name,Quantity
"Apples",30
"Pears",25
"Lemons,Limes",40.5
"Peaches",34.5
"Cherries",20
```

This file can be loaded with:

```
\DTLread[name=fruit,format=csv]{fruit.csv}
```

Again, the database name can be set as the default, if preferred. The `format=csv` setting is the default and so may be omitted. Since the data contains numeric values that may need to be parsed, it’s also useful to switch on the `store-datum` option to reduce parsing.

```
\DTLsetup{store-datum,default-name=fruit}
\DTLread{fruit.csv}
```

3.2.8. Profits (CSV)

The “profits” database has three columns with the labels: `Year`, `Profit` and `Units`. There are three negative values for the `Profit` column (that is, they are in fact losses not profits) which have been formatted slightly differently. Two have the minus sign before the currency symbol and one has the sign after the symbol. Both formats are supported. (Example 113 demonstrates how to automatically reformat the values to tidy them up.)

This sample database is used in the following examples:

- 113. Automatically Reformatting Data While Loading a CSV file;
- 127. Horizontal Bar Chart;

3. Databases (datatool package)

- 133. Horizontal Bar Chart with Labels (Default Alignment);
- 134. Horizontal Bar Chart with Labels (`lower-label-style=same`);
- 135. Horizontal Bar Chart with Labels (`lower-label-style=below`);
- 136. Horizontal Bar Chart with Labels (`lower-label-style=above`);
- 137. Horizontal Bar Chart with Upper Labels Over the Bars (negative `upper-label-offset`);
- 139. Bar Chart With Axes;
- 140. Bar Chart With Rotated Tick Labels;
- 143. Single Colours for Positive and Negative Bars;
- 144. Shaded Bar;
- 145. Hook at Every Bar.

The “profits” database is read in from the file `profits.csv`, which contains the following content:

```
Year,Profit,Units
1999,"-\$4,673",12467
2000,"\$2,525.49",8965
2001,"\$1,673.52",14750
2002,"-\$1,320.01",14572
2003,"\$5,694.83",13312
2004,"\$-451.67",9764
2005,"\$6,785.20",11235
```

Note that this uses `\$` rather than a literal `$` symbol, so `csv-content=tex` is required:

```
\DTLread[name=profits,format=csv,csv-content=tex]
{profits.csv}
```

Again, the database name can be set as the default, if preferred, and `format=csv` is the default so it may be omitted. Since the data contains numeric values that may need to be parsed, it's also useful to switch on the `store-datum` option to reduce parsing.

```
\DTLsetup{store-datum,default-name=profits}  
\DTLread[csv-content=tex]{profits.csv}
```

3.2.9. Time to Growth (CSV)

The “growth1” and “growth2” databases represents data obtained from hypothetical microbiological experiments, where a microbial population is observed at various time points. The two different sets of data correspond to different temperatures. (For example, the “growth1” data may have had the temperature set to 6 degrees and “growth2” may have had the temperature set to 8 degrees.) The first column in each case is the time observations. The other columns have the population figures as a log count.

These sample databases are used in the following examples:

- 149. Scatter Plot (One Database);
- 150. Scatter Plot (Two Databases);
- 151. Scatter Plot (Action);
- 159. Scatter Plot (Two Databases with Name Map);
- 165. Line and Scatter Plot (Two Databases);
- 169. Setting the Plot Bounds;
- 170. Rounding the Tick Labels;
- 171. Changing the Axis Style;
- 172. Grid;
- 173. Custom Grid Lines;
- 174. Plot Encapsulated in a Box;
- 175. Plot Encapsulated in a Box Without Ticks;
- 180. Side-Axes, Extended Axes and Boxed;
- 181. No Side-Axes, Extended Axes and Boxed.

The “growth1” database is read in from the file `growth1.csv`, which contains the following content:

3. Databases (datatool package)

```
Time,Experiment 1,Experiment 2
0,3.13,3.4
15,3.42,3.45
30,3.67,3.5
45,4.2,3.64
60,4.9,3.8
```

This file can be loaded with:

```
\DTLread[name=growth1,format=csv]{growth1.csv}
```

Since all values are plain numbers, an alternative is:

```
\DTLsetup{store-datum}
\DTLread[
  name=growth1,
  format=csv,
  csv-content=no-parse,
  data-types=decimal
]{growth1.csv}
```

This indicates that all columns have non-localised decimal content and by setting `store-datum` first, they won't need parsing when used in a numerical context.

The “growth2” database is read in from the file `growth2.csv`, which contains the following content:

```
Time,Experiment 1,Experiment 2
0,3.14,3.2
15,3.51,3.53
30,3.79,3.61
45,4.5,4.25
60,5.1,4.9
```

This file can be loaded with:

```
\DTLread[name=growth2,format=csv]{growth2.csv}
```

Again, since all values are plain numbers, an alternative is:

```
\DTLsetup{store-datum}  
\DTLread[  
  name=growth2,  
  format=csv,  
  csv-content=no-parse,  
  data-types=decimal  
{growth2.csv}
```

3.2.10. Time to Growth (TSV)

The “growthdata” database is an alternative to the above time to growth data. In this case the data is provided in a TSV file. Instead of having a single time column with two columns for the results of each experiment, it has four columns containing the time and log count for each experiment.

This sample database is used in the following examples:

- 112. Loading a TSV File;
- 152. Scatter Plot (One Database, Two Sets of Data);
- 153. Scatter Plot (Two Databases, Two Sets of Data);
- 154. Scatter Plot (Two Databases, Multiple Sets of Data);
- 155. Scatter Plot With Mismatched X and Y Columns;
- 156. Scatter Plot with Custom Legend Labels (One Database, Two Sets of Data);
- 157. Scatter Plot with Custom and Default Legend Labels (One Database, Two Sets of Data);
- 158. Scatter Plot with an Omitted Legend Label (One Database, Two Sets of Data);
- 160. Scatter Plot with Legend Label Mappings (Two Databases, Multiple Sets of Data);
- 161. Scatter Plot with Legend Label Mappings and Custom formatting (Two Databases, Multiple Sets of Data);
- 162. Scatter Plot with Custom Legend Labels (Two Databases, Multiple Sets of Data);
- 163. Scatter Plot with Shifted Legend (Two Databases, Multiple Sets of Data);

3. Databases (datatool package)

- 164. Scatter Plot with Custom Legend (Two Databases, Multiple Sets of Data);
- 166. Scatter Plot with Custom Colours and Styles (Two Databases, Multiple Sets of Data);
- 167. Scatter Plot with the Same Line Colour for Each Stream in a Given Database (Two Databases, Multiple Sets of Data);
- 168. Scatter Plot with Plot Marks Reset (Two Databases, Multiple Sets of Data).

The tab character is represented by the `↵` symbol. The first file is `growth.tsv`:

```
Experiment 1↵↵Experiment 2↵
Time↵Log Count↵Time↵Log Count
0↵2.9↵0↵3.31
15↵3.14↵10↵3.45
30↵3.26↵25↵3.61
45↵4.01↵40↵3.76
60↵4.2↵55↵3.89
```

This represents a spreadsheet where the first row originally had “Experiment 1” spanning the first two columns and “Experiment 2” spanning the last two columns. It was then exported to a TSV file, which doesn’t support column spanning entries, so “Experiment 1” is now in the first column and “Experiment 2” is in the third. This line needs to be omitted when parsing the file, which can be done with the `csv-skip-lines` option.

There is a similar second database “growthdata2” in the file `growth2.tsv`, but it has an extra pair of columns for a third experiment:

```
Experiment 1↵↵Experiment 2↵↵Experiment 3↵
Time↵Log Count↵Time↵Log Count↵Time↵Log Count
0↵3.21↵0↵3.39↵0↵3.28
15↵3.43↵10↵3.51↵10↵3.45
30↵3.68↵25↵3.65↵20↵3.57
45↵4.4↵40↵3.84↵30↵3.64
60↵4.8↵55↵3.92↵40↵3.95
```

In both files, the actual headers are in the second line: “Time”, “Log Count”, “Time” and “Log Count” (and, for the second file, another “Time” and “Log Count”). Note that they are duplicated, which means they are not suitable as unique column keys. Therefore it’s necessary to override the default behaviour to ensure unique keys. The format needs to be set to ensure that the tab character is recognised as the separator and has its category code changed so that it can be distinguished from a space.

Since the data contains numeric values, it can be more efficient to switch on the `store-datum` setting to reduce parsing if, for example, the data needs to be displayed in a graph.

```
\DTLsetup{store-datum}  
\DTLread[  
  name=growthdata, format=tsv, csv-skip-lines=1,  
  keys={Exp1Time, Exp1Count, Exp2Time, Exp2Count}  
]{growth}  
\DTLread[  
  name=growthdata2, format=tsv, csv-skip-lines=1,  
  keys={Exp1Time, Exp1Count, Exp2Time, Exp2Count,  
        Exp3Time, Exp3Count }  
]{growth2}
```

Note that `\DTLread` will assume a `tsv` extension with `format=tsv` so file extension may be omitted.

As with the CSV time to growth files, since the values are all plain numbers, a faster method is to switch off element parsing:

```
\DTLsetup{store-datum}  
\DTLread[  
  name=growthdata, format=tsv, csv-skip-lines=1,  
  keys={Exp1Time, Exp1Count, Exp2Time, Exp2Count},  
  csv-content=no-parse,  
  data-types=decimal  
]{growth}  
\DTLread[  
  name=growthdata2, format=tsv, csv-skip-lines=1,  
  keys={Exp1Time, Exp1Count, Exp2Time, Exp2Count,  
        Exp3Time, Exp3Count },  
  csv-content=no-parse,  
  data-types=decimal  
]{growth2}
```

3.2.11. Generic X/Y Data (CSV)

The “xydata” database just contains two columns of numbers that range from negative to positive.

This sample database is used in the following examples:

- 107. Loading Data With No Parsing;
- 108. Loading Data With No Parsing and Columns Identified as Decimal;

3. Databases (datatool package)

- 109. Loading Data With No Parsing and Columns Identified as Decimal and Currency;
- 110. Loading Data With No Parsing and Columns Identified as Decimal and Currency with Reformatting;
- 176. Positive and Negative Axes;
- 177. Extending the Axes;
- 178. Changing the Tick Label Node Style;
- 179. Side Axes;
- 182. Redefining the Start and End Hooks.

The “xydata” database is read in from the file `xydata.csv`, which contains the following content:

```
X, Y
-3.5, -2.75
-3, 3
-2.5, -1
-1, 1.5
1, -4.2
2.6, 1.8
3.2, -0.4
```

This file can be loaded with:

```
\DTLread[name=xydata, format=csv]{xydata.csv}
```

Again, since the data only contains plain numbers, it's faster to switch off parsing:

```
\DTLsetup{store-datum}
\DTLread[
  name=xydata,
  format=csv,
  csv-content=no-parse,
  data-types=decimal
]{xydata.csv}
```


3.3. Action Command

Some of the commands provided by datatool are quite long and it can be difficult to remember the syntax. Version 3.0 provides:

```
\DTLaction[⟨settings⟩]{⟨action⟩}
```

This will perform a command associated with the given action, with the arguments correctly set according to the values given in *⟨settings⟩*. For example:

```
\DTLaction{new}
```

is equivalent to:

```
\DTLnewdb{⟨default-name⟩}
```

where *⟨default-name⟩* is the default name (which can be changed with `default-name` in `\DTLsetup`). Alternatively, you can supply the database name:

```
\DTLaction[name=mydata]{new}
```

This is equivalent to:

```
\DTLnewdb{mydata}
```

Available actions are listed in §3.3.1 and settings are listed in §3.3.2. The *⟨action⟩* argument will be trimmed by `\DTLaction` to remove any leading or trailing spaces.

Example 65 is essentially equivalent to Example 70. It defines the “pricelist” database using actions (see §3.2.5) and then displays the database using the `display` action:

65

```
\DTLaction{display}
```

The “pricelist” database has null values as the Notes column isn’t set in every row (see §3.10).

↑ Example 65: Creating and Displaying a Database with `\DTLaction`

Product	Quantity	Price	Notes
The Adventures of Duck and Goose	1,452	\$1.99	NULL
Duck and Goose on Holiday	94	\$2.99	limited edition
The Return of Sir Quackalot	3	\$4.99	NULL

3. Databases (datatool package)

An action may have one or more return values consisting of a *primary* return value and (optionally) *secondary* return values that have an associated property name. There are several ways of fetching the return values.

The primary and secondary values can be obtained with:

```
\DTLget[⟨property⟩]{⟨cs⟩}
```

This will define the (token list variable) control sequence $\langle cs \rangle$ to the value obtained from the most recent `\DTLaction` in the current scope. Secondary return values should be identified by their property name. The $\langle property \rangle$ should be empty or omitted for the primary value.

Secondary (but not primary) values can also be obtained with the `return` setting, which should provide a comma-separated list of $\langle cs \rangle = \langle property \rangle$ assignments. Each listed control sequence $\langle cs \rangle$ will be defined to the value of the secondary property identified by $\langle property \rangle$.

If no return value is available (for example, the action failed or requested information was unavailable or the property name is unknown) then $\langle cs \rangle$ will be defined to a null value (see §3.10).

If you get a null value for an action that doesn't produce any errors or warnings, check the supplied action settings (such as `options` for the `aggregate` action) and also check that you have correctly spelt the property names.

For example, the `column index` action has the column index as the primary return value:

```
\DTLaction[key=Price]{column index}  
\DTLget{\colidx}  
Column index: \colidx.
```

```
\DTLuse{⟨property⟩}
```

This gets the primary or secondary return value and then uses it. Note that this command is only expandable if the argument $\langle property \rangle$ is empty (that is, for the primary return value). Otherwise it will expand to a robust internal command. If you need the value associated with a property in an expandable context, you will first have to fetch it with `\DTLget` or with the `return` option.

```
\DTLifaction{⟨property⟩}{⟨true⟩}{⟨false⟩}
```

Expands to $\langle true \rangle$ if the last action (within the current scope) set the return value identified by $\langle property \rangle$, otherwise it expands to $\langle false \rangle$. An empty $\langle property \rangle$ indicates the primary return value.

3.3.1. Defined Actions

All actions recognise the optional `return` setting, although it will have no effect with actions that don't have secondary return values. The descriptions below only identify optional settings where support varies according to the action.

3.3.1.1. Creation and Editing

```
\DTLaction[⟨settings⟩]{new}
```

Creates a new database. This action has one optional setting: `name`, which should be a single name. There are no required settings. Other action settings are ignored. The return value is the database name, which can also be accessed via the `name` secondary return property.

The `new` action internally uses `\DTLnewdb`. For example:

```
\DTLaction[name=mydata]{new}
```

is equivalent to:

```
\DTLnewdb{mydata}
```

Note that new databases are always globally defined.

```
\DTLaction[⟨settings⟩]{delete}
```

Deletes (undefines) a database (that is, the internal commands associated with the database are undefined). This action has one optional setting: `name`, which should be a single name. There are no required settings. Other action settings are ignored. The return value is the database name, which can also be accessed via the `name` return property. The other return properties are `rows` and `columns`, which will be set to the row and column count before the database was deleted.

```
\DTLaction[⟨settings⟩]{clear}
```

Clears a database (that is, the database is made empty but remains defined). This action has one optional setting: `name`, which should be a single name. There are no required settings. Other action settings are ignored. The return value is the database name, which can also be accessed via the `name` return property. The other return properties are `rows` and `columns`, which will be set to the row and column count before the database was cleared.

```
\DTLaction[⟨settings⟩]{new row}
```

Adds a new row to a database. This action has two optional settings: `name` (which should be a single name) and `assign`. There are no required settings. Other action settings are ignored.

3. Databases (datatool package)

As with `\DTLnewrow`, the `global` option determines whether or not the database is altered globally or locally.

The `assign` setting allows you to set the values for the new row at the same time. You can also add values to this new row with the `new entry` action afterwards. It's more efficient and more compact to set the values while creating the new row, but if some values should be expanded but not others, use the `new entry` action for those that need expanding. For example:

```
\DTLaction[
  assign={ Name = {José Arara}, Score = {68},
          Award = {\$2,453.99} }
]{new row}
```

This is equivalent to:

```
\DTLaction{new row}
\DTLaction[ key=Name, value={José Arara} ]
{new entry}
\DTLaction[ key=Score, value={68} ]{new entry}
\DTLaction[ key=Award, value={\$2,453.99} ]
{new entry}
```

The primary return value is the index of the new row, which will be the same as the updated row count. There is also a secondary return value that can be accessed with the `name` property, which will be the database name. The `row` property can also be used, which is the same as the primary return value. The difference is that `\DTLuse{ }` is expandable but `\DTLuse{row}` isn't.

The internal action of `new row` without the `assign` setting is essentially the same as `\DTLnewrow`. For example:

```
\DTLsetup{default-name={mydata}}
\DTLaction{new row}
```

which is the same as:

```
\DTLaction[name={mydata}]{new row}
```

is equivalent to:

```
\DTLnewrow{mydata}
```

Whereas with the `assign` setting, the `new row` action effectively implements not only `\DTLnewrow` but also one or more instances of `\DTLnewdbentry`.



```
\DTLaction[⟨settings⟩]{new entry}
```

Adds a new entry to the last row of a database. As with `\DTLnewdbentry`, the database must have at least one row, and the `global` option determines whether or not the database is altered globally or locally.

This action has one optional setting: `name`, which should be a single name. The required settings are: `value` (or `expand-value` or `expand-once-value`) and either `key` or `column`. Other action settings are ignored.

This action has secondary return values, which can be accessed with `\DTLget` or `\DTLuse` or the `return` setting, referenced by the following property names:

- `name`: the database name;
- `column`: the index of the new column;
- `key`: the column key;
- `row`: the index of the row (which will be the same as the row count);
- `type`: the data type integer identifier (see §2.2). Note that while the `type` action setting is a keyword, the `type` return value is a number.

The primary return value (accessed with an empty property) is the column index, so you can access the column index with either `\DTLuse{column}` or `\DTLuse{}`, but only the latter is expandable.



In general, it's better to have the default `new-value-expand=false`, and use `expand-value` or `expand-once-value` for the values that require expanding. (Unless the majority of your values require expansion.)

Note the difference between using the `\DTLsetup` option `new-value-expand=true` and the action setting `expand-value`. The first performs a protected expansion. For example:

```
\DTLsetup{new-value-expand=true}
\DTLaction[key=Price,value=\$1,234]{new entry}
```

This will add `\protect \$1,234` to the default database. Whereas the following:

```
\DTLsetup{new-value-expand=false}
\DTLaction[key=Price,expand-value=\$1,234]{new
entry}
```

3. Databases (datatool package)

will add `\protect \T1\textdollar 1,234` to the default database. In the case of currency, it's better not to expand the value otherwise the currency symbol may expand to something that's not recognised as a currency unit.



The `new entry` action internally uses `\DTLnewdbentry` if `key` is set. If `column` is used instead, a similar function is used, but be aware that listing column indexes out of order when the columns haven't yet been defined may have unexpected side-effects.

If you try to use both `key` and `column` this will cause an error and the column index will be ignored. If you use `key` and a column hasn't yet been defined, the column count will increase by 1 and a new column will be created at the end. If you use `column` and no column with that index has been defined, then a new column will be created with the key obtained by expanding `\dtldefaultkey <column-idx>` and, if the index is greater than the current number of columns, the database will be expanded so that it has a column count of `<column-idx>`.



```
\DTLaction[<settings>]{add column}
```

This action may be used to append a column to a database. Although the `new entry` action will automatically create an undefined column, you may prefer to define your columns in advance to ensure the ordering and to provide additional column metadata.

The `add column` action has optional settings: `name` (which should be a single name), `key`, `type`, and `value` (or `expand-value` or `expand-once-value`). Note that the `column` setting should not be used and will trigger an error if set. All other settings are ignored. The `\DTLsetup global` option determines whether or not the database is altered globally or locally.

Column Key

The column key, which must be unique to the database, will be obtained from the `key` setting, if provided. Otherwise, it will be obtained by expanding `\dtldefaultkey <col-idx>`, where `<col-idx>` is the index of the new column.

Column Header

The column header will be set to the `value`, if provided. Otherwise, it will be set to the column key.

Column Type

The column type will be set according to the `type` setting, if provided. Otherwise, the unknown type will be assumed. Note that the type will be updated if an entry with a greater type precedence is added to the column. For example, if you set `type=integer` but then add a decimal number to this column, then the column type will be updated to decimal.

3. Databases (datatool package)

This action has secondary return values, which can be accessed with `\DTLget` or `\DTLuse` or the `return` setting, referenced by the following property names:

- `name`: the database name;
- `column`: the index of the new column (which will be the same as the updated column count);
- `key`: the column key;
- `header`: the column header;
- `type`: the data type integer identifier (see §2.2).

The primary return value (accessed with an empty property) is the column index, so you can access the column index with either `\DTLuse{column}` or `\DTLuse{}`, but only the latter is expandable. (Alternatively, use `\DTLcolumncount{<name>}`.)

Example 66 creates a database and adds columns with actions:

66

```
\DTLaction{new}
\DTLaction{new row}
\DTLaction{add column}
Added column \DTLuse{column}
(key: \DTLuse{key}; header: \DTLuse{header})
to database '\DTLuse{name}'.

\DTLaction[key=quantity]{add column}
Added column \DTLuse{column}
(key: \DTLuse{key}; header: \DTLuse{header})
to database '\DTLuse{name}'.

\DTLaction[key=price,value=Price (\$)]{add column}
Added column \DTLuse{column}
(key: \DTLuse{key}; header: \DTLuse{header})
to database '\DTLuse{name}'.
```

↑ Example 66: Adding New Columns Using Actions



Added column 1 (key: Column1; header: Column1) to database 'untitled'.

Added column 2 (key: quantity; header: quantity) to database 'untitled'.

Added column 3 (key: price; header: Price (\$)) to database 'untitled'.

3.3.1.2. Querying

≡

\DTLaction[*<settings>*] {**find**}

Finds the first row in the database to match the supplied criteria. This action involves iterating over the database, applying the criteria to each row. If you want to lookup by a unique value, you may find it faster to use the `select row` action. Unlike the `select row` action, the `find` action doesn't change the current row (unless explicitly requested with the `select=true` option), so it may be used within the body of `\DTLforeach` to either lookup another row in the current database or in another database.

The `find` action doesn't have any required settings, but if none are provided it will simply find the first row of the database (if the database isn't empty). The optional settings are:

- `name`: identifies the database;
- `row`: identifies the row index to start the search from (defaults to 1, if omitted);
- `row2`: identifies the row index to end the search (defaults to the last row, if omitted);
- `assign`: an assignment list of `<cs>=<col-key>` pairs to define placeholder commands for each row, before the match function is used;
- `options`: may be used to specify options specific to the `find` action, see below.

The `options` value may include the following `<key>=<value>` options.

≡

direction=*<value>*

initial: **ascending**

Indicates the search direction. The value may be one of: `ascending` (or `asc`) or `descending` (or `desc`). If `direction=ascending`, the search will start from the smallest row index. If `direction=descending` the search will start from the largest row index.

○

select=*<boolean>*

default: **true**; *initial*: **false**

A boolean option that governs whether the first matching row to be found should be selected as the current row. If a match is found with `select=true`, then `\dtlgetrow` will be used to set the `\dtlcurrentrow` token register (and related registers) for use with actions (such as `row aggregate`) or commands (such as those described in §3.16.1). If unsuccessful or if `select=false`, the `\dtlcurrentrow` token register won't be changed.

≡

function=*<cs>*

Sets the match criteria function to `<cs>`, which must be defined to take a single argument, where the function definition expands to that argument to indicate a match and does nothing otherwise.



```
inline={ <definition> }
```

An inline alternative to `function`.

The default match function is simply a first of one function, which means that the first row (or last row with `direction=descending`) in the range `row-row2`, will match, provided the database isn't empty. If the `assign` setting is used, the placeholders can be referenced in the function. They will also still be available after the action, and will have their values from the matching row or from the final row in the search range if no match was found. They will be null if the database is empty. If there was no corresponding value in the row, they will be set to either `\DTLnumbernull` (if the column has a numeric data type) or `\DTLstringnull` otherwise (see §3.10).

The primary return value will be the row index which satisfied the match. The return value will not be set if no match was found. The secondary values will be set to the values of the matching row, where the property name is the column key. This means that you can access the values from the match even if you didn't set the corresponding assignment in `assign`.

For example, the following simply fetches all the values from row 2:



```
\DTLaction[row=2]{find}
```

Each value can then be displayed with `\DTLuse{<col-key>}`, where `<col-key>` is the column key.

The following finds the first row where the surname field is "Smith" and the forename field is "John":



```
\DTLaction[
  assign={\Surname=surname,\Forename=forename},
  options={
    inline={\DTLifstringeq{\Surname}{Smith}
      {\DTLifstringeq{\Forename}{John}{#1}}{}}
  ]{find}
\DTLifaction{}% test for primary return value
{\Forename\_\Surname\_\found on row \DTLuse{}}
{Not found}.
```



```
\DTLaction[<settings>]{column index}
```

Obtains the column index corresponding to the given `key`. This action does not create any typeset output. It performs a similar function as `\DTLgetcolumnindex` but it won't trigger an error if there's no column with the given key. Instead you need to test the return value with

3. Databases (datatool package)

`\DTLifnull`. Use `\dtlcolumnindex` instead if you want a single expandable function with no error checking.

An error will occur if the database is undefined or if the key is missing. This action has one optional setting: `name` (which should be a single name), and one required setting: `key`. Other settings are ignored.

The primary return value (if successful) is the column index, which may be accessed with `\DTLuse{}` or `\DTLget{<cs>}`. The `name` return property will be set to the database name, and the `key` return property will be set to the column key (if provided). The `column` property can also be referenced to obtain the column index, if successful.

For example:

```
\DTLaction[key=Price]{column index}
\DTLget{\colidx}
\DTLifnull{\colidx}{No such column}
{Column index: \colidx}.
```

```
\DTLaction[<settings>]{column data}
```

This action is similar to `column index` but gets all the column metadata (column index, key, type and header) from either the key or index.

The primary return value is the column key (regardless of whether the `key` or `column` setting was used). The secondary return properties are: `column` (the column index), `key` (the column key), `type` (the data type), and `header` (the column header). The return value will be null if the column doesn't exist.

An error will occur if the database is undefined or if there is no `key` or `column` setting or if both are provided. This action has one optional setting: `name` (which should be a single name), and one required setting: either `key` or `column` (but not both). Other settings are ignored.

```
\DTLaction[<settings>]{select row}
```

Selects a row and sets the `\dtlcurrentrow` token register for use with actions (such as `row aggregate`) or commands (such as those described in §3.16.1).

If the current row has already been selected (that is, `\dtlcurrentrow` and `\dtldb-name` have already been set), for example within the hooks used by `\DTLdisplaydb`, then you can instead use the `current row values` action to access information in the current row.

If you know the row index, you can use the `row` setting to select that row. This will internally use `\dtlgetrow`.

3. Databases (datatool package)

If you don't know the row index, but want to find the first row that exactly matches a particular value for a specific column then you need to use `value` (or `expand-value` or `expand-once-value`) for the required value and either `column` or `key` (but not both) to identify the column. In this case, the action will be similar to `\dtlgetrowforvalue` to find the first row that exactly matches the given value, but it won't trigger an error if no match is found.

If you want to match by a more complex test, such as a regular expression, use the `find` action instead with `function` or `inline` and `select=true` set in the action `options`.



Note that `value={ }` indicates an empty (not null) value. If you want to find the first row that doesn't have a particular column set, you can instead use the `find` action and search for a null value.

In either case, the `name` setting (which should be a single name) may be used to identify the database (which must exist). If omitted, the default is used. You can't have both `row` and a column identifier (`column` or `key`) set.



As with the underlying `\dtlgetrowforvalue`, this action (when matching a column) is primarily intended to work with a column which has unique values, such as an identification number. If you require a match on multiple columns or a regular expression match, you will need to iterate over the database or use the `find` action.

If successful, this action will set the token registers `\dtlcurrentrow`, `\dtlbefore-row` and `\dtlafterrow`, and also the placeholders `\dtldbname` (expands to the database name), `\dtlrownum` (the row index) and `\dtlcolumnnum` (the column index). If unsuccessful, `\dtlrownum` will be zero.

No return values will be set if unsuccessful, otherwise the primary return value is the row index (which will be the same as `\dtlrownum`), and the secondary return values will be the value of each entry found in the current row with the return property key the same as the column key.

The later Example 68 uses `\dtlgetrowforvalue` to select a row with a particular value from the "marks" database (see §3.2.1). Example 67 replaces this cumbersome command with the `select row` action. First the row selection:

67



```
\DTLaction[
  name=marks,
  key=StudentNo,
  value={105987}
]{select row}

Student \DTLuse{Forename} \DTLuse{Surname}
(105987) .
```

3. Databases (datatool package)

Then calculate the mean for the columns Assign1, Assign2 and Assign3. This can be done by column index, for example, `columns={4-6}` or by column key, for example, `keys={Assign1-Assign3}`. Since Assign3 is the last column of the database, an open-ended range may be used:

```
\DTLaction[
  keys={Assign1-},
  options={mean},
  datum={round=1}
]{current row aggregate}

Average mark: \DTLuse{mean}.

(Actual value: \DTLget[mean]{\theMean}
\DTLdatumvalue{\theMean}.)
```

Bear in mind that the second `\DTLaction` will clear the return values from the first, so if you need to continue referencing those values, take care to scope the second instance.

↑ Example 67: Select row action

Row selection:
Student Zoë Adams (105987).
Average mark: 52.3.
(Actual value: 52.33333333333333.)

```
\DTLaction[⟨settings⟩]{current row values}
```

If the current row has already been selected (that is, `\dtlcurrentrow` and `\dtldbname` have already been set), for example within the hooks used by `\DTLdisplaydb` or with `\dtl-getrow`, then the `current row values` action can be used to access values within the current row rather than using the more cumbersome `\dtlgetentryfromcurrentrow` for each required column.

For example, to fetch all values in the current row and use the values from the “Forename” and “Surname” columns:

```
\DTLaction{current row values}
Name: \DTLuse{Forename} \DTLuse{Surname}.
```

To store the values in placeholder commands with `\DTLget`:

```
\DTLaction{current row values}
\DTLget[Forename]{\Forename}
\DTLget[Surname]{\Surname}
```

Alternatively, with the `return` setting:

```
\DTLaction[
  return={
    \Forename=Forename,
    \Surname=Surname
  }
]{current row values}
```

There are no required settings. If you only want the values from a subset of columns you can identify those columns with `columns` and/or `keys`. Otherwise all columns will be assumed. A warning will occur if the `name` option is set as the name is expected to be in `\dtldbname`. An error will occur if `\dtldbname` hasn't been set. All other settings are ignored.

For example, the following collects the values for the columns with the labels “Title”, “Price”, “Quantity”, “Total” and the columns with the indexes 1 and 2:

```
\DTLaction[
  keys={Title,Price,Quantity,Total},
  columns={1,2}
]{current row values}
```

The primary return value is the number of values collected. This may be less than the total number of columns in the database or less than the list of supplied keys if there are missing columns in the current row. The secondary return properties are the column keys and the return value the corresponding element (which may have been parsed and converted into a datum item if the `datum` option was set, or if conversion automatically occurred with `store-datum` when the database was created).

Example 78 uses the `current row values` action to fetch row entries within the post-row hook of `\DTLdisplaydb` to append a total column to the table.

3.3.1.3. Aggregates

```
\DTLaction[⟨settings⟩]{aggregate}
```

Aggregates numerical data in one or two columns of the identified database. Either the `key` or `column` must be set (but not both) to identify the required column. The `key2` or `column2`

3. Databases (*datatool* package)

values may also be set (but not both) if a second column is also required. Optional settings are: `name` (which should be a single name) and `options`, which should be a comma-separated list of the aggregate functions to apply. The aggregate functions are as follows:

- `sum`: sum all numeric items in the given column. The return value will be in the `sum` property for the first column and, if applicable, `sum2` for the second column.
- `mean`: calculates the mean (average) of all numeric items in the given column. The return value will be in the `mean` property for the first column and, if applicable, `mean2` for the second column. This function automatically implements the `sum` function, since the total is required to calculate the mean.
- `variance`: calculates the variance of all numeric items in the given column. The return value will be in the `variance` property for the first column and, if applicable, `variance2` for the second column. This function automatically implements the `mean` function, since the mean is required to calculate the variance.
- `sd`: calculates the standard deviation of all numeric items in the given column. The return value will be in the `sd` property for the first column and, if applicable, `sd2` for the second column. This function automatically implements the `variance` function, since the variance is required to calculate the standard deviation.
- `min`: calculates the minimum of all numeric items in the given column. The return value will be in the `min` property for the first column and, if applicable, `min2` for the second column.
- `max`: calculates the maximum of all numeric items in the given column. The return value will be in the `max` property for the first column and, if applicable, `max2` for the second column.

If `options` is empty, the only functions will be to count and gather numeric items in a sequence.

The primary return value is the total number of numeric items in the first column. (Non-numeric items are skipped.) This will typically be the same as the row count, unless there are null or non-numeric items.



Return values that are numeric will be plain numbers. This is different to most of the aggregate commands described in §3.13, such as `\DTLmaxforkeys`, that return formatted numbers.

The secondary return value properties are:

- `name`: the database name.
- `column`: the index of the first column.
- `count`: the number of numeric items in the first column. This is the same as the primary return value.

3. Databases (datatool package)

- `seq`: the sequence of numeric items found in the first column. If you use `\DTLget [seq] {<cs>}` (or `return={<cs>=seq}`) then `<cs>` will be in the form of a `l3seq` sequence variable.
- `min`: if `min` was included in the `options` list, then this property will be set to the minimum value in the first column.
- `max`: if `max` was included in the `options` list, then this property will be set to the maximum value in the first column.
- `sum`: if `sum` was included in the `options` list (or implied by a function that requires the sum), then this property will be set to the sum of all numeric values in the first column.
- `mean`: if `mean` was included in the `options` list (or implied by a function that requires the mean), then this property will be set to the mean of all numeric values in the first column.
- `variance`: if `variance` was included in the `options` list (or implied by a function that requires the variance), then this property will be set to the variance of all numeric values in the first column.
- `sd`: if `sd` was included in the `options` list, then this property will be set to the standard deviation of all numeric values in the first column.

Additionally, if `key2` or `column2` have been set:

- `column2`: the index of the second column.
- `count2`: the number of numeric items in the second column.
- `seq2`: the sequence of numeric items found in the second column. If you use `\DTLget [seq2] {<cs>}` (or `return={<cs>=seq2}`) then `<cs>` will be in the form of a `l3seq` sequence variable.
- `min2`: if `min` was included in the `options` list, then this property will be set to the minimum value in the second column.
- `max2`: if `max` was included in the `options` list, then this property will be set to the maximum value in the second column.
- `sum2`: if `sum` was included in the `options` list (or implied by a function that requires the sum), then this property will be set to the sum of all numeric values in the second column.
- `mean2`: if `mean` was included in the `options` list (or implied by a function that requires the mean), then this property will be set to the mean of all numeric values in the second column.

3. Databases (datatool package)

- `variance2`: if `variance` was included in the `options` list (or implied by a function that requires the variance), then this property will be set to the variance of all numeric values in the second column.
- `sd2`: if `sd` was included in the `options` list, then this property will be set to the standard deviation of all numeric values in the second column.

```
\DTLaction[<settings>]{row aggregate}
```

Calculate aggregates for the current iteration of `\DTLmapdata`. (See Example 87.)

```
\DTLaction[<settings>]{current row aggregate}
```

Calculate aggregates for the current row stored in `\dtlcurrentrow`.

The actions `row aggregate` and `current row aggregate` essentially perform the same function. The difference between them is that `row aggregate` is for use within `\DTLmapdata` and `current row aggregate` is for use within `\DTLforeach` or after selecting a current row with the `select row` action or with commands like `\dtl-getrow`.

In either case, the database name should already be set in the `\dtldbname` placeholder, so the `name` option will trigger a warning, if set, and an empty `\dtldbname` will trigger an error. These actions are similar to `aggregate` but they aggregate items in the columns of the current row that have a numeric value.

By default all columns in the current row will be checked, but you can restrict the function to a subset of columns with the `columns` and/or `keys` options.

The `options` setting is as for the `aggregate` action. The primary return value is the number of numeric columns contributing to the aggregates. The secondary return value properties are:

- `name`: the database name (same as `\dtldbname`). If any unexpected results occur, check this return value matches the expected name.
- `row`: the row index (same as the value of `\dtlrownum`). If any unexpected results occur, check this return value matches the expected row index.
- `columns`: the list of indexes of all column in the subset or empty if no subset specified.
- `count`: the number of numeric items in the subset. This is the same as the primary return value.
- `seq`: the sequence of numeric items found in the subset. If you use `\DTLget[seq]{<cs>}` (or `return={<cs>=seq}`) then `<cs>` will be in the form of a `!3seq` sequence variable.
- `min`: if `min` was included in the `options` list, then this property will be set to the minimum value in the subset.

3. Databases (datatool package)

- max: if max was included in the `options` list, then this property will be set to the maximum value in the subset.
- sum: if sum was included in the `options` list (or implied by a function that requires the sum), then this property will be set to the sum of all numeric values in the subset.
- mean: if mean was included in the `options` list (or implied by a function that requires the mean), then this property will be set to the mean of all numeric values in the subset.
- variance: if variance was included in the `options` list (or implied by a function that requires the variance), then this property will be set to the variance of all numeric values in the subset.
- sd: if sd was included in the `options` list, then this property will be set to the standard deviation of all numeric values in the subset.

Example 68 uses the “marks” database (see §3.2.1) and calculates the average marks for each student within `\DTLmapdata`:

68

```
\DTLmapdata[name=marks]{
  \DTLmapget{key=Forename} \DTLmapget{key=Surname}
  average marks:
  \DTLaction[
    columns={4-},
    options={mean}
  ]{row aggregate}
  \DTLuse{mean}.
}
```

For comparison, the example also uses `\DTLforeach`:

```
\DTLforeach{marks}
{ \Forename=Forename, \Surname=Surname }
{
  \Forename\_\Surname\_ average mark:
  \DTLaction[
    columns={4-},
    options={mean}
  ]{current row aggregate}
  \DTLuse{mean}.
}
```

3. Databases (datatool package)

And selects a particular row:

```
\dtlgetrowforvalue{marks}{\dtlcolumnindex{marks}
{StudentNo}}{105987}
Student 105987 average mark:
\DTLaction[
  columns={4-},
  options={mean}
]{current row aggregate}
\DTLuse{mean}.
```

The rather cumbersome `\dtlgetrowforvalue` can be replaced with the `select row` action, as in the earlier Example 67.

↑ Example 68: Row aggregate actions

Map data:
John Smith, Jr average marks: 65.66666666666667.
Jane Brown average marks: 79.66666666666667.
Jane Brown average marks: 78.33333333333333.
Andy Brown average marks: 49.33333333333333.
Zoë Adams average marks: 52.33333333333333.
Roger Brady average marks: 63.33333333333333.
Clare Verdon average marks: 47.66666666666667.
For each:
John Smith, Jr average mark: 65.66666666666667.
Jane Brown average mark: 79.66666666666667.
Jane Brown average mark: 78.33333333333333.
Andy Brown average mark: 49.33333333333333.
Zoë Adams average mark: 52.33333333333333.
Roger Brady average mark: 63.33333333333333.
Clare Verdon average mark: 47.66666666666667.
Row selection:
Student 105987 average mark: 52.33333333333333.

3.3.1.4. Tabulation

```
\DTLaction[⟨settings⟩]{display}
```

This action may be used to display a database using the same underlying function as `\DTL-`

3. Databases (datatool package)

`displaydb*`. This action has optional settings: `name` (which should be a single name) and `options` to pass any options to `\DTLdisplaydb*`. Other settings are ignored.

There's no primary return value, but there are secondary return values that can be accessed with the properties: `name` (the database name), `columns` (the number of columns displayed), and `rows` (the number of rows displayed).

For example:

```
\DTLaction[options={omit-columns={1,3}}]{display}
```

This is essentially the same as:

```
\DTLdisplaydb*[omit-columns={1,3}]{<default-name>}
```

where `<default-name>` is obtained from the `default-name` option. The action has the advantage over `\DTLdisplaydb` as you can use the return values to find out how many columns or rows were displayed (which may not necessarily be the same as the column count or row count).

```
\DTLaction[<settings>]{display long}
```

This is similar to the `display` action, but it uses the underlying function of `\DTLdisplaylongdb`, which uses `longtable` instead of `tabular`. This action has optional settings: `name` (which should be a single name) and `options` to pass any options to `\DTLdisplaylongdb`. Other settings are ignored.

There's no primary return value, but there are secondary return values that can be accessed with the properties: `name` (the database name), `columns` (the number of columns displayed), and `rows` (the number of rows displayed).

3.3.1.5. Modifying a Database

```
\DTLaction[<settings>]{sort}
```

Sorts a database using `\DTLsortdata`. This action has optional settings: `name` (the database name), and `options` (the options to pass to the optional argument of `\DTLsortdata`). There is one required settings: `assign`, which should be the criteria to pass in the final argument of `\DTLsortdata`.

The primary return value should be equal to the number of rows of the database if no errors occurred. The secondary return values can be accessed with the properties: `name` (the database name, which will always be set), `columns` (the number of columns in the database after sorting) and `rows` (the number of rows in the database). The column count of the database may increase if the options include instructions to add the sort or group information to the database. See §3.14.1 for further details.

3.3.1.6. Other

The databar package provides the `bar chart` and `multibar chart` actions. The datapie package provides the `pie chart` action. The dataplot package provides the `plot` action.

3.3.2. Action Settings

Action settings may only be used in the optional argument of `\DTLaction` and can't be used in `\DTLsetup`. They are reset to their default values by `\DTLaction` before the optional argument is processed. Settings that aren't required by the given action are usually ignored, but an unrequired setting may occasionally generate an error or warning if it's likely that the setting may accidentally be used instead of the correct one (for example, setting `column` when the column can only be identified with `key`).

```
name=<db-name(s)>
```

The database name or (where supported) the list of names. If omitted, the value of the general `default-name` option is used. For example:

```
\DTLaction[name=mydata]{new}
```

Some actions don't permit the database name to be specified as it's expected to be provided by `\dtldbname` (such as `current row aggregate`). Actions that require a single name will take the first from the list and ignore the rest of the list.

```
key=<label>
```

The unique column key. This must be set to a non-empty value for an action that allows a column reference by ID, except in the case of actions that use `keys` for a list of keys. Typically, you won't be able to use both `key` and `column`.

```
key2=<label>
```

If an action requires a second column reference, this should be used to reference the second column by its unique ID. This is intended for use by actions that require at most two columns, not for actions that use `keys` for a list of keys. Typically, you won't be able to use both `key2` and `column2`.



```
column= $\langle n \rangle$ 
```

The column index. This must be set to a positive number for an action that allows a column reference by index, except in the case of actions that use `columns` for a list of column indexes.



```
column2= $\langle n \rangle$ 
```

If an action requires two column references, this should be used to reference the second column by its index. This is intended for use by actions that require at most two columns, not for actions that use `columns` for a list of column indexes.



```
columns={  $\langle list \rangle$  }
```

If an action allows an arbitrary number of column references, the `columns` option can be used to reference the required columns by their index in a comma-separated list.

The list may include ranges in the form $\langle n_1 \rangle - \langle n_2 \rangle$, where $\langle n_1 \rangle$ is the start of the range and $\langle n_2 \rangle$ is the end. If $\langle n_1 \rangle$ is omitted then 1 is assumed and if $\langle n_2 \rangle$ is omitted, the last column is assumed. For example, `columns={ -4 }` is equivalent to `columns={ 1-4 }` and `columns={ 1, 3-5 }` indicates columns 1, 3, 4, and 5. Where a range with a start and end value is provided, the start value must be less than or equal to the end value.



```
keys={  $\langle list \rangle$  }
```

If an action allows an arbitrary number of column references, the `keys` option can be used to reference the required columns by their key in a comma-separated list. Typically, an action that allows a list of columns may allow both `keys` and `columns` and will merge the subsets. As with `columns`, the list may include ranges in the form $\langle key1 \rangle - \langle key2 \rangle$, where $\langle key1 \rangle$ is the start of the range and $\langle key2 \rangle$ is the end. As with `columns`, if the start range is omitted, the first column is assumed, and if the end range is omitted, the last column is assumed. For example, the “marks” database (see §3.2.1) may have `keys={ Assign1- }` (as in Example 87).

Unlike `columns`, if the associated column index of $\langle key1 \rangle$ is greater than the associated column index of $\langle key2 \rangle$, the start and end references will be switched round.



```
row= $\langle n \rangle$ 
```

The row index. This must be set to a positive number for an action that requires a row reference by index.



The row index is a reference to the internal data and is unrelated to references in the original source (such as line numbers in a CSV file).

3. Databases (datatool package)

```
row2=<n>
```

The second row index. This must be set to a positive number for an action that requires a second row reference by index.

```
assign={ <key=value list> }
```

A $\langle key \rangle = \langle value \rangle$ list of assignments. For example, this can be used in the `new row` action to assign values to specific columns according to the column key, in this case the $\langle key \rangle$ part in $\langle key \rangle = \langle value \rangle$ is the column key. In the case of actions such as `pie chart` and `bar chart`, each $\langle key \rangle$ part is a placeholder command.

```
options={ <list> }
```

A comma-separated list or $\langle key \rangle = \langle value \rangle$ list used by certain actions. In the case of the `display` action, this provides the option list to pass to `\DTLdisplaydb*`. For example:

```
\DTLaction[options={only-keys={Product,Price}}]
{display}
```

This is equivalent to:

```
\DTLdisplaydb*[only-keys={Product,Price}]{<default-name>}
```

Whereas with the `aggregate` action, `options` provides a list of required aggregate functions.

```
value=<value>
```

A value needed by certain actions. For example, in the case of the `new entry` action, the `value` setting is the value to add to the database:

```
\DTLaction[key=Price,value=\$1.23]{new entry}
```

This is equivalent to:

```
\DTLnewdbentry{<default-name>}{Price}{\$1.23}
```

where $\langle default-name \rangle$ is obtained from the `default-name` setting.



If `value`, `expand-value` or `expand-once-value` occur in the same option list then they will override each other. The last one in the list will take precedence.



`expand-value=<text>`

This is equivalent to using the `value` key with `<text>` fully expanded.



`expand-once-value=<text>`

This is equivalent to using the `value` key with `<text>` expanded once. This is the best setting to use if you have a placeholder command or token list. For example,



```
\newcommand{\price}{\$1.23}
\DTLaction[
  key=Price,
  expand-once-value=\price
]{new entry}
```



`type=<value>`

The data type (see §2.2), where the value may be one of: `string`, `integer` (or `int`), `decimal` (or `real`), or `currency`.



`return={ <assign-list> }`

Secondary (but not primary) values can also be obtained with the `return` setting, which should provide a comma-separated list of `<cs>=<property>` assignments. Each listed control sequence `<cs>` will be defined to the value of the secondary property identified by `<property>`. This may be used instead of, or in addition to, using `\DTLget`.



`datum={ <key=value list> | true | false }` *default: **true**; initial: **false***

This setting governs whether or not secondary return values should be formatted as datum items. It's primarily intended as a shortcut for actions such as `aggregate` to avoid the cumbersome use of `\dtlround` and `\DTLdecimaltolocale` to format the results.



The `datum` setting doesn't affect primary return values. However, since the primary return value is often (but not always) duplicated in the secondary set, the formatted value can be obtained from the applicable secondary property. Complex secondary values that have their own markup, such as the `seq` return property for the `aggregate` action are also not affected.

Available values are: `datum=false` (don't format secondary return values), `datum=true` (format secondary return values without changing the `datum` settings) or `datum={key=value list}` to enable with the given subset of `datum` settings. For example, `datum={round=2, currency=\$}`. Note that `datum=true` is essentially the same as `datum={}`.



The original numeric value can still be obtained with a combination of `\DTLget` to fetch the value as a datum control sequence and `\DTLdatumvalue` to extract the plain number (see Example 67). With `\DTLuse`, the formatted number will be inserted into the document. There's little advantage in using `datum` with text-only return values.

If `datum` is not set to `false`, then the secondary value format (in the string part of the datum item) can be adjusted according to the following options, which may be set in `datum={key=value list}`. However, in the case of secondary return values that simply provide elements from the database (such as those from the `select row` action), the return values will be datum items (obtained using `\DTLparse`), but won't be governed by the options listed below.



locale-integer=*<boolean>*

default: true; initial: false

If this boolean option is `true`, then the string part of secondary return values that are known to always be integers (if set), such as a column or row index, will be formatted according to the current localisation setting.



locale-decimal=*<boolean>*

default: true; initial: true

If this boolean option is `true`, then the string part of calculated numeric datum items (such as `sum` or `mean`) will be formatted according to the current localisation setting. If this option is `false`, the string part will use a plain number but it will still be affected by the `currency` and `round` options. Note that the `sum` return property is always considered a decimal in this context, even if only integer values were summed.



currency=`false|match|default` | *<symbol>*
initial: match

default: default;

This option only governs decimal return values that have been calculated (such as the `sum` or

mean in the `aggregate` action). Available option values:

- `false`: no currency symbol is inserted;
- `match`: the matching currency symbol will be inserted if one was found in the original data;
- `default`: the default currency symbol will be inserted before all calculated decimal values (regardless of whether or not the original values were identified as currency);
- `<symbol>`: the given currency symbol will be inserted before all calculated decimal values (regardless of whether or not the original values were identified as currency).

round=`<number>` | `false`

default: 0; initial: false

This option only governs decimal return values that have been calculated (such as the sum or mean in the `aggregate` action) and indicates whether the value should be rounded. The keyword `false` or a negative value may be used to prevent rounding. Otherwise the value should be set to a non-negative number indicating the required number of decimal places.

Example 69 uses the “pricelist” database (see §3.2.5), which has an integer column labelled “Quantity” and a currency column labelled “Price”. The aggregates for both columns can be obtained with the `aggregate` action:

69

```
\DTLaction[
  key=Quantity,
  key2=Price,
  options={sd,min,max}
]{aggregate}
```

```
Quantity column index: \DTLuse{column}.
Total quantity: \DTLuse{sum}.
Average quantity: \DTLuse{mean}.
Quantity standard deviation: \DTLuse{sd}.
Minimum quantity: \DTLuse{min}.
Maximum quantity: \DTLuse{max}.
```

```
Price column index: \DTLuse{column2}.
Total price: \DTLuse{sum2}.
Average price: \DTLuse{mean2}.
Price standard deviation: \DTLuse{sd2}.
Minimum price: \DTLuse{min2}.
Maximum price: \DTLuse{max2}.
```

3. Databases (*datatool* package)

This displays all the statistics as plain numbers (Example 69). Using `datum` will produce formatted numbers for the calculated values (but not for the column index):

```
\DTLaction[
  datum={round=2},
  key=Quantity,
  key2=Price,
  options={sd,min,max}
]{aggregate}
```

Note that this will convert the total, minimum and maximum quantities to decimals rounded to 2 decimal places (but not the column index). The actual numeric values can be obtained with `\DTLget` and `\DTLdatumvalue`:

```
Quantity column index: \DTLuse{column}.
Total quantity: \DTLuse{sum}
(\DTLget[sum]{\theTotal}\DTLdatumvalue{\theTotal}).
Average quantity: \DTLuse{mean}.
Quantity standard deviation: \DTLuse{sd}.
Minimum quantity: \DTLuse{min}
(\DTLget[min]{\theMin}\DTLdatumvalue{\theMin}).
Maximum quantity: \DTLuse{max}
(\DTLget[max]{\theMax}\DTLdatumvalue{\theMax}).
```

Note the difference if a currency symbol is enforced:

```
\DTLaction[
  datum={round=2,currency},
  key=Quantity,
  key2=Price,
  options={sd,min,max}
]{aggregate}
```

This converts all the quantity aggregate values to currency, which is inappropriate in this case.

↑ Example 69: Automatically Formatting Values Calculated by Actions



1 Default `datum=false`

Quantity column index: 2. Total quantity: 1549. Average quantity: 516.3333333333333.
Quantity standard deviation: 662.6584506532927. Minimum quantity: 3.
Maximum quantity: 1452.

Price column index: 3. Total price: 9.97. Average price: 516.3333333333333.
Price standard deviation: 513.011516104225. Minimum price: 1.99. Maximum price: 4.99.

2 `datum={round=2}`

Quantity column index: 2. Total quantity: 1,549.00 (1549). Average quantity: 516.33. Quantity standard deviation: 662.66. Minimum quantity: 3.00 (3). Maximum quantity: 1,452.00 (1452).

Price column index: 3. Total price: \$9.97. Average price: \$516.33. Price standard deviation: \$513.01. Minimum price: \$1.99. Maximum price: \$4.99.

3 `datum={round=2,currency}`

(All aggregate values become currency!)

Quantity column index: 2. Total quantity: \$1,549.00 (1549). Average quantity: \$516.33. Quantity standard deviation: \$662.66. Minimum quantity: \$3.00 (3). Maximum quantity: \$1,452.00 (1452).

Price column index: 3. Total price: \$9.97. Average price: \$516.33. Price standard deviation: \$513.01. Minimum price: \$1.99. Maximum price: \$4.99.

3.4. Creating a New Database

This section describes commands that may be used in a document to create a database or to locally or globally alter a database. The `global` option determines whether or not the modifications to the database are global or local, except for those commands that are listed as specifically global only. Note that new databases are always globally defined.

The `new-value-trim` option determines whether or not values are trimmed before adding to a database, and the `new-value-expand` option determines whether or not values should be expanded before adding. The `store-datum` option determines whether or not the values should be as a datum item.

3. Databases (datatool package)



A new database can be created from data in an external file using `\DTLread`. In that case, the database is always defined globally because `\DTLread` introduces an implicit group to localise the settings passed in the optional argument. See §3.15 for further details.



```
\DTLgnewdb{<db-name>}
```

Globally defines a new database with the label `<db-name>`. If a database already exists with the given label, an error will occur. Alternatively, you can use the `new` action:

```
\DTLaction[name={<db-name>}] {new}
```



New databases are always global, regardless of the `global` option, because the underlying registers used to store the database structure have to be globally defined, so `\DTLnewdb` is equivalent to `\DTLgnewdb`.

Before you can add any data to a database, you must start a new row.



```
\DTLnewrow{<db-name>} modifier: *
```

This adds a new row to the database identified by the label `<db-name>`. The `global` option determines whether or not the change is global. If a database with the given label doesn't exist, an error will occur with the unstarred version. The starred version `\DTLnewrow*` doesn't check for existence. Alternatively, you can use the `new row` action (which checks for existence):

```
\DTLaction[name={<db-name>}] {new row}
```

Once you have added a new row, you can add entries to that row with:



```
\DTLnewdbentry{<db-name>}{<col key>}{<value>} modifier: *
```

This adds an entry with the given value to the column identified by the label `<col key>` in the last row of the database identified by the label `<db-name>`. The `global` option determines whether or not the change is global. Alternatively, you can use the `new entry` action (which checks for existence):

```
\DTLaction[
  name={\db-name},
  key={\col key},
  value={\value}
]{new entry}
```

If a database with the given label doesn't exist or the row already contains an entry in that column, an error will occur with the unstarred version. The starred version `\DTLnewdb-entry*` doesn't check for existence of the database, but will still trigger an error if an entry for the given column already exists.

If a column with the given label doesn't yet exist, it will be created and the default metadata will be assigned. The `store-datum` option determines whether or not the value is stored in the database as a datum item. It will be parsed regardless of that setting in order to set or update the column data type. The `new-value-trim` option determines whether or not the value should have leading and trailing spaces trimmed. The `new-value-expand` option determines whether or not the value should be expanded.

Note that with the default `new-value-expand=false`, you can expand a particular value in `\DTLaction{new entry}` with the `expand-value` or `expand-once-value` action option. With `new-value-expand=true`, the value will always have protected expansion applied.

Example 70 creates a database labelled “mydata” as follows:

70

```
% custom expandable command:
\newcommand{\limiteded}{limited edition}
% define data
\DTLnewdb{mydata}
\DTLnewrow{mydata}% create a new row
% Add entries to the first row:
\DTLnewdbentry
{mydata}% database label
{Product}% column key
{The Adventures of Duck and Goose}% value
\DTLnewdbentry
{mydata}% database label
{Quantity}% column key
{1,452}% value
\DTLnewdbentry
{mydata}% database label
{Price}% column key
{\$1.99}% value
\DTLnewrow{mydata}% create a new row
```

3. Databases (datatool package)

```
% Add entries to the second row:
\DTLnewdbentry
{mydata}% database label
{Product}% column key
{Duck and Goose on Holiday}% value
\DTLnewdbentry
{mydata}% database label
{Quantity}% column key
{94}% value
\DTLnewdbentry
{mydata}% database label
{Price}% column key
{\$2.99}% value
% the next value needs to be expanded:
\DTLsetup{new-value-expand}
\DTLnewdbentry
{mydata}% database label
{Notes}% column key
{\limiteded}% value
% switch off expansion:
\DTLsetup{new-value-expand=false}
\DTLnewrow{mydata}% create a new row
% Add entries to the third row:
\DTLnewdbentry
{mydata}% database label
{Product}% column key
{The Return of Sir Quackalot}% value
\DTLnewdbentry
{mydata}% database label
{Quantity}% column key
{3}% value
\DTLnewdbentry
{mydata}% database label
{Price}% column key
{\$4.99}% value
```

Note that the second row has introduced a fourth column with the label “Notes”. Since the other rows don’t have this column set, an attempt to access it will result in a null value. Expansion needs to be switched on when a value must be expanded. This is commonly the case with placeholder commands. The setting must be switched off again or it will cause the currency symbol to prematurely expand in the next row (which means the datum parser won’t be able to detect it as currency).

The contents of the database can now be displayed with:

```
\DTLdisplaydb{mydata}
```

This displays the database in a tabular environment, as shown in Example 70.

↑ Example 70: Creating a New Database with a Label

Product	Quantity	Price	Notes
The Adventures of Duck and Goose	1,452	\$1.99	NULL
Duck and Goose on Holiday	94	\$2.99	limited edition
The Return of Sir Quackalot	3	\$4.99	NULL

See Example 65 for an equivalent document using `\DTLaction`.

Short commands, such as `\DTLnewdbentry` don't permit `\par` on the argument. If you have a value that spans multiple paragraphs, you will need to mark the paragraph breaks with:

```
\DTLpar
```

This is a robust command that simply does `\par`.

3.5. Deleting or Clearing a Database

Deleting or clearing a database simply undefines or resets the underlying commands and registers that are used to represent the database.

```
\DTLdeletedb{<db-name>}
```

Deletes the database identified by the label `<db-name>` (that is, the internal commands associated with the database are undefined). The `global` option determines whether or not the change is global. If a database with the given label doesn't exist, an error will occur. Alternatively, you can use the `delete` action:

```
\DTLaction[name={<db-name>}] {delete}
```

```
\DTLgdeletedb{<db-name>}
```

Globally deletes the database identified by the label `<db-name>`, regardless of the `global` setting. If a database with the given label doesn't exist, an error will occur.

```
\DTLcleardb{<db-name>}
```

Clears the database identified by the label *<db-name>*. That is, the database is made empty (no rows or columns) but is still defined. The `global` option determines whether or not the change is global. If a database with the given label doesn't exist, an error will occur. Alternatively, you can use the `clear` action:

```
\DTLaction[name={<db-name>}] {clear}
```

```
\DTLgcleardb{<db-name>}
```

Globally clears the database identified by the label *<db-name>*, regardless of the `global` setting. If a database with the given label doesn't exist, an error will occur.

3.6. Database Conditionals and Metadata

You can test if a database exists with:

```
\DTLifdbexists{<db-name>}{<true>}{<false>}
```

This does *<true>* if a database with the label *<db-name>* exists, otherwise it does *<false>*.

You can test if a database is empty with:

```
\DTLifdbempty{<db-name>}{<true>}{<false>}
```

This does *<true>* if a database with the label *<db-name>* is empty, otherwise it does *<false>*. If a database with the given label doesn't exist, an error will occur.

If you have `LATEX3` syntax enabled, you can also use:

```
\datatool_db_state:nnnn{<db-name>}{<not empty>}{<empty>}{<not exists>}
```

This is a shortcut that combines `\DTLifdbexists` and `\DTLifdbempty`. If the database identified by *<db-name>* exists and is not empty, this does *<not empty>*, if it exists but is empty, this does *<empty>*. If the database doesn't exist, this does *<not exists>*.

The metadata associated with a database consists of the identifying database label, the number of columns, the number of rows, and the column metadata.

3. Databases (datatool package)

Rows in a database are only identified by an index (starting from 1). Columns may be identified by an index (starting from 1) or by a key (label) that must be unique to the database. You can test if a database has a column with a given key with:

```
\DTLifhaskey{<db-name>}{<key>}{<true>}{<false>}
```

modifier: *

This does *<true>* if the database identified by the label *<db-name>* has a column labelled with the given *<label>*, and *<false>* otherwise. The unstarred version will trigger an error if the database doesn't exist. The starred version will do *<false>* if the database doesn't exist.

If you have L^AT_EX3 syntax enabled you may instead use:

```
\datatool_if_has_key:nnTF {<db-name>} {<key>} {<true>}  
{<false>}  
\datatool_if_has_key_p:nn {<db-name>} {<key>}
```

This tests if the database with the label *<db-name>* exists and has a column with the given *<key>*. (*\DTLifhaskey** is now simply defined to use *\datatool_if_has_key:nnTF*.)

```
\DTLrowcount{<db-name>}
```

Expands to the number of rows in the database identified by the label *<db-name>*. No check is made for existence, but you will get a “Missing number, treated as zero” error if the database hasn't been defined.

```
\DTLcolumncount{<db-name>}
```

Expands to the number of columns in the database identified by the label *<db-name>*. No check is made for existence, but you will get a “Missing number, treated as zero” error if the database hasn't been defined.

The column metadata not only consists of the index and unique key, but also a header and the column data type (see §2.2). When an entry is added to a database the entry is parsed to determine its data type, and the column metadata is updated. For example, if an integer is the first element to be added to a column, the column metadata will be updated to set the column data type to integer. If a decimal is then added to that column, the metadata will be updated to “real number”. If another integer is added, the metadata won't be updated as the “real number” type takes precedence.

Some advanced commands require the column index rather than the column key. You can lookup the index from the key with:

```
\DTLgetcolumnindex{<cs>}{<db-name>}{<col key>}
```

modifier: *

3. Databases (datatool package)

This will define the control sequence `<cs>` to expand to the index of the column labelled `<col key>` for the database identified by the label `<db-name>`. The starred version doesn't check if the database or column exists. The unstarred version will trigger an error if either the database doesn't exist or doesn't have a column with the given key. Alternatively, you can use the `column index` action:

```
\DTLaction[key={<col key>}] {column index}
Index: \DTLuse{}.
Or: \DTLget{<cs>} index: \cs.
```

Note that `\DTLgetcolumnindex` is robust. If you want the column index in an expandable context you can use:

```
\dtlcolumnindex{<db-name>}{<col key>}
```

This will expand to 0 if the database or column don't exist, otherwise it will expand to the column index. (Note that prior to version 3.0, this would expand to `\relax` if the database or column didn't exist.)

If you want the reverse, that is the column key given the column index, you can use:

```
\DTLgetkeyforcolumn{<cs>}{<db-name>}{<col idx>} modifier: *
```

This gets the key for the column with the index `<col idx>` from the database identified by the label `<db-name>` and defines the control sequence `<cs>` to expand to that value. The starred version `\DTLgetkeyforcolumn*` doesn't check if the database or column exists.

The column data type is used in some commands, such as `\DTLdisplaydb` (where the column data type determines the column alignment). However, if the data isn't stored as a datum item, it will have to be reparsed for commands like `\DTLmaxforkeys`. It's more efficient to use `store-datum=true`, but if you do so, you will need to remember that a command that is set to an element value (such as those in an assignment list in `\DTLforeach`) will be a datum control sequence rather than a command whose expansion text is the original value.

You can look up the column data type with:

```
\DTLgetdatatype{<cs>}{<db-name>}{<col key>} modifier: *
```

This will define the control sequence `<cs>` to expand to the data type ID for the column labelled `<col key>` in the database identified by `<db-name>`. The starred version doesn't check if the database and column are defined. The unstarred version will trigger an error if either are undefined.

Alternatively, you can use the `column data` action, which will get the index, key, type and header from the column key or column index. For example, the following will display the meta data for the first column:

3. Databases (datatool package)

```
\DTLaction[column=1]{column data}
Column 1 key: \DTLuse{key},
title: \DTLuse{header},
type: \DTLuse{type}.
```

The following fetches the meta data for the column identified by the key Name:

```
\DTLaction[key=Name]{column data}
\DTLget[column]\colidx Column index: \colidx.
\DTLget[header]\colheader Column title: \colheader.
\DTLget[type]\coltype Column type: \coltype.
```

The data type ID will be one of: 0 (string), 1 (int), 2 (real), 3 (currency), or empty for unset (which typically means there are no non-empty elements in the column). There are commands provided that expand to the corresponding ID:

```
\DTLunsettype
```

This expands to nothing and so can be used to check for the unset ID. Note that this is different from the way that datatool-base identifies unknown types (which have an ID of -1). The other IDs have the same numeric value as those used by datatool-base.

```
\DTLstringtype
```

This expands to 0 and so can be used to check for the string ID.

```
\DTLinttype
```

This expands to 1 and so can be used to check for the integer ID.

```
\DTLrealtype
```

This expands to 2 and so can be used to check for the real number ID.

```
\DTLcurrencytype
```

This expands to 3 and so can be used to check for the currency ID.

The column header defaults to the column key, but may be changed. For example, when reading a CSV or TSV file with `\DTLread`, the headers can be set with the `headers` option. Alternatively, you can use:

```
\DTLsetheader{<db-name>}{<col key>}{<header>}
```

*modifier: **

This sets the header for the column labelled *<col key>* in the database identified by *<db-name>*. The starred version doesn't check if the database exists. The unstarred version will trigger an error if the database doesn't exist.

Normally new column metadata is automatically added when an entry is added with a new key. However, you can also add a new column with:

```
\DTLaddcolumn{<db-name>}{<col key>}
```

*modifier: **

This increments the column count for the database identified by *<db-name>* and assigns the label *<col key>*. The header is also set to *<col key>* and the data type is initialised as "unknown". This doesn't add an entry to the database. It just modifies the metadata.

If you want to add a new column with a header that isn't the same as the column key, then you use:

```
\DTLaddcolumnwithheader{<db-name>}{<col key>}{<header>} modifier: *
```

This has the same effect as:

```
\DTLaddcolumn{<db-name>}{<col key>}
\DTLsetheader{<db-name>}{<col key>}{<header>}
```

but it's shorter and slightly quicker. Alternatively:

```
\DTLaction[name={<db-name>}, key={<col key>}, value={<header>}]
{add column}
```

or set the database name as the default so you don't have to keep supplying it:

```
\DTLsetup{default-name={<db-name>}}
\DTLaction[key={<col key>}, value={<header>}] {add column}
```

Be careful about defining columns that aren't required as you can end up with null values (which isn't the same as an empty value). Example 71 defines two columns but no value is added to the second column in any of the rows:

71

```
\DTLnewdb{mydata}
\DTLaddcolumnwithheader{mydata}{name}{Name}
\DTLaddcolumnwithheader{mydata}{address}{Address}
```

```
\DTLnewrow{mydata}
\DTLnewdbentry{mydata}{name}{Zoë}
\DTLnewrow{mydata}
\DTLnewdbentry{mydata}{name}{José}
\DTLnewrow{mydata}
\DTLnewdbentry{mydata}{name}{Dickie}
% this row has an empty name:
\DTLnewrow{mydata}
\DTLnewdbentry{mydata}{name}{}

Number of rows: \DTLrowcount{mydata}.
Number of columns: \DTLcolumncount{mydata}.

\DTLdisplaydb{mydata}
```

Missing values show as “NULL” (see §3.10).

↑ Example 71: Column with No Values

Number of rows: 4. Number of columns: 2.

Name	Address
Zoë	NULL
José	NULL
Dickie	NULL
	NULL

3.7. Displaying the Contents of a Database

A database can be displayed in a tabulated form using one of the following commands.

```
\DTLdisplaydb[⟨omit list⟩]{⟨db-name⟩}
\DTLdisplaydb*[⟨options⟩]{⟨db-name⟩}
```

Displays the content of the database identified by the label $\langle db-name \rangle$ in a tabular environment. The optional argument $\langle omit list \rangle$ should be a comma-separated list of column keys (not indexes) to omit. To allow for greater flexibility, there is also a starred version where the optional argument is a $\langle key \rangle = \langle value \rangle$ list of options. If the database is large and requires multiple pages, you can use the following instead:



```
\DTLdisplaylongdb[<options>]{<db-name>}
```

This displays the data in a longtable environment (you will need to load the longtable package). The options for `\DTLdisplaydb*` and `\DTLdisplaylongdb` are listed in §3.7.1. Associated commands, which can be redefined to make slight adjustments, are listed in §3.7.2. Examples are provided in §3.7.3.

There are analogous actions: `display` (for `\DTLdisplaydb*`) and `display long` (for `\DTLdisplaylongdb`). For example:

```
\DTLaction[name={mydata},options={only-keys=
{Name,Description}}]{display}
```

or

```
\DTLsetup{default-name=mydata}
\DTLaction[options={only-keys={Name,Description}}]
{display}
```

are equivalent to:

```
\DTLdisplaydb*[only-keys={Name,Description}]{mydata}
```

As from version 3.0, both `\DTLdisplaydb` and `\DTLdisplaylongdb` use a private token list variable *<content-tl-var>* to construct the content of the tabular or longtable environment. This removes the loop from within the alignment and avoids the need to globally set variables. Once construction of *<content-tl-var>* has been completed, *<content-tl-var>* is then expanded.

The `pre-content` option value is placed immediately before *<content-tl-var>*, so you can set `pre-content` to `\show` for debugging purposes. This will show the tokens in *<content-tl-var>* in the transcript (and *<content-tl-var>* won't be expanded).

3.7.1. Display Options

Most of these options can be used with both `\DTLdisplaydb*` and `\DTLdisplaylongdb`. However, some are only available with one or other command. If you are using the `display` or `display long` actions, you can specify these options in the `options` action setting.



When datatool v2.0 introduced `\DTLdisplaylongdb`, options were defined using the xkeyval interface. In version 3.0, the xkeyval package has been dropped in favour of the newer l3keys interface. If you previously used `\setkeys{displaylong}{<options>}` to set the options (instead of using the optional argument), you will need to

```
switch to \DTLsetup{display={\langle options \rangle}}.
```

3.7.1.1. General

```
init=\langle code \rangle
```

initial: empty

The value should be code to insert after the options have been processed and before the content token list variable construction starts. This may be used to initialise variables for use in other hooks (see Example 78).

```
pre-content=\langle code \rangle
```

initial: empty

The value should be code to insert immediately before the token list containing the tabular environment. You can set the value to `\show` for debugging purposes, and it will show rather than typeset the content. Any local redefinitions within `\langle code \rangle` will be scoped. For example, this option can be used to adjust `\tabcolsep` or `longtable` settings.

You can't use `pre-content` to locally redefine customization commands, such as `\dtldisplaydbenv`, as they will have already been expanded.

```
per-row=\langle number \rangle
```

initial: 1

The number of database rows per tabular (or longtable) row. Note that the row number `\dtlrownum` is incremented per included database row and the column number `\dtlcolumnnum` is incremented per included column regardless of this option. For example, with `per-row=2` then `\dtlrownum` and the `\langle row-num \rangle` argument of `\DTLdisplaydbAddItem` will first be 1 and then 2 for the first tabular line that follows the header (not including any additional content that may have been inserted by hooks).

```
row-idx-map-inline=\langle defn \rangle
```

The value should be the definition of a command that takes a single parameter which will be the loop iteration index (1 for the first row, 2 for the second etc). The command definition should expand to the row index to be used for the current iteration. The default behaviour is a direct mapping from iteration index to the database row index.



The row index mapping function must expand to an integer between 1 and the total number of columns for the database. The function is applied before filtering.

For example, to display the contents of the database in reverse order:

```
row-idx-map-inline={\DTLrowcount{\dtldbname}-#1+1}
```

```
row-idx-map-function=<cs>
```

As above but the value is a command that takes a single argument. For convenience, `\DTLdisplayTBrowidxmap` is provided for use with `per-row` that will arrange the data rows from top to bottom instead of left to right, but note that this won't work if filtering omits rows.

```
post-row-inline=<defn>
```

A hook is provided while the tabular content is constructed at the end of each row (before the `\dtldisplaycr` separating the rows). Unlike `\dtldisplaystartrow`, which has its expansion text inserted into the content token list, the post-row hook determines whether or not to append content to the token list variable or accumulate information for later insertion. The `post-row-inline` option provides a way to define this hook inline.

The hook takes two arguments: `<content-tl-var>` (the content token list, which contains the tabular content under construction) and `<row idx>` (the database row index). If filtering has been applied, you can access the filtered row index (which is the current tabular row count excluding header and extra content) with the register `\dtlrownum`. If no rows have been filtered, `\dtlrownum` will have the same value as `<row idx>`. The hook isn't used for excluded rows.



The `\dtlcurrentrow` token register will be available within the hook, so you access row information with commands such as `\dtlgetentryfromcurrentrow` or `\DTLassignfromcurrentrow` or actions such as `current row aggregate`.

```
post-row-function=<cs>
```

As `post-row-inline` but provides the name of a function that takes two arguments.

`tabular-env=<env-name>`

default: **tabular**; initial: **tabular**

Only available with `\DTLdisplaydb*` and indicates the environment to use. This is a shortcut that redefines `\dtldisplaydbenv` to `<env-name>` but additionally checks that the given environment is defined and redefines `\dtldisplayvalign` to empty, unless `<env-name>` is `tabular` or `array`, in which case `\dtldisplayvalign` is redefined to `c` unless it is currently defined to `t` or `b`.

For example (requires the `tabularray` package):

```
\DTLdisplaydb*[tabular-env=tblr]{mydata}
```

`longtable-env=<env-name>`

default: **longtable**; initial: **longtable**

Only available with `\DTLdisplaylongdb` and indicates the environment to use. This is a shortcut that redefines `\dtldisplaylongdbenv` to `<env-name>` but additionally checks that the given environment is defined. Note that the environment needs to support longtable syntax.

For example (requires the `tabu` package):

```
\DTLdisplaylongdb[longtable-env=longtabu]{mydata}
```

3.7.1.2. Alignment

`string-align=<col-spec>`

initial: **l**

This option specifies the alignment for columns with the string data type. It simply redefines `\dtlstringalign` to `<col-spec>`.

`integer-align=<col-spec>`

initial: **r**

This option specifies the alignment for columns with the integer data type. It simply redefines `\dtlintalign` to `<col-spec>`.

`int-align=<col-spec>`

alias: `integer-align`

A synonym of `integer-align`.

`decimal-align=<col-spec>`

initial: **r**

This option specifies the alignment for columns with the decimal (real) data type. It simply redefines `\dtlrealalign` to `<col-spec>`.

`real-align=<col-spec>`

alias: `decimal-align`

A synonym of `decimal-align`.

`currency-align=<col-spec>`

initial: **r**

This option specifies the alignment for columns with the currency data type. It simply redefines `\dtlcurrencyalign` to `<col-spec>`.

`inter-col=<align-spec>`

initial: **empty**

This option specifies the inter-column alignment markup. It simply redefines `\dtlbetweencols` to `<align-spec>`. For example, `inter-col={ | }` for a vertical line.

`pre-col=<align-spec>`

initial: **empty**

This option specifies the pre-column alignment markup. It simply redefines `\dtlbeforecols` to `<align-spec>`. For example, `pre-col={ | }` for a vertical line.

`post-col=<align-spec>`

initial: **empty**

This option specifies the post-column alignment markup. It simply redefines `\dtlaftercols` to `<align-spec>`. For example, `post-col={ | }` for a vertical line.

`align-specs=<specs>`

initial: **empty**

This option is essentially a manual override. The value should be the complete column specifications for the table. If this setting has a non-empty value, the value will be used for the tabular or longtable `<align-spec>` argument. The options `string-align`, `integer-align`, `decimal-align`, `currency-align`, `inter-col`, `pre-col`, `post-col` will be ignored (although they will still set the underlying commands).

3.7.1.3. Headers and Footers

The header is essentially in the form:

3. Databases (datatool package)

```

<pre-head>
\dtlcolumnheader{<align>}{<header 1>} &
\dtlcolumnheader{<align>}{<header 2>} &
...
\dtlcolumnheader{<align>}{<header n>}
<post-head>

```

where `<pre-head>` is the expansion text of `\dtldisplaystarttab` (the same as the value of the `pre-head` option), and `<header 1>` etc are the column headers, and `<post-head>` is the value of the `post-head` option. If `\dtldisplayafterhead` (the same as the value of `after-head`) is not empty, this is then followed by:

```
\dtldisplaycr \dtldisplayafterhead
```

Each column title in the header row is aligned with:

```
\dtlcolumnheader{<align>}{<text>}
```

where `<align>` is the header column alignment. This is simply defined as:

```
\multicolumn{1}{<align>}{\dtlheaderformat{<title>}}
```

The `<align>` argument is obtained by:

```
\dtladdheaderalign <align-token-tl> {<type>}{<col-num>}{<max-cols>}
```

This adds the appropriate `<align>` to `<align-token-tl>`, taking into account the `pre-col`, `inter-col` and `post-col` settings. The default definition will use `c` (centred) regardless of the data type. (Note used if `header-row` is set.) The token list `<align-token-tl>` is cleared before calling this command, so you can either append or set it.

```
pre-head=<code>
```

initial: empty

The value should be code to insert immediate before the header. This option simply redefines `\dtldisplaystarttab` to `<code>`. Note that with `\DTLdisplaylongdb`, this will come after the caption, if provided.

```
post-head=<code>
```

initial: empty

The value should be code to insert at the end of the header (before the end of row). This option redefines `\l_datatool_post_head_tl` to `<code>`. Note that this is different to `after-head` which comes after the row end.

`after-head=<code>`

initial: *empty*

This option simply redefines `\dtldisplayafterhead`, which comes after the header, to `<code>`.

For example, to add `\midrule` (booktabs) after the header, you can either redefine `\dtldisplayafterhead` to `\midrule`:

```
\renewcommand{\dtldisplayafterhead}{\midrule}
```

or you can use `after-head`:

```
\DTLsetup{display={after-head={\midrule}}}
```

or (for a particular longtable):

```
\DTLdisplaylongdb[after-head={\midrule}]{mydata}
```

`header-row={<code>}`

initial: *empty*

This option is a manual override that may be used to explicitly set the header row. An empty value indicates the default header, which is obtained from the column metadata. For example:

```
\DTLdisplaydb*[header-row={Name & Description}]
{mydata}
```

You will need to include any required formatting and make sure you have the correct number of columns.

`no-header=<boolean>`

default: **true**; initial: **false**

A boolean option that indicates that the header should be omitted. Note that this option will not only omit the header row but also the `pre-head`, `post-head` and `after-head`.

The following options default to `\c_novalue_tl` which indicates the setting is switched off. This makes it possible to distinguish between switching off the setting, and enabling the setting but requiring an empty value. For example, `caption={ }` will create a caption with empty text (the table number will be displayed). Whereas the default `caption=\c_novalue_tl` (which would require $\text{\LaTeX 3}$ syntax to be enabled) will switch off the setting.

`caption=<text>`

(Only available with `\DTLdisplaylongdb`.) If set, `\caption` will be inserted at the start of the header with the supplied value in the argument.

`cont-caption=<text>`

(Only available with `\DTLdisplaylongdb`.) If this option is set in addition to `caption`, the `\caption` argument for the first header (`\endfirsthead`) will be obtained from `caption` but the `\caption` argument for the subsequent headers (`\endhead`) will be obtained from `cont-caption`. This option is ignored if `caption` isn't set.

`contcaption=<text>`

alias: `cont-caption`

A synonym of `cont-caption`.

`short-caption=<text>`

(Only available with `\DTLdisplaylongdb`.) If this option is set in addition to `caption`, the optional argument of `\caption` in the first header will be set to this value. This option is ignored if `caption` isn't set.

`shortcaption=<text>`

alias: `short-caption`

A synonym of `short-caption`.

`label=<label>`

(Only available with `\DTLdisplaylongdb`.) If this option is set in addition to `caption`, the `\caption` in the first header will be followed by `\label{<label>}`. This option is ignored if `caption` isn't set.

`foot=<code>`

If this option is set, the value will be inserted as the footer. In the case of `\DTLdisplaydb`, `\dtldisplaycr <code>` be inserted before `\dtldisplayendtab` at the end of the tabular environment (see `\DTLdisplaydbAddEnd`). In the case of `\DTLdisplaylongdb`, `<code>\endfoot` will be inserted at the start of the longtable environment (see `\DTLdisplaylongdbAddBegin`).

```
last-foot=<code>
```

(Only available with `\DTLdisplaylongdb`.) If this option is set, `<code>\endlast-foot` will be inserted (by `\DTLdisplaylongdbAddBegin`) as the last footer of the longtable.

```
lastfoot=<code>
```

alias: `last-foot`

A synonym of `last-foot`.

3.7.1.4. Filtering

```
row-condition-inline=<defn>
```

The value should be an inline function that takes three arguments: `<content-tl-var>` (the content token list variable), `<row-idx>` (the row index), and `<true>` which will contain the code to be performed if the row should be included. The function should expand to the third argument (`<true>`) if the row should be included and expand to nothing otherwise. The `<true>` code will include the post-row hook provided by `post-row-inline` or `post-row-function`.

If you try to reference `\dtlrownum` within the filter function before the `<true>` code, it will contain the filtered row index of the previous row to be added. The variable is incremented in the `<true>` argument.

The `<content-tl-var>` argument is the token list variable used to construct the tabular or longtable body, which the function may append content to (regardless of whether or not it expands to `<true>`), but bear in mind that the content will be before `\tabularnewline` (which is added to the content token list at the start of the code provided in the third argument of the filter function, since it's not possible to tell at the end of the previous row if there are any additional rows that won't be filtered).

For example, to omit all the odd rows:

```
\DTLdisplaydb*[row-condition-inline=
{\ifodd#2\else#3\fi}]{mydata}
```

Or to include all rows but insert a blank row before every even row:

```
\DTLdisplaydb*[row-condition-inline={
  \ifodd#2 \else \appto#1{\tabularnewline}\fi #3} ]
{mydata}
```

```
row-condition-function=<cs>
```

Instead of using an inline function with `row-condition-inline`, you can provide a function with `row-condition-function`. The supplied command `<cs>` must have three arguments as per the inline function for `row-condition-inline`.

Both filter options can access information from the current database row with current row commands such as `\dtlgetentryfromcurrentrow` as in Example 75, or an action that acts on the current row, such as `current row aggregate`.

The following options cancel each other out. Only the last listed in the options list will have effect.

```
omit-columns={ <col idx list> }
```

The value should be a comma-separated list of column indexes, indicating which columns should be omitted.

```
omit-keys={ <col key list> }
```

The value should be a comma-separated list of column keys, indicating which columns should be omitted. Note that this has replaced the now-deprecated `omit` option.

```
only-columns={ <col idx list> }
```

The value should be a comma-separated list of column indexes, indicating which columns should be included.

```
only-keys={ <col key list> }
```

The value should be a comma-separated list of column keys, indicating which columns should be included.



With `only-keys` and `only-columns`, the table column order will match the inclusion order (that is, the order given in the option value). Otherwise, the table column order will match the column index order (with excluded columns omitted, if applicable).

3.7.2. Associated Commands

The following commands are used by `\DTLdisplaydb` and `\DTLdisplaylongdb` to format the table contents.



```
\dtlheaderformat{<text>}
```

Used to format the column headers. This defaults to just `\textbf{<text>}`. Not used if `no-header` is set.



Version 3.0 has changed the definition of `\dtlheaderformat`. Previously, the definition included `\hfil` to centre the text. The alignment is now performed with `\dtlcolumnheader` and `\dtlheaderformat` just deals with any font change.



```
\dtlstringformat{<text>}
```

Used to format values in columns with the string data type. The default definition simply expands to `<text>`.



```
\dtlnumericformat{<text>}
```

Used to format values in columns with a numeric data type. This command is used by the following:



```
\dtlintformat{<text>}
```

Used to format values in columns with the integer data type. The default definition expands to `\dtlnumericformat{<text>}`.



```
\dtlrealformat{<text>}
```

Used to format values in columns with the decimal data type. The default definition expands to `\dtlnumericformat{<text>}`.

`\dtlcurrencyformat{<text>}`

Used to format values in columns with the currency data type. The default definition expands to `\dtlnumericformat{<text>}`.

The following token list commands are provided to insert extra content into the tabular or longtable body.

`\dtldisplaystarttab`

initial: empty

The expansion text of this command is inserted before the header. You can either redefine this command or use the `pre-head` option. Not used if `no-header` is set.

`\dtldisplayafterhead`

initial: empty

The expansion text of this command is inserted after the header. You can either redefine this command or use the `after-head` option. Not used if `no-header` is set.

`\dtldisplayendtab`

initial: empty

The expansion text of this command is inserted before the end of the tabular or longtable environment.

`\dtldisplaystartrow`

initial: empty

The expansion text of this command is inserted at the start of each row (except for the first row of data). Note that you can also insert content by redefining `\DTLdisplaydbAddItem` and adding a test for the first column with `\datatool_if_row_start:nnTF`, as in Example 85 in §3.7.3.14. (Content can be inserted into the end of each row with `post-row-inline` or `post-row-function`).

`\dtldisplaydbenv`

initial: tabular

This should expand to the environment name used by `\DTLdisplaydb`. Note that if you redefine this command to use an environment that doesn't take `tabular`'s optional vertical alignment argument, you will also need to redefine `\dtldisplayvalign` to expand to nothing.

You can use the `tabular-env` option instead, which will also redefine `\dtldisplayvalign`.

`\dtldisplayvalign`

initial: c

3. Databases (datatool package)

The expansion of this command should be the vertical alignment specifier for the optional argument of tabular. (Only used with `\DTLdisplaydb`.) This may be redefined to empty if the optional argument should be omitted.

For example, to switch to `tblr` (provided by `tabularray`):

```
\renewcommand{\dtldisplaydbenv}{tblr}  
\renewcommand{dtldisplayvalign}{}  

```

```
\dtldisplaylongdbenv initial: longtable  

```

This should expand to the environment name used by `\DTLdisplaylongdb`. For example, to switch to `longtabu` (provided by `tabu`):

```
\renewcommand{\dtldisplaylongdbenv}{longtabu}  

```

Alternatively, you can use the `longtable-env` option to redefine `\dtldisplaylongdbenv`.

```
\dtlstringalign initial: l  

```

The expansion of this command should be the column alignment specifier for columns with the string data type. The option `string-align` redefines this command. Not used if `align-specs` is set to a non-empty value.

```
\dtlintalign initial: r  

```

The expansion of this command should be the column alignment specifier for columns with the integer data type. The option `integer-align` (or `int-align`) redefines this command. Not used if `align-specs` is set to a non-empty value.

```
\dtlrealalign initial: r  

```

The expansion of this command should be the column alignment specifier for columns with the decimal (real) data type. The option `decimal-align` (or `real-align`) redefines this command. Not used if `align-specs` is set to a non-empty value.

```
\dtlcurrencyalign initial: r  

```

3. Databases (datatool package)

The expansion of this command should be the column alignment specifier for columns with the currency data type. The option `currency-align` redefines this command. Not used if `align-specs` is set to a non-empty value.

`\dtlbeforecols`

initial: empty

The expansion of this command is inserted at the start of the alignment specification. The option `pre-col` redefines this command. Not used if `align-specs` is set to a non-empty value.

`\dtlbetweencols`

initial: empty

The expansion of this command is inserted between columns in the alignment specification. The option `inter-col` redefines this command. Not used if `align-specs` is set to a non-empty value.

`\dtlaftercols`

initial: empty

The expansion of this command is inserted at the end of the alignment specification. The option `post-col` redefines this command. Not used if `align-specs` is set to a non-empty value.

`\dtldisplaycr`

initial: \tabularnewline

The expansion text of this command is inserted between rows. The default definition is `\tabularnewline` rather than `\\` as `\\` can interfere with paragraph columns. This allows `\dtlstringalign` to be redefined to `p` where the final column has the string data type.

The following are robust commands.

`\DTLdisplaydbAddItem{<content-tl-var>}{<item>}{<fmt-cs>}{<type>}{<row-num>}{<row-idx>}{<col-num>}{<col-idx>}`

This command is used to append an item to the `<content-tl-var>` token list variable. The `<fmt-cs>` argument is the formatting command applicable to the data type (such as `\dtlstringformat`). The default definition of this command simply appends `<fmt-cs>{<item>}` to `<content-tl-var>`. The remaining arguments are ignored by default.

The `<type>` argument is an integer representing the data type, the `<row-idx>` argument is the index to the row in the database that contains `<item>`, and the `<col-idx>` argument is the index to the column in the database that contains `<item>`.

The `<row-num>` and `<col-num>` arguments correspond to the value of the integer variables that are incremented for each included row and each included column, respectively. If you use the

3. Databases (datatool package)

default options, then these will be the same as the row index and column index, but if you change the order of the columns or exclude columns or filter out rows or change the mapping from the loop index to the row index, then they will be different.

With `per-row=1`, the $\langle row-num \rangle$ will correspond to the tabular (or longtable) row of data, excluding the header and any rows inserted by hooks. So the first row following the header will have $\langle row-num \rangle$ equal to 1. If all rows in the database are included, then this will also correspond to the database row index. However, if some rows are excluded via the filter function, then the $\langle row-num \rangle$ may be less than the corresponding row index.

Similarly, the $\langle col-num \rangle$ will correspond to the tabular (or longtable) column number. So $\langle col-num \rangle$ will be 1 for the first displayed column but this may not be the first column in the database.

It becomes more complicated if `per-row` is greater than 1, as the column number $\langle col-num \rangle$ will be reset to 1 at the start of each included database row. The row number $\langle row-num \rangle$ is incremented at the start of each included database row. So the first line of the table after the header row will start with $\langle row-num \rangle$ equal to 1 but then $\langle row-num \rangle$ will be 2 at the start of the next block of data, which is on the same line of the table.

This means that if you want to redefine `\DTLdisplaydbAddItem` to insert content at the start of a row, you can test if $\langle col-num \rangle$ is 1 if you know that `per-row=1` but if you want to take into account multiple rows of data per tabular row then you need to use the following (which requires L^AT_EX3 syntax enabled):

📌

```
\datatool_if_row_start:nnTF { $\langle row-num \rangle$ } { $\langle col-num \rangle$ } { $\langle true \rangle$ }  
{ $\langle false \rangle$ }  
\datatool_if_row_start_p:nn { $\langle row-num \rangle$ } { $\langle col-num \rangle$ }
```

If `per-row=1` then this just tests if $\langle col-num \rangle$ is 1, otherwise it will perform modulo arithmetic on $\langle row-num \rangle$ to determine if $\langle col-num \rangle$ corresponds to the first column.

The `per-row` option sets the integer variable `\l_datatool_display_per_row_int`, which may be used in the definition of `\DTLdisplaydbAddItem` for more complex requirements. (See Example 83 in §3.7.3.12.)

A token list variable $\langle align-token-tl \rangle$ is used to contain the alignment specification that will then be passed to the alignment argument of the tabular or longtable environment. For each column to be shown in the table (omitting any that have been filter by the options), the following command is used to append to $\langle align-token-tl \rangle$:

📌

```
\dtladdalign { $\langle align-token-tl \rangle$ } { $\langle type \rangle$ } { $\langle col-num \rangle$ } { $\langle max-cols \rangle$ }
```

The $\langle type \rangle$ is the numeric identifier indicating the column data type, $\langle col-num \rangle$ is the (tabular or longtable) column number (not the column index) and $\langle max-cols \rangle$ is the total number of columns to be displayed.



The `\dtladdalign` command won't be used if `align-specs` is set to a non-empty value. Instead, `\align-token-tl` will be set to the value of `align-specs`.

The `\type` argument determines whether to append the expansion text of `\dtlstringalign`, `\dtlintalign`, `\dtlrealalign` or `\dtlcurrencyalign` to `\align-token-tl`.

The last two arguments determine whether to add the expansion text of `\dtlbeforecols` (`\col-num` = 1), `\dtlaftercols` (`\col-num` = `\max-cols`) or `\dtlbetweencols` ($1 < \text{\col-num} \leq \text{\max-cols}$). Note that `\dtlaftercols` is placed after the alignment specifier, so that it occurs at the end, whereas `\dtlbeforecols` and `\dtlbetweencols` are placed before.



```
\DTLdisplaydbAddBegin{\content-tl-var}{\align-spec}{\header}
```

Used by `\DTLdisplaydb` to add the beginning of the tabular environment to the `\content-tl-var` token list variable. The `\align-spec` argument is the content of the `\align-token-tl` token list variable containing the column alignment specification (created with `\dtladdalign`), and the `\header` argument is the token list containing the header row (where each column header is encapsulated with `\dtlheaderformat`).

The tokens added to `\content-tl-var` will be in the form:

```
\begin{\tab-env}[{\v-align}]{\align-spec}  
\pre-header \header \post-head \cr  
\after-head
```

where `\tab-env` is the expansion of `\dtldisplaydbenv`, `\v-align` is the expansion text of `\dtldisplayvalign`, `\pre-header` is the expansion text of `\dtldisplaystarttab`, `\cr` is the expansion text of `\dtldisplaycr`, and `\after-head` is the expansion text of `\dtldisplayafterhead`. The `[{\v-align}]` optional argument will be omitted if `\dtldisplayvalign` is empty.



The `align-specs` option can be used to set `\align-spec` explicitly, the `header-row` option can be used to set `\header` explicitly, and the `no-header` option will omit `\pre-header` `\header` `\post-head` `\cr` `\after-head`.



```
\DTLdisplaydbAddEnd{\content-tl-var}
```

3. Databases (datatool package)

Adds the end tabular code to $\langle content-tl-var \rangle$ for $\backslash DTLdisplaydb$. This consists of:

```
 $\langle final-content \rangle \backslash end \{ \langle tab-env \rangle \}$ 
```

where $\langle final-content \rangle$ is the expansion text of $\backslash dtldisplayendtab$, and $\langle tab-env \rangle$ is the expansion of $\backslash dtldisplaydbenv$,

For example, if you want to use `tblr` (provided by `tabularray`) instead of `tabular` and you don't need all the hooks:

```
\RenewDocumentCommand\DTLdisplaydbAddBegin{mmm}{%  
  \appto#1{\begin{tblr}}{#2}#3\\}%  
}  
\RenewDocumentCommand\DTLdisplaydbAddEnd{m}{%  
  \appto#1{\end{tblr}}}%  
}
```

```
\DTLdisplaylongdbAddBegin{ $\langle content-tl-var \rangle$ }{ $\langle align-spec \rangle$ }{ $\langle header \rangle$ }
```

Analogous to $\backslash DTLdisplaydbAddBegin$ but used by $\backslash DTLdisplaylongdb$ to add the start of the longtable environment to $\langle content-tl-var \rangle$. This is more complicated than $\backslash DTLdisplaydbAddBegin$ as it also inserts the caption, label and footer, according to the `caption`, `short-caption`, `cont-caption`, `label`, `foot`, and `last-foot` options.

```
\DTLdisplaylongdbAddEnd{ $\langle content-tl-var \rangle$ }
```

Adds the end longtable code to $\langle content-tl-var \rangle$ for $\backslash DTLdisplaylongdb$. This consists of:

```
 $\langle final-content \rangle \backslash end \{ longtable \}$ 
```

where $\langle final-content \rangle$ is the expansion text of $\backslash dtldisplayendtab$.

```
\DTLdisplayTBrowidxmap{ $\langle idx \rangle$ }
```

Provided for use with `row-idx-map-function`, this command expands to a row index that will arrange data rows from top to bottom instead of left to right when `per-row` is greater than 1. Note that this is only designed to work when no rows are omitted.

The following variables require L^AT_EX3 syntax enabled and are used in some of the above commands.

3. Databases (datatool package)

`\l_datatool_caption_tl` *initial: \c_novalue_tl*

A token list assigned by the `caption` key.

`\l_datatool_short_caption_tl` *initial: \c_novalue_tl*

A token list assigned by the `short-caption` key.

`\l_datatool_cont_caption_tl` *initial: \c_novalue_tl*

A token list assigned by the `cont-caption` key.

`\l_datatool_label_tl` *initial: \c_novalue_tl*

A token list assigned by the `label` key.

`\l_datatool_foot_tl` *initial: \c_novalue_tl*

A token list assigned by the `foot` key.

`\l_datatool_last_foot_tl` *initial: \c_novalue_tl*

A token list assigned by the `last-foot` key.

`\l_datatool_post_head_tl` *initial: \c_novalue_tl*

A token list assigned by the `post-head` key.

`\l_datatool_include_header_bool` *initial: true*

A boolean variable corresponding to the inverse of the `no-header` key.

`\l_datatool_display_per_row_int`

Set by the `per-row` option.

`\l_datatool_display_tab_rows_int`

This integer variable is set to the number of rows in the database divided by `\l_datatool_display_per_row_int` rounded up. Bare in mind that this doesn't take any filtering

into account. It's used in `\DTLdisplayTBrowidxmap` to calculate the loop index to row index mapping.

3.7.3. Examples

3.7.3.1. Changing the Alignment

Example 72 uses the “product” database (see §3.2.4). This database is short enough to be produced on a single page within a tabular environment:

72

```
\DTLdisplaydb{products}
```

The database has five columns and some of the titles (in the first column) are quite long, which can lead to an overly wide table. It would be better to allow line wrapping. This can be done by changing the string column alignment (`string-align`) but this would also affect the second and third columns. In this case, it's simpler to use `align-specs` to override the default alignments for all the columns. Example 72 has:

```
\DTLdisplaydb*[align-specs={p{0.4\linewidth}llrr}]{products}
```

3.7.3.2. Omitting Columns

Example 73 uses the “product” database (see §3.2.4).

73

The unstarred version of `\DTLdisplaydb` has an optional argument that must be a comma-separated list of column keys that identify which columns to omit. For example, if I want to omit the Quantity and Price columns:

```
\DTLdisplaydb[Quantity,Price]{products}
```

With the starred version, the optional argument is a `<key>=<value>` list:

```
\DTLdisplaydb*[omit-keys={Quantity,Price}]{products}
```

Alternatively:

3. Databases (datatool package)

↑ Example 72: Display Data with Custom Alignment

Title	Author	Format	Quantity	Price (\$)
The Adventures of Duck and Goose	Sir Quackalot	paperback	3	10.99
The Return of Duck and Goose	Sir Quackalot	paperback	5	19.99
More Fun with Duck and Goose	Sir Quackalot	paperback	1	12.99
Duck and Goose on Holiday	Sir Quackalot	paperback	3	11.99
The Return of Duck and Goose	Sir Quackalot	hardback	3	19.99
The Adventures of Duck and Goose	Sir Quackalot	hardback	9	18.99
My Friend is a Duck	A. Parrot	paperback	20	14.99
Annotated Notes on the ‘Duck and Goose’ chronicles	Prof Macaw	ebook	10	8.99
‘Duck and Goose’ Cheat Sheet for Students	Polly Parrot	ebook	50	5.99
‘Duck and Goose’: an allegory for modern times?	Bor Ing	hardback	0	59.99
Oh No! The Chickens have Escaped!	Dickie Duck	ebook	11	2.0

```
\DTLaction[options={omit-keys={Quantity,Price}}]
{display}
```

These all produce the table shown in Example 73.

↑ Example 73: Display Data in a Table Omitting Columns

Title	Author	Format
The Adventures of Duck and Goose	Sir Quackalot	paperback
The Return of Duck and Goose	Sir Quackalot	paperback
More Fun with Duck and Goose	Sir Quackalot	paperback
Duck and Goose on Holiday	Sir Quackalot	paperback
The Return of Duck and Goose	Sir Quackalot	hardback
The Adventures of Duck and Goose	Sir Quackalot	hardback
My Friend is a Duck	A. Parrot	paperback
Annotated Notes on the ‘Duck and Goose’ chronicles	Prof Macaw	ebook
‘Duck and Goose’ Cheat Sheet for Students	Polly Parrot	ebook
‘Duck and Goose’: an allegory for modern times?	Bor Ing	hardback
Oh No! The Chickens have Escaped!	Dickie Duck	ebook

3.7.3.3. Column Inclusion List

Example 74 uses the “product” database (see §3.2.4).

As an alternative to the previous example, you may prefer to list the columns you want (an inclusion list). The following indicates that the Author, Title and Price columns should be include. The other columns will be omitted:

```
\DTLdisplaydb*[only-keys={Author,Title,Price}]
{products}
```

Or:

```
\DTLaction[options={only-keys={Author,Title,Price}}]
{display}
```

These all produce the table shown in Example 74.

Example 74: Display Data in a Table with Named Columns

Author	Title	Price (\$)
Sir Quackalot	The Adventures of Duck and Goose	10.99
Sir Quackalot	The Return of Duck and Goose	19.99
Sir Quackalot	More Fun with Duck and Goose	12.99
Sir Quackalot	Duck and Goose on Holiday	11.99
Sir Quackalot	The Return of Duck and Goose	19.99
Sir Quackalot	The Adventures of Duck and Goose	18.99
A. Parrot	My Friend is a Duck	14.99
Prof Macaw	Annotated Notes on the ‘Duck and Goose’ chronicles	8.99
Polly Parrot	‘Duck and Goose’ Cheat Sheet for Students	5.99
Bor Ing	‘Duck and Goose’: an allegory for modern times?	59.99
Dickie Duck	Oh No! The Chickens have Escaped!	2.0

Note that with `only-keys` and `only-columns`, the table column order will match the listed columns. Whereas with `omit-keys` and `omit-columns`, the table column order will be in the order that the columns were added to the database.

3.7.3.4. Skipping Rows

Example 75 uses the “product” database (see §3.2.4).

The `row-condition-inline` option provides a way to omit rows. If the condition is quite complicated, you may prefer to define a handler function and reference it with `row-condition-function`.

In Example 75, a command called `\productfilter` is defined which fetches the value from the “Quantity” column from the current row and only includes rows where that value is greater than zero:

```
\newcommand{\productfilter}[3]{%
  \dtlgetentryfromcurrentrow
    {\theQuantity}%
    {\dtlcolumnindex{\dtldbname}{Quantity}}%
  \DTLifnumgt{\theQuantity}{0}{#3}{}%
}
```

This command can now be used as the filter function:

```
\DTLaction[
  options={
    row-condition-function=\productfilter,
    only-keys={Title,Quantity,Price}
  }
]{display}
```

↑ Example 75: Display Data in a Table with Filtered Rows

Title	Quantity	Price (\$)
The Adventures of Duck and Goose	3	10.99
The Return of Duck and Goose	5	19.99
More Fun with Duck and Goose	1	12.99
Duck and Goose on Holiday	3	11.99
The Return of Duck and Goose	3	19.99
The Adventures of Duck and Goose	9	18.99
My Friend is a Duck	20	14.99
Annotated Notes on the ‘Duck and Goose’ chronicles	10	8.99
‘Duck and Goose’ Cheat Sheet for Students	50	5.99
Oh No! The Chickens have Escaped!	11	2.0

3.7.3.5. Referencing Rows

Example 76 uses the “marks” database (see §3.2.1).

Unlike `\DTLforeach`, there is no associated row counter for the display commands. However, you can borrow the `\DTLforeach` counters as long as there is no conflict. Example 76 does this to number and label each row. Note that labels must be unique. This can be achieved if the database has a column that contains unique values. For the marks database, this is the student number.

The `\DTLdisplaydbAddItem` hook can be redefined to increment the counter and insert the label at the start of each row. The start of the row is determined by testing if the seventh argument of `\DTLdisplaydbAddItem` (`<col-num>`) is one. The `current row values` action can be used to fetch the student number for the current row.

This can either be done using $\LaTeX$ 3 syntax:

76

```

\ExplSyntaxOn
\RenewDocumentCommand \DTLdisplaydbAddItem
{ m m m m m m m m }
{
  \int_compare:nNtT { #7 } = { \c_one_int }
  {
    \DTLaction[ return={ \StudentNo = StudentNo } ]
    { current ~ row ~ values }
    \tl_put_right:Nn #1 { \DTLrowincr \label }
    \tl_put_right:Nx #1 { { \StudentNo } }
  }
  \tl_put_right:Nn #1 { #3 { #2 } }
}
\ExplSyntaxOff

```

or with etoolbox commands:

```

\RenewDocumentCommand\DTLdisplaydbAddItem
{ m m m m m m m m }{%
  \ifnum #7 = 1
    \DTLaction[return={\StudentNo = StudentNo}]
    {current row values}%
    \appto#1{\DTLrowincr\label}%
    \eappto#1{{\StudentNo}}%
  \fi
  \appto#1{#3{#2}}%
}

```

In either case, this modification ensures that the first column of each row starts with:

```
\DTLrowincr\label{ \StudentNo }
```

where $\langle StudentNo \rangle$ is the value of the `StudentNo` entry for the current row.

Note that it's not necessary for the student number to be displayed in the table. The information is obtained by looking up the value with the `current row values` action. An alternative method is to find out which column the student number is in and insert the code into the start of that column. This can either be done by referencing the display column number argument ($\langle col-num \rangle$) or the database column index argument ($\langle col-idx \rangle$).

The counter needs to be reset before the data is displayed:

3. Databases (*datatool* package)

```
\DTLrowreset
```

The data can then be displayed:

```
\DTLaction{display}
```

and a particular row can be referenced:

```
Row \ref{103569}  
shows the details for student 103569.
```

Or if you need to look up the registration number from the student's name:

```
\DTLaction[  
  assign={  
    \Surname=Surname,  
    \Forename=Forename,  
    \StudentNo=StudentNo  
  },  
  options={  
    inline={%  
      \DTLlifstringeq{\Surname}{Brown}  
      {\DTLlifstringeq{\Forename}{Andy}{#1}}{}{}{}%  
    }  
  }  
]{find}  
Row \ref{\StudentNo}  
shows the details for Andy Brown.
```

(This method can't be used for Jane Brown, as there are two students with that name.)

↑ Example 76: Referencing Rows from Displayed Data

Surname	Forename	StudentNo	Assign1	Assign2	Assign3
Smith, Jr	John	102689	68	57	72
Brown	Jane	102647	75	84	80
Brown	Jane	102646	64	92	79
Brown	Andy	103569	42	52	54
Adams	Zoë	105987	52	48	57
Brady	Roger	106872	68	60	62
Verdon	Clare	104356	45	50	48

Row 4 shows the details for Andy Brown.

Note that this example doesn't show the value of the counter. This can be added with a slight adjustment to the code in the modified `\DTLdisplaydbAddItem` to show the counter value after it has been incremented. For the $\text{\LaTeX}$ 3 version, the change is:

```
\tl_put_right:Nn #1 { \DTLrowincr \DTLthe-  
row. ~ \label }
```

For the etoolbox version, the change is:

```
\appto#1{\DTLrowincr\DTLtherow. \label}%
```

This puts the value at the start of the surname column. Example 3.7.3.6 below places the value in a separate column.

3.7.3.6. Inserting an Extra Column at the Start

Example 77 uses the “marks” database (see §3.2.1).

The previous Example 76 incremented a counter at the start of each row with `\refstepcounter` and added a label, but the value of the counter wasn't shown, although a modification was suggested that would put the value at the start of the first column (which contains the surname).

Example 77 modifies Example 76 to insert an extra column at the start that has the counter value. This means that the alignment and header information will need to be adjusted.

First, `\DTLdisplaydbAddItem` needs to insert an extra alignment. For the $\text{\LaTeX}$ 3 code, the modification is:

3. Databases (datatool package)

```
\tl_put_right:Nn #1 { \DTLrowincr \label }
\tl_put_right:Nx #1 { { \StudentNo } }
\tl_put_right:Nn #1 { \DTLtherow & }
```

For the etoolbox version, the change is:

```
\appto#1{\DTLrowincr\label}%
\eappto#1{{\StudentNo}}%
\appto#1{\DTLtherow &}%
```

The extra column can be added to the alignment specifier using `pre-col` and a corresponding header (possibly empty) needs to be inserted into the header row:

```
\DTLrowreset
\DTLaction[
  options={
    pre-col={r},
    pre-head={\bfseries Row &}
  }
]{display}
```

↑ Example 77: Inserting a Column at the Start of Displayed Data

Row	Surname	Forename	StudentNo	Assign1	Assign2	Assign3
1	Smith, Jr	John	102689	68	57	72
2	Brown	Jane	102647	75	84	80
3	Brown	Jane	102646	64	92	79
4	Brown	Andy	103569	42	52	54
5	Adams	Zoë	105987	52	48	57
6	Brady	Roger	106872	68	60	62
7	Verdon	Clare	104356	45	50	48

Row 4 shows the details for Andy Brown.

3.7.3.7. Adding an Extra Column at the End

78

Example 78 uses the “product” database (see §3.2.4).

The previous Example 77 inserted an extra column at the start. Extra columns can be added to the end using a similar manner. Example 78 uses a slightly different approach that uses the post-row function and explicitly sets the alignment specification (`align-specs`) and appends the extra column header with `post-head`.

Example 78 has an extra column with the header “Total” that contains the value obtained by multiplying the quantity by the price. The booktabs package is required for the horizontal rules.

```
\newcommand{\productappendtotal}[2]{%
  \DTLaction[
    return={
      \theQuantity=Quantity,
      \thePrice=Price
    }
  ]{current row values}%
  \DTLmul{\theTotal}{\theQuantity}{\thePrice}%
  \DTLround{\theTotal}{\theTotal}{2}%
  \appto#1{&}%
  \eappto#1{\expandonce\theTotal}%
  \DTLadd{\runningtotal}{\runningtotal}{\theTotal}%
}
\DTLaction[
  options={
    only-keys={Title,Quantity,Price},
    pre-head={\toprule},
    after-head={\midrule},
    align-specs={lrrr},
    post-head={& \dtlcolumnheader{c}{Total}},
    init={\def\runningtotal{0}},
    post-row-function=\productappendtotal,
    foot={\midrule & & \runningtotal}
  }
]{display}
```

Example 78: Display Data in a Table with an Extra Column

Title	Quantity	Price (\$)	Total
The Adventures of Duck and Goose	3	10.99	32.97
The Return of Duck and Goose	5	19.99	99.95
More Fun with Duck and Goose	1	12.99	12.99
Duck and Goose on Holiday	3	11.99	35.97
The Return of Duck and Goose	3	19.99	59.97
The Adventures of Duck and Goose	9	18.99	170.91
My Friend is a Duck	20	14.99	299.80
Annotated Notes on the ‘Duck and Goose’ chronicles	10	8.99	89.90
‘Duck and Goose’ Cheat Sheet for Students	50	5.99	299.50
‘Duck and Goose’: an allegory for modern times?	0	59.99	0.00
Oh No! The Chickens have Escaped!	11	2.0	22.00
			1,123.96

3.7.3.8. Altering Individual Cell Formatting

Example 79 uses the “balance” database (see §3.2.6).

The “balance” database contains numeric data. Since the numbers will be repeatedly parsed, it’s best to switch on the `store-datum` setting. A default name for the database can be set at the same time:

```
\DTLsetup{store-datum,default-name=balance}
```

Suppose now that the negative numbers should be shown in red. This can be done by redefining `\dtlrealformat`:

```
\renewcommand{\dtlrealformat}[1]{\DTLiflt{#1}{0}{\color{red}}}{#1}
```

An alternative method is to redefine `\DTLdisplaydbAddItem` to perform the test for a particular column, in this case column 4 (the balance). This also conveniently provides a way to total the incoming and outgoing columns (columns 2 and 3). Note that some cells are empty so these are skipped.

```

\newcommand{\theInTotal}{0}
\newcommand{\theOutTotal}{0}
\RenewDocumentCommand \DTLdisplaydbAddItem
{ m m m m m m m m }
{%
  \DTLifnulloreempty{#2}{}% skip null or empty
  {%
    \dtlifnumeq{#8}{4}% balance column
    {%
      \DTLifnumlt{#2}{0}{\appto#1{\color{red}}}{}}%
    }%
    {%
      \dtlifnumeq{#8}{2}% in column
      {\DTLadd{\theInTotal}{\theInTotal}{#2}}%
      {%
        \dtlifnumeq{#8}{3}% out column
        {\DTLadd{\theOutTotal}{\theOutTotal}{#2}}}%
      }%
    }%
    \appto#1{#3{#2}}%
  }%
}

```

The totals can then be appended with the `foot` option. As with the previous example, I've added rules provided by booktabs:

```

\DTLaction[options={
  after-head={\midrule},
  foot=
{\midrule Totals & \theInTotal & \theOutTotal &}
}
]{display}

```

Note that the eighth argument of `\DTLdisplaydbAddItem` (*col-idx*) is the column index, which will always be an integer, so an integer comparison can be used instead of the decimal `\dtlifnumeq`. (Likewise for the *type*, *row-num*, *row-idx* and *col-num* arguments.) Since the `store-datum` option has been set, the second argument (*item*) will be in the datum item format except where it's empty or null, so the actual numeric value can be obtained with `\datatool_datum_value:Nnnnn`, which will require $\text{\LaTeX}3$ syntax and the *type* argument can be checked to ensure that the supplied value is numeric. If you switch on $\text{\LaTeX}3$ syntax, you may as well directly use the `l3int` and `l3fp` commands:

```

\ExplSyntaxOn
\fp_new:N \l_my_in_total_fp
\fp_new:N \l_my_out_total_fp
\newcommand{\theInTotal}{
  \fp_to_decimal:N \l_my_in_total_fp
}
\newcommand{\theOutTotal}{
  \fp_to_decimal:N \l_my_out_total_fp
}
\RenewDocumentCommand \DTLdisplaydbAddItem
{ m m m m m m m m }
{
  \int_compare:nNnT { #4 } > { \c_datatool_string_
int }
  {
    \int_case:nn { #8 }
    {
      { 2 }
      {
        \fp_add:Nn \l_my_in_total_fp
          { \datatool_datum_value:Nnnnn #2 }
      }
      { 3 }
      {
        \fp_add:Nn \l_my_out_total_fp
          { \datatool_datum_value:Nnnnn #2 }
      }
      { 4 }
      {
        \fp_compare:nNnT
          { \datatool_datum_value:Nnnnn #2 } <
        { \c_zero_fp }
        {
          \tl_put_right:Nn #1 { \color { red } }
        }
      }
    }
    \tl_put_right:Nn #1 { #3 { #2 } }
  }
}
\ExplSyntaxOff

```

3. Databases (datatool package)

Note that this relies on the data being stored as datum items. If you're not sure if this is the case, you can use `\DTLparse`, which will check and convert if required.

Example 79: Adjusting the Item Hook to Calculate Totals and Show Negative Numbers in Red

Description	in (£)	Out (£)	Balance (£)
Travel expenses		230	-230
Conference fees		400	-630
Grant	700		70
Train fare		70	0
Totals	700	700	

3.7.3.9. Two Database Rows Per Tabular Row (Left to Right)

Example 80 uses the “scores” database (see §3.2.2).

To make it clearer how the data is arranged, Example 80 sorts the data by surname and then first name. Since the data contains UTF-8 characters, localisation support is used:

```
\usepackage[locales=en]{datatool}
```

This requires `datatool-english`, which needs to be installed separately. The sorting is performed with the `sort` action:

```
\DTLaction[assign={surname, forename}]{sort}
```

In order to save space, you may want two database rows per tabular row. This can be done with the `per-row` option.

```
\DTLaction[
  options={
    per-row=2,
    only-keys={forename, surname, score}
  }
]{display}
```

3. Databases (datatool package)

This results in the data tabulated with (below the header) database row one (Zoë Adams) and two (Roger Brady) on the first line, three (Andy Brown) and four (Jane Brown) on the second line, five (Quinn Ó Coinn) and six (Evelyn O’Leary) on the third line, and seven (John Smith, Jr) and eight (Clare Vernon) on the fourth line. That is, the data is arranged from left to right, shifting down a line every other block of data. For a top to bottom arrange, see Example 81.

Example 80: Display Two Database Rows Per Tabular Row

First Name	Surname	Score (%)	First Name	Surname	Score (%)
Zoë	Adams	52	Roger	Brady	58
Andy	Brown	42	Jane	Brown	75
Quinn	Ó Coinn	91	Evelyn	O’Leary	81.5
John	Smith, Jr	68	Clare	Vernon	45

3.7.3.10. Two Database Rows Per Tabular Row (Top to Bottom)

Example 81 uses the “scores” database (see §3.2.2).

Example 81 is an alternative to Example 80 that has two database rows per tabular row, but they are now arranged from top to bottom instead of from left to right.

This example won’t work if rows are omitted.

Example 81 is as Example 80 except that it uses the `row-idx-map-function` option to change the mapping from the loop index to the selected row index:

```
\DTLaction[
  options={
    per-row=2,
    row-idx-map-function=\DTLdisplayTBrowidxmap,
    only-keys={forename,surname,score}
  }
]{display}
```



Example 81: Display Two Database Rows Per Tabular Row (Top to Bottom)




First Name	Surname	Score (%)	First Name	Surname	Score (%)
Zoë	Adams	52	Quinn	Ó Coinn	91
Roger	Brady	58	Evelyn	O’Leary	81.5
Andy	Brown	42	John	Smith, Jr	68
Jane	Brown	75	Clare	Vernon	45

3.7.3.11. Stripy Table

Example 82 uses the “marks” database (see §3.2.1).

The new row command (`\tabularnewline` or `\\`) is inserted at the start of each row, except for the first row. This is done after filtering (applied with `row-condition-inline` or `row-condition-function`). It can’t be done at the end of the previous row as there’s no way of telling at that point if there will be another row.

This means that although you can use the filter function to insert code into the content token list variable, that code will be inserted at the end of the previous row before the new line command. Therefore, if you want to insert content at the start of a row, it needs to be done in the `\DTLdisplaydbAddItem` command. The seventh argument of that command is the tabular (or longtable) column number. This may be different from the eighth argument, which is the database column index. This means that if you want to insert content at the start of a row, you need to test if the seventh argument is 1.

This example assumes the default `per-row=1`. See Example 83 for `per-row=2`.

The default definition of `\DTLdisplaydbAddItem` uses a `\TeX3` `\tl` command:

```
\NewDocumentCommand \DTLdisplaydbAddItem
{ m m m m m m m m }
{
  \tl_put_right:Nn #1 { #3 { #2 } }
}
```

The first argument is the content token list variable, the second argument is the column content $\langle item \rangle$ and the third argument is the formatting command $\langle fmt-cs \rangle$. The above definition simply appends $\langle fmt-cs \rangle \{ \langle item \rangle \}$ to the token list. This is equivalent to:

3. Databases (datatool package)

```
\NewDocumentCommand \DTLdisplaydbAddItem
{ m m m m m m m m }
{\appto#1{#3{#2}}}
```

Example 82 creates a stripy table with rows alternately coloured blue and green. The `colortbl` package provides `\rowcolor`:

```
\usepackage{colortbl}
```

In order to insert `\rowcolor` at the start of a row, `\DTLdisplaydbAddItem` needs to be redefined to test if the seventh argument is equal to one. The fifth argument is the row number (excluding the header), so the simplest way to alternate is to test if that value is odd or even. For example:

```
\ExplSyntaxOn
\RenewDocumentCommand \DTLdisplaydbAddItem
{ m m m m m m m m }
{
  \int_compare:nNnT { #7 } = { \c_one_int }
  { % first column
    \int_if_odd:nTF { #5 }
    { % odd row
      \tl_put_right:Nn #1 { \rowcolor { blue } }
    }
    { % even row
      \tl_put_right:Nn #1 { \rowcolor { green } }
    }
  }
  \tl_put_right:Nn #1 { #3 { #2 } }
}
\ExplSyntaxOff
```

The data can simply be displayed using the `display` action:

```
\DTLaction{display}
```


↑ Example 82: Display Data in a Stripy Table

Surname	Forename	StudentNo	Assign1	Assign2	Assign3
Smith, Jr	John	102689	68	57	72
Brown	Jane	102647	75	84	80
Brown	Jane	102646	64	92	79
Brown	Andy	103569	42	52	54
Adams	Zoë	105987	52	48	57
Brady	Roger	106872	68	60	62
Verdon	Clare	104356	45	50	48

3.7.3.12. Stripy Two Database Rows Per Tabular Row

Example 83 uses the “scores” database (see §3.2.2).

Suppose now that you want both a stripy table (like Example 82) and two database rows per tabular row (like Example 80). Simply adding the option `per-row=2` to Example 82 will cause a problem as the $\langle col-num \rangle$ argument of `\DTLdisplaydbAddItem` will loop round, which will result in `\rowcolor` being inserted in the middle of the tabular row.

Example 83 adjusts the redefinition of `\DTLdisplaydbAddItem` to use `\datatool_if_row_start:nnT` instead of simply testing if the seventh argument is 1. Bear in mind that the $\langle row-num \rangle$ argument will also increment mid tabular row as a new row of data is fetched. This means that $\langle row-num \rangle$ will always have an odd value at the start of each tabular row, so it's now not as simple as testing if $\langle row-num \rangle$ is odd:

```
\ExplSyntaxOn

\RenewDocumentCommand \DTLdisplaydbAddItem
{ m m m m m m m m }
{
  \datatool_if_row_start:nnT { #5 } { #7 }
  { % first column

    \int_compare:nNnTF
      { \int_mod:nn #5 2 * \l_datatool_display_
per_row_int }
      = \c_one_int
    {
```

3. Databases (datatool package)

```

\dtl_put_right:Nn #1 { \rowcolor { blue } }
}
{
\dtl_put_right:Nn #1 { \rowcolor { green } }
}
}
\dtl_put_right:Nn #1 { #3 { #2 } }
}
\ExplSyntaxOff

```

The data is display with:

```

\DTLaction[
options={
per-row=2,
only-keys=forename,surname,score
}
]{display}

```

Example 83: Display Stripy Two Database Rows Per Tabular Row

First Name	Surname	Score (%)	First Name	Surname	Score (%)
Jane	Brown	75	John	Smith, Jr	68
Quinn	Ó Coinn	91	Evelyn	O'Leary	81.5
Zoë	Adams	52	Clare	Vernon	45
Roger	Brady	58	Andy	Brown	42

3.7.3.13. Two Fields in One Column

Example 84 uses the “marks” database (see §3.2.1).

The marks database has a Surname and a Forename column. Example 84 redefines `\DTLdisplaydbAddItem` to show both the surname and forename in the same column. The other columns show the student registration (which disambiguates the students with the same name) and the assignment marks. This requires the header row to be set with `header-row=t` to override the default (which would only show “Surname” in the first column):

```

\DTLaction[
  options={
    only-keys=
    {Surname,StudentNo,Assign1,Assign2,Assign3},
    header-row=
    {Name & Reg.\_No., & Mark 1 & Mark 2 & Mark 3}
  }
]{display}

```

The `\DTLdisplaydbAddItem` command is redefined to test for the first column (as in the earlier Example 3.7.3.11). Remember that the tabular column number is in the seventh argument, whereas the database column index is in the eighth argument. In this case, the Surname column is both the first column in the table and also the first column in the database. The example tests if the column number is equal to one and, if true, fetches the forename from the current row (using the `current row values` action), and stores it in one of the public scratch token list variables (`\l_tmpa_tl`). This uses $\text{\LaTeX}$ 3 commands, so the syntax needs to be switched on temporarily.

```

\ExplSyntaxOn
\RenewDocumentCommand \DTLdisplaydbAddItem
{ m m m m m m m m m }
{
  \int_compare:nNnTF { #7 } = { \c_one_int }
  {
    \DTLaction[ return={ \l_tmpa_tl = Forename } ]
    { current ~ row ~ values }
    \datatool_if_null_or_empty:NTF \l_tmpa_tl
    {
      \tl_put_right:Nn #1 { #3 { #2 } }
    }
    {
      \tl_put_right:Nx #1
      {
        \exp_not:N #3 { \exp_not:n { #2 } ,
          ~ \exp_not:V \l_tmpa_tl }
      }
    }
  }
}
{
  \tl_put_right:Nn #1 { #3 { #2 } }
}

```

```

    }
  }
\ExplSyntaxOff

```

The test for null or empty (see §3.10) isn't necessary in this example, as there are no null values in the example database. It's provided so that the example can be adapted for other databases.



This example has nested actions. The new definition of `\DTLdisplaydbAddItem` uses the `current row values` action and this will be used within the `display` action. However, the underlying display function introduces a scope to limit the effects of the display options, but that scoping also prevents the inner action from interfering with the return values of the outer action.

Note that the scratch variable must be expanded before being added to the content token list variable as its value changes at each iteration. However, if there's a possibility that full expansion will cause a problem for the formatting command, surname or forename then they should be protected from expansion (which is achieved with `\exp_not:N`, `\exp_not:n` and `\exp_not:V` in the above).

If you prefer not to use L^AT_EX3 commands, the above can be rewritten with `\appto`, `\eappto` and `\expandonce`, which are provided by `etoolbox`, `\noexpand` (a T_EX primitive) and `\unexpanded` (an ϵ -T_EX primitive).



```

\RenewDocumentCommand \DTLdisplaydbAddItem
{ m m m m m m m m }
{
  \ifnum #7 = 1
    \DTLaction[return={\Forename=Forename}]
      {current row values}
    \DTLifnulllorempty{\Forename}%
      {%
        \appto#1{#3{#2}}}%
      {%
        \eappto#1{\noexpand#3{\unexpanded{#2}},
          \expandonce\Forename}}%
      }%
    \else
      \appto#1{#3{#2}}}%
    \fi
}

```

Example 84: Display Two Fields in One Column

Name	Reg. No.,	Mark 1	Mark 2	Mark 3
Smith, Jr, John	102689	68	57	72
Brown, Jane	102647	75	84	80
Brown, Jane	102646	64	92	79
Brown, Andy	103569	42	52	54
Adams, Zoë	105987	52	48	57
Brady, Roger	106872	68	60	62
Verdon, Clare	104356	45	50	48

3.7.3.14. Calculations, Filtering and Row Highlighting

Example 85 uses the “marks” database (see §3.2.1).

This example combines elements from previous examples to show the student surname and forename in the same column (as Example 84), calculates the average score which is appended in an extra column (similar to Example 78), filters out the rows where the average is below 50 (similar to Example 75) and highlights the rows where the average is above 70 (similar to Example 3.7.3.11).

The `colortbl` package is needed for this example as it uses `\rowcolor` to highlight rows.

```
\usepackage{colortbl}
```

Since this example will perform arithmetic calculations and comparisons, the marks database is loaded with the `store-datum` setting on.

```
\DTLsetup{store-datum,default-name=marks}
\DTLread{studentmarks.csv}
```

The row condition may be used to skip rows, in a similar way to Example 75 but in this case the average needs to be calculated, which can be done with the `current row aggregate` action.

```
\newcommand{\rowfilter}[3]{%
  \DTLaction[
```

```

options={mean}, datum={round=1},
keys={Assign1-},
return={\AverageScore=mean}
]{current row aggregate}% calculate average
\DTLifnumlt{\AverageScore}{50}%
{}% skip if average less than 50
{#3}%
}

```

The redefinition of `\DTLdisplaydbAddItem` is similar to that for Example 84 described in §3.7.3.13. However, a check is added to determine if the average is above 70 so that `\rowcolor` can be added to the content. Note that it can't be added in the custom `\rowfilter` hook as it will end up before `\tabularnewline`, which will be inserted at the start of the code provided in the third argument of the filter function.



If you want to redefine `\DTLdisplaydbAddItem` in order to insert content at the start of a row, make sure to test if the seventh argument (the tabular column number) is 1 rather than the eighth argument (the database column index).

The definition used in §3.7.3.13 already tests the column number rather than the column index, so only a small addition is required. Note that since the `datum` action option was set, the result (`\AverageScore`) will be a datum control sequence so the actual numerical value can be obtained as a plain number with `\DTLdatumvalue`, which means that `\fp_compare:nNnT` can be used instead of `\DTLifgt`. With $\text{\LaTeX 3}$ commands, the definition is now:



```

\ExplSyntaxOn
\RenewDocumentCommand \DTLdisplaydbAddItem
{ m m m m m m m m }
{
  \int_compare:nNnTF { #7 } = { \c_one_int }
  {
    % insert highlight if average greater than 70
    \fp_compare:nNnT
      { \DTLdatumvalue \AverageScore } > { 70 }
      {
        \tl_put_right:Nn #1 { \rowcolor {yellow} }
      }
    \DTLaction[ return={\l_tmpa_tl=Forename} ]
      { current ~ row ~ values }
  }
}

```

3. Databases (datatool package)

```
\datatool_if_null_or_empty:NTF \l_tmpa_tl
{
  \tl_put_right:Nn #1 { #3 { #2 } }
}
{
  \tl_put_right:Nx #1
  {
    \exp_not:N #3 { \exp_not:n { #2 } ,
      ~ \exp_not:V \l_tmpa_tl }
  }
}
{
  \tl_put_right:Nn #1 { #3 { #2 } }
}
\ExplSyntaxOff
```

Alternatively, if you prefer to use the etoolbox commands:



```
\RenewDocumentCommand \DTLdisplaydbAddItem
{ m m m m m m m m }
{
  \ifnum #7 = 1
  % highlight if average greater than 70
  \DTLifnumgt{\AverageScore}{70}%
  {\appto#1{\rowcolor{yellow}}}{}%
  \DTLaction[return={\Forename=Forename}]
  {current row values}
  \DTLifnulloreempty{\Forename}%
  {\appto#1{#3{#2}}}%
  {%
    \eappto#1{\noexpand#3{\unexpanded{#2}},
      \expandonce\Forename}}%
  }%
  \else
  \appto#1{#3{#2}}%
  \fi
}
```

The `post-row-function` can be used to append the average (which should still be available with the custom `\AverageScore` calculated in the above) in a similar manner to

Example 78.

```
\newcommand{\appendaverage}[2]{%
  \appto#1{\&}%
  \eappto#1{\expandonce\AverageScore}%
}
```

The data is again displayed with the `display` action, but this time there are only three columns: the student's name, the student number and the average score. As with Example 78, the tabular alignment specification must be provided with `align-specs`.

```
\DTLaction[
  options={
    only-keys={Surname,StudentNo},
    align-specs={lrr},
    post-row-function=\appendaverage,
    row-condition-function=\rowfilter,
    header-row={Name & Reg.\_No. & Average}

  }

]{display}
```

↑ Example 85: Displaying Data with Calculations, Filtering and Row

Highlighting

Name	Reg. No.	Average
Smith, Jr, John	102689	65.7
Brown, Jane	102647	79.7
Brown, Jane	102646	78.3
Adams, Zoë	105987	52.3
Brady, Roger	106872	63.3

3.8. Iterating Through a Database



Iteration is problematic within tabular-like environments. If you really need to have a loop within a tabular-like environment, use `\DTLforeach` not `\DTLmapdata`. However, be aware of its limitations. Consider constructing the tabular contents to move the loop outside of the tabular body. (See §3.9.)

The newer command, `\DTLmapdata`, is described in §3.8.1. The older command `\DTLforeach` is described in §3.8.2. Both can be used to iterate over rows of a database and both have a setting that allows the database to be edited. In the case of `\DTLmapdata`, the edits are locally added to a pending buffer and only applied at the end of `\DTLmapdata` and may be discarded with `\DTLmapdatabreak*`. Note that it's also possible to edit a database outside of a loop. See §3.12 for those commands.

3.8.1. Iterating Over Rows with `\DTLmapdata`



```
\DTLmapdata [⟨key=value list⟩] {⟨loop-body⟩}
```

Iterates over the data in a database, performing `⟨loop-body⟩` at each iteration. The available options are:



```
name=⟨value⟩
```

Identifies the database. If omitted, the default name is assumed (as given by the `default-name` option).



```
read-only=⟨boolean⟩
```

default: **true**; initial: **true**

If true, this setting switches on read-only mode. If false, the database may be edited using the commands described in §3.8.1.2, such as `\DTLsetentry` and `\DTLrmrow`. Note that the current row editing commands for use with `\DTLforeach` are not compatible with `\DTLmapdata`.



```
allow-edits=⟨boolean⟩
```

default: **true**; initial: **false**

This option is an antonym of `read-only`. If true, this setting switches off read-only mode.

The `⟨loop-body⟩` argument of `\DTLmapdata` (which may contain paragraph breaks) is performed at each iteration. Unlike `\DTLforeach`, which has an optional argument to

3. Databases (*datatool* package)

implement filtering, if you want to skip a row, you will need to add the appropriate conditional within the loop body.

If you have a long loop, you may prefer to use an environment instead.



```
\begin{DTLenvmapdata} [ <key=value list> ]  
<content>  
\end{DTLenvmapdata}
```

This just uses `\DTLmapdata` on the environment body. See Example 193 in §9.6.1 which uses `DTLenvmapdata` for mail merging with the supplementary person package.



The `DTLenvmapdata` environment automatically trims leading and trailing spaces from the loop body, but `\DTLmapdata` doesn't.

In both cases, the command `\dtldbname` will be defined to expand to the database name, and, at the start of each iteration, the current row number is stored in the `\dtlrownum` register. Unlike `\DTLforeach`, there is no associated row counter. However, you can borrow the `\DTLforeach` counters as long as there is no conflict. This simplest method is to use `\DTLrowreset` before the start of `\DTLmapdata` and use `\DTLrowincr` in the loop body. Alternatively, you can define your own counter.

The loop may be prematurely terminated with:



```
\DTLmapdatabreak
```

*modifier: **

The loop will terminate at this point, not at the end of the current iteration. (This is different to breaking out of `\DTLforeach` with `\dtlbreak`, which will cause the loop to terminate at the end of the current iteration.)

With the default `read-only=true`, there's no difference between the starred and unstarred version of `\DTLmapdatabreak`. Otherwise, the starred version will discard any edits as well as breaking out of the loop, whereas the unstarred version will break out of the loop but still apply the edits.

Note that since `\DTLmapdatabreak` will skip all following content in the rest of the loop iteration, it can't be placed within a primitive `\if... \fi` conditional as the closing `\fi` will be lost. Similarly, it can't occur within a non-expandable or scoped context.



If you need to nest `\DTLmapdata`, you must scope the inner loop to prevent conflict. Note that `\begin` implicitly creates a group, so the `DTLenvmapdata` environment will automatically be scoped.

Example 86 uses the student scores database (see §3.2.2) and simply iterates over each row, printing the database name and row index. The loop will terminate on the third row. Note that

 86

3. Databases (datatool package)

the environment body has the leading and trailing spaces trimmed so “(after break)” at the end of one loop runs into “scores” at the start of the next.

```
Command: \DTLmapdata{
  \dtldbname: row \the\dtlrownum.
  \dtlifnumeq{\dtlrownum}{3}{\DTLmapdatabreak}{}
  (after break)
}

Environment:
\begin{DTLenvmapdata}
  \dtldbname: row \the\dtlrownum.
  \dtlifnumeq{\dtlrownum}{3}{\DTLmapdatabreak}{}
  (after break)
\end{DTLenvmapdata}
```

↑ Example 86: Iterating Over Rows with \DTLmapdata and DTLenvmapdata

Command: scores: row 1. (after break) scores: row 2. (after break)
scores: row 3.

Environment: scores: row 1. (after break)scores: row 2. (after
break)scores: row 3.

3.8.1.1. Accessing Data in the Current Row of \DTLmapdata

Within the loop body of \DTLmapdata, you can access values in the current iteration row using:

```
\DTLmapget { <key=value list> }
```

The argument is a <key>=<value> list. Allowed options are listed below. Either **key** or **column** is required to identify the value by its column label or index. If both are used, they will override each other.

```
key=<value>
```

The value should be the label used to identify the required column.

```
column=⟨value⟩
```

The numeric value should be the index used to identify the required column.

```
return=⟨cs⟩
```

If set to a non-empty value, the value should be a command which will be defined to the value obtained from the column (identified by `key` or `column`). If the value is empty or omitted (the default), the value will simply be inserted into the document.

If `return=cs` is set, you can test if `⟨cs⟩` is null with `\DTLifnull`. For example:

```
\begin{DTLenvmapdata}
Dear
\DTLmapget{key=title,return=\Title}% fetch title
\DTLifnull\Title{}{\Title}% ignore title if not set
\DTLmapget{key=forename} \DTLmapget{key=surname}

% ...
\end{DTLenvmapdata}
```

In the above, if the title isn't set, it will be omitted, but if the forename or surname columns aren't set, they will appear as "NULL". (Note that a null value is not the same as an empty value, see §3.10.)

If you used the `store-datum` option to store values in the database as datum items, then `⟨cs⟩` will be a datum control sequence. You will then be able to use datum commands such as `\DTLdatumvalue` and `\DTLdatumtype`. (See §2.2 for further details.) Similarly for `⟨cs⟩` in `\DTLmaprow`, and likewise for assignments in `\DTLforeach` and similar commands.

Rather than repeatedly using `\DTLmapget` for each column, you can instead use:

```
\DTLmaprow{⟨cs⟩}{⟨body⟩}
```

This command may only be used within the loop body of `\DTLmapdata` (or `DTLenvmapdata`) and iterates over the columns in the current (`\DTLmapdata`) iteration row. At the start of each iteration, the register `\dtlcolumnnum` is set to the column index and `⟨cs⟩` is defined to the column value. Then `⟨body⟩` is done. You can prematurely break the loop with:



```
\DTLmaprowbreak
```

This will stop the current `\DTLmaprow` loop at that point but won't stop the `\DTLmapdata` loop. As with `\DTLmapdatabreak`, this command should not be placed inside a primitive conditional or within a non-expandable or scoped context.

Alternatively, if you prefer the `<cs>=<col-key>` syntax used with `\DTLforeach`:



```
\DTLmapgetvalues{<assign-list>}
```

*modifier: **

The `<assign-list>` argument should be a comma-separated list of `<cs>=<col-key>` where `<cs>` is a command and `<col-key>` is the label identifying the required column.

If there is no value matching the given column key then the placeholder command `<cs>` will be assigned to null (which can be tested with `\DTLifnull`). The unstarred `\DTLmapgetvalues` will trigger an error if the database has no column defined with the given label `<col-key>` (as opposed to the column being defined for the database but no value set in that column for the given row), whereas the starred `\DTLmapgetvalues*` for won't and will simply set `<cs>` to null.

For example:



```
\DTLmapgetvalues
{\thePrice=Price,\theQuantity=Quantity}
```

This is essentially equivalent to:



```
\DTLmapget{key=Price,return=\thePrice}
\DTLmapget{key=Quantity,return=\theQuantity}
```

3.8.1.2. Editing Rows

If `read-only=false` (or `allow-edits=true`), the following commands are available for use in the loop body of `\DTLmapdata` (or within the body of the `DTLenvmapdata` environment).

All edits are saved in a local buffer and are only applied to the database at the end of `\DTLmapdata`. This means that any change applied within a scope in the loop body will be lost once the scope is ended. However, the final update to the database at the end will be applied according to the current `global` setting. Pending edits can be discarded by breaking out of the loop with `\DTLmapdatabreak*`.



The final updates to the database are only performed after the loop has finished. For example, if the database has 4 rows at the start of the loop and 1 row is removed with `\DTLrmrow`, then `\DTLrowcount` will still expand to 4 until `\DTLmapdata` has completed. Edits to the current iteration row will be picked up by any following `\DTLmapget`, but `\DTLmaprow` will only pick up any changes made before the `\DTLmaprow` loop starts.

Some of the edit commands may take a `<key>=<value>` argument. Common options are:



`column=<column-index>`

Identifies a column by its numeric index. The column does not have to exist if appending is allowed, but the index must be greater than 0.



`key=<column-key>`

Identifies a column by its key. Some commands may allow both `column` and `key`. If the column exists, they must both reference the same column. If the column doesn't exist, then `column` is taken as the column reference and `key` provides a key for that column if a new column needs to be defined.



`value=<value>`

Identifies a value, where one is required.



`expand-value=<value>`

As `value` but will fully expand the value.



`expand-once-value=<value>`

As `value` but will expand the value once.



`\DTLrmrow`

Removes the current iteration row. Note that this won't affect the row indexes of the following iterations or the total row count, as the edit won't take effect until the loop terminates.



`\DTLrmentry{<key=value list>}`

3. Databases (datatool package)

Removes the entry in the column identified by either `column` or `key`. Note that this won't delete the entire column but any reference to this column for the given row will result in a null value (see §3.10).

For example:

```
\DTLmapdata[allow-edits]{\DTLrmentry{column4}}
```

This will remove the entry in column 4 for every row, but column 4 will still be defined.

```
\DTLsetentry{<key=value list>}
```

Sets the column identified by either `column` or `key` to the value identified by `value` or `expand-value` or `expand-once-value`. Note that the general `new-value-expand` and `store-datum` settings will be respected, but the expansion with `expand-value` or `expand-once-value` occurs before the `new-value-expand` setting is implemented.

If the current iteration row already has a value in the given column, the existing value will be replaced with the new value. If the current row doesn't have a value in the given column, the new value will be added. If the desired column doesn't exist, you will need to identify it with `column`. If you also set `key` that will be used as the label for the new column, otherwise it will be given the default label (obtained from `\dtldefaultkey <col-idx>`).

Since the database doesn't get updated until the loop has terminated, you can use `column={\DTLcolumncount{\dtldbname}+1}` to append a column. However, it's simpler to create the new column first, as in Example 87.

Example 87 loads the “marks” database (see §3.2.1) and uses `\DTLmapdata` to iterate over each row and append a column containing the average marks for each assignment. The `row aggregate` action may be used within `\DTLmapdata` to calculate the average. For example:

```
\DTLaction[key=Average]{add column}
\DTLmapdata[allow-edits]{%
  \DTLaction[
    keys={Assign1,Assign2,Assign3},
    options={mean},
    datum={round=1},% round the result
    return={\Mean=mean}
  ]
}
```

87

3. Databases (datatool package)

```
{row aggregate}
\DTLifnull{\Mean}% test the return value
{}% row aggregate failed!
{% average calculated successfully
  \DTLsetentry{key=Average,expand-value=\Mean}
}
}
```

A range may be used in `keys=`, which may be more convenient:

```
\DTLaction[
  keys={Assign1-Assign3},
  options={mean},
  datum={round=1},% round the result
  return={\Mean=mean}
]
```

Since the new column will have a null value at this point (which will be skipped by the `row aggregate` action), an open ended range may be used instead:

```
\DTLaction[
  keys={Assign1-},
  options={mean},
  datum={round=1},% round the result
  return={\Mean=mean}
]
```

The database can then be sorted by the average mark and displayed:

```
\DTLaction[assign={{Average=desc}}]{sort}
\DTLaction{display}
```


Example 87: Iterating Over Rows with `\DTLmapdata` to Append a

Column	Surname	Forename	StudentNo	Assign1	Assign2	Assign3	Average
	Brown	Jane	102647	75	84	80	79.7
	Brown	Jane	102646	64	92	79	78.3
	Smith, Jr	John	102689	68	57	72	65.7
	Brady	Roger	106872	68	60	62	63.3
	Adams	Zoë	105987	52	48	57	52.3
	Brown	Andy	103569	42	52	54	49.3
	Verdon	Clare	104356	45	50	48	47.7

3.8.2. Iterating Over Rows with `\DTLforeach`

The newer command `\DTLmapdata`, described in §3.8.1, locally sets internal variables at each iteration. This means that it can't be used within a tabular-like environment where the loop body contains `&` or `\` as internal variables will be lost when the scope changes between columns and rows.

The commands and environments in this section are older and have extra overhead. They were designed to work in tabular-like environments, so they perform global assignments to workaround the automatic scoping within cells. However, alignment can be sensitive to certain commands occurring at the start of a cell or row that can trigger “misplaced `\noalign`” errors. In which case, it may be better to use the method suggested in §3.9.

`\DTLforeach` [*condition*] { *db-name* } { *assign-list* } { *body* } *modifier*: *

Iterates over each row of the database identified by *db-name*. This command may be nested up to three times. The starred version is read-only and therefore faster. The unstarred version allows the database to be edited within the loop.

Corresponding environments are also available.

`\begin{DTLenvforeach}` [*condition*] { *db-name* } { *assign-list* }
content
`\end{DTLenvforeach}`

This is equivalent to:

`\DTLforeach` [*condition*] { *db-name* } { *assign-list* } { *content* }

```
\begin{DTLenvforeach*} [⟨condition⟩] {⟨db-name⟩} {⟨assign-list⟩}
⟨content⟩
\end{DTLenvforeach*}
```

This is equivalent to:

```
\DTLforeach* [⟨condition⟩] {⟨db-name⟩} {⟨assign-list⟩} {⟨content⟩}
```

Any commands applicable to `\DTLforeach` are also applicable to the corresponding `DTLenvforeach` environment.

Verbatim content can't be used in `⟨body⟩` for either the command or environment.

After `\DTLforeach` (or `DTLenvforeach`), the final loop index may be saved with:

```
\DTLsavelastrowcount {⟨cs⟩}
```

This will locally define `⟨cs⟩` to the final index of the last `\DTLforeach` loop. Note that this refers to the counter increment at each iteration where the `⟨condition⟩` evaluates to true, and so may not be the database row count. If used within the body of a `\DTLforeach` loop, it will refer to the last `\DTLforeach` loop at the next level up. Where the level index is obtained from the following:

```
\dtlforeachlevel
```

The current level register is incremented at the start of `\DTLforeach` and corresponds to the level index. This will be 1 in the outermost `\DTLforeach`, 2 within the first nested `\DTLforeach`, and 3 within the second nested `\DTLforeach`. Each level has an associated row counter, which is incremented with `\refstepcounter`, which means you can use `\label` at the start of `⟨body⟩` in order to reference a row number, but obviously some method will be needed to make the label unique, see Example 90 in §3.8.2.2.3.

Ideally `\label` should occur as soon as possible after `\refstepcounter`, and it must be within the same scope. Therefore if you need to label the rows, ensure that `\label` occurs right at the start of `⟨body⟩`.

DTLrowi

3. Databases (datatool package)

The DTLrowi counter keeps track of the current row for the first level.

DTLrowi

No

The DTLrowii counter keeps track of the current row for the second level.

DTLrowii

No

The DTLrowiii counter keeps track of the current row for the third level.

i

The counter corresponding to the current level is only incremented for rows where the *<condition>* is satisfied. This means that if the counter is referenced in *<condition>* it will still have its value from the previous row.

DTLrow

No

To avoid duplicate hypertargets with hyperref, there is a level 0 counter DTLrow that is incremented at the start of every `\DTLforeach`. This ensures that `\theHDTLrowi`, `\theHDTLrowii` and `\theHDTLrowiii` expand to unique values.

These counters may be used in other contexts, such as with `\DTLmapdata`, but take care that such use doesn't conflict with `\DTLforeach`. Bear in mind that these counters don't have automatic resets (that is, they are not defined with a parent counter). For convenience, the following commands are provided to minimise conflict.

`\DTLrowreset`

Pin

Increments DTLrow and resets the counter DTLrow $\langle n \rangle$ where $\langle n \rangle$ is one more than `\dtlforeachlevel`.

`\DTLrowincr`

Pin

Increments the counter DTLrow $\langle n \rangle$ where $\langle n \rangle$ is one more than `\dtlforeachlevel`.

`\DTLtherow`

Pin

Uses `\theDTLrow $\langle n \rangle$` where $\langle n \rangle$ is one more than `\dtlforeachlevel`.

At the start of each iteration, the assignments given in *<assign-list>* are made. This argument should be a *<cs>*–*<col-key>* comma-separated list, where *<cs>* is a command and *<col-key>* is the key uniquely identifying a column.



Each placeholder command $\langle cs \rangle$ will be **globally** defined to the value in the column identified by $\langle col-key \rangle$ (or null, if not set) of the current row. There is no test to ensure that the command isn't already defined.

The $\langle condition \rangle$ provided in the optional argument is tested using `\ifthenelse` (so the commands in §2.4.2 may be used). If the condition evaluates to true, the row counter for the current level is incremented, and the $\langle body \rangle$ of the loop is performed. The placeholder commands in $\langle assign-list \rangle$ may be referenced within the $\langle condition \rangle$. If the optional argument is omitted, `\boolean{true}` is assumed.

The following commands are defined for use in $\langle body \rangle$. Additionally, the `\dtlcurrentrow` token register is set to the special internal database markup for the current row. Commands or actions that are designed for use with that register, such as the `current row aggregate` action, may be used in $\langle body \rangle$, but avoid the editing commands described in §3.16.1 that alter `\dtlcurrentrow`. Instead, use the commands described in §3.8.2.1 to modify the current row.



```
\DTLcurrentindex
```

This will expand to the numeric value of the current row counter.



```
\dtlbreak
```

If used within $\langle body \rangle$, this command will cause the loop to terminate at the end of the current iteration. Note that this is different to `\DTLmaprowbreak`, which breaks out of the current loop immediately.



```
\DTLiffirstrow{ $\langle true \rangle$ }{ $\langle false \rangle$ }
```

Expands to $\langle true \rangle$, if this row is the first (that satisfies the condition). That is, if the corresponding `DTLrow $\langle I \rangle$` counter is 1. Otherwise, it expands to $\langle false \rangle$. Bear in mind that this tests the loop index not the database row index.



```
\DTLiflastrow{ $\langle true \rangle$ }{ $\langle false \rangle$ }
```

Expands to $\langle true \rangle$, if this row is the last (that satisfies the condition). Prior to version 3.0, this would never evaluate to $\langle true \rangle$ if any rows had been filtered. As from version 3.0, this compares the corresponding `DTLrow $\langle I \rangle$` counter with the total number of rows in the database less the number of omitted rows.



```
\DTLifoddrow{ $\langle true \rangle$ }{ $\langle false \rangle$ }
```

Expands to *<true>*, if the loop index (that is, the value of the corresponding DTLrow*<I>* counter) is odd.

`\DTLforeachkeyinrow` Iterates over each column in the current row of `\DTLforeach`, globally defines *<cs>* to the value in that column, and does *<code>*. This internally iterates through the column metadata with `\dtlforeachkey`, assigning `\dtlkey`, `\dtlcol`, `\dtltype` and `\dtlheader` to the column key, column index, column type and column header, respectively.

3.8.2.1. Editing the Current Row within `\DTLforeach`

If the read-only `\DTLforeach*` is used, no changes can be made to the database. With the editable unstarred version, the current row may be changed using the commands below, and the database will be updated after each iteration. This makes it slower than the read-only version.



This is a different approach to the `allow-edits` mode for `\DTLmapdata`, which stores pending edits in a buffer and only updates the database after the loop has terminated.

With the editable unstarred `\DTLforeach`, the following commands may be used to modify the current row of the database. These commands will trigger an error with the read-only version.



These commands don't follow the `global`, `new-value-expand` or `store-datum` settings. They always perform a global change, expand the given value, if applicable, and don't use the datum format.



```
\DTLappendtorow{<col-key>}{<value>}
```

Appends an entry with the protected expansion of *<value>* to the current row for the column identified by *<col-key>*. The row must not already have that column set.



```
\DTLreplaceentryforrow{<col-key>}{<value>}
```

Replaces the entry for the column identified by *<col-key>* in the current row with the protected expansion of *<value>*.



```
\DTLremoveentryfromrow{<col-key>}
```

Removes the entry in the column identified by *<col-key>* from the current row. An error will occur if the given column isn't set.

`\DTLremovecurrentrow`

Removes the current row from the database. Note that this is different from removing all entries from the row with `\DTLremoveentryfromrow`, which would result in a blank row.

3.8.2.2. Examples

Most of these examples can now be performed with either `\DTLdisplaydb*` or `\DTLmapdata`.

3.8.2.2.1. Displaying Data

Example 88 is an alternative to Example 74 (see §3.7.3.3) that uses `\DTLforeach*` instead of `\DTLdisplaydb`. Note that the use of `\DTLdisplaydb` is simpler and less problematic (see §3.9).

88

```
\begin{tabular}{llr}
\bfseries Author & \bfseries Title & 
\bfseries Price (\$)%
\DTLforeach*{products}%
{\Author=Author,\Title=Title,\Price=Price}%
{\% start new row
 \Author & \Title & \Price
}
\end{tabular}
```

Note that the new row command `\%` has been placed at the start of the final argument. This is necessary as placing it at the end of the argument will cause an unwanted row at the end of the table. This is a feature of the loop mechanism.

3.8.2.2.2. Stripy Table

Example 89 is an alternative to Example 82 in §3.7.3.11 that alternately colours the rows of data blue or green.

89

Take care when using a conditional to deal with any alignment. The conditional must be expandable. In this case, `\DTLifoddrow` is used, which is expandable.

This example requires the `colortbl` package which provides `\rowcolor`.

↑ Example 88: Display Data in a Table with \DTLforeach

Author	Title	Price (\$)
Sir Quackalot	The Adventures of Duck and Goose	10.99
Sir Quackalot	The Return of Duck and Goose	19.99
Sir Quackalot	More Fun with Duck and Goose	12.99
Sir Quackalot	Duck and Goose on Holiday	11.99
Sir Quackalot	The Return of Duck and Goose	19.99
Sir Quackalot	The Adventures of Duck and Goose	18.99
A. Parrot	My Friend is a Duck	14.99
Prof Macaw	Annotated Notes on the 'Duck and Goose' chronicles	8.99
Polly Parrot	'Duck and Goose' Cheat Sheet for Students	5.99
Bor Ing	'Duck and Goose': an allegory for modern times?	59.99
Dickie Duck	Oh No! The Chickens have Escaped!	2.0

```
\begin{tabular}{llrrrr}
\bfseries Surname & \bfseries Forename &
\bfseries StudentNo & \bfseries Assign1 &
\bfseries Assign2 & \bfseries Assign3%
\DTLforeach*{marks}
{\Surname=Surname, \Forename=Forename,
 \StudentNo=StudentNo, \AssignI=Assign1,
 \AssignII=Assign2, \AssignIII=Assign3}
{%
 \DTLifoddrow{\\rowcolor{blue}}{\\rowcolor{green}}
 \Surname & \Forename &
 \StudentNo & \AssignI &
 \AssignII & \AssignIII
}%
\end{tabular}
```

↑ Example 89: Using `\DTLforeach` to Display a Stripy Table

Surname	Forename	StudentNo	Assign1	Assign2	Assign3
Smith, Jr	John	102689	68	57	72
Brown	Jane	102647	75	84	80
Brown	Jane	102646	64	92	79
Brown	Andy	103569	42	52	54
Adams	Zoë	105987	52	48	57
Brady	Roger	106872	68	60	62
Verdon	Clare	104356	45	50	48

Example 85 in §3.7.3.14, which highlights rows after computing the average score, may seem similar to this example, but `\DTLifnumlt` is a robust command and will therefore cause a problem. Either use the method provided in Example 85 or in Example 93.

3.8.2.2.3. Displaying the Data with an Extra Column at the Start

Example 90 is an alternative to Example 77 that inserts the row index and label in a separate column at the start.

90

```
\begin{tabular}{rllrrrr}
\bfseries Row & \bfseries Surname &
\bfseries Forename & \bfseries StudentNo &
\bfseries Assign1 & \bfseries Assign2 &
\bfseries Assign3%
\DTLforeach*{marks}
{\Surname=Surname, \Forename=Forename,
 \StudentNo=StudentNo, \AssignI=Assign1,
 \AssignII=Assign2, \AssignIII=Assign3}
{%
 \label{\StudentNo}
 \\ \theDTLrowi &
 \Surname & \Forename &
 \StudentNo & \AssignI &
 \AssignII & \AssignIII
}%
\end{tabular}
```

Note that the `\label` must occur in the same scope as `\refstepcounter`. This means that it has to go before the new row `\\` which automatically starts a new scope.

As with Example 77, the student number for the referenced row is obtained with the `find` action.

Example 90: Displaying Data with Row Numbers Using `\DTLforeach`

Row	Surname	Forename	StudentNo	Assign1	Assign2	Assign3
1	Smith, Jr	John	102689	68	57	72
2	Brown	Jane	102647	75	84	80
3	Brown	Jane	102646	64	92	79
4	Brown	Andy	103569	42	52	54
5	Adams	Zoë	105987	52	48	57
6	Brady	Roger	106872	68	60	62
7	Verdon	Clare	104356	45	50	48

Row 4 shows the details for Andy Brown.

3.8.2.2.4. Displaying the Data with an Extra Column at the End

Example 91 is an alternative to Example 78 (see §3.7.3.7) that uses `\DTLforeach*` instead of `\DTLdisplaydb` to display the data with an extra column showing the total (quantity times price) as well as the running total at the end. Note that the scoping introduced by the cells in the tabular environment means that the running total must be globally updated.

91

As with Example 78, the `booktabs` package is used for horizontal rules at the start and end of the table.

```
\newcommand{\runningtotal}{0}
\begin{tabular}{lrrr}
\toprule
\bfseries Title & \bfseries Quantity & 
\bfseries Price & \bfseries Total%
\DTLforeach*{products}% database
{\theTitle=Title, \theQuantity=Quantity,
 \thePrice=Price}%
{%
\DTLiffirstrow{\midrule}{\theTitle &
\theQuantity & \thePrice & 
\DTLmul{\theTotal}{\theQuantity}{\thePrice}%
\DTLround{\theTotal}{\theTotal}{2}%
\DTLgadd{\runningtotal}{\runningtotal}{\theTotal}%
\theTotal
}%
\midrule & & & \runningtotal
\end{tabular}
```

Note that this example has a conditional (`\DTLiffirstrow`) that determines whether to just start a new row with `\` or to start a new row and insert a horizontal rule with `\midrule`.

rule. If you need something like this in your document, the conditional command must be expandable otherwise you will get an error (see §3.9).

↑ Example 91: Using \DTLforeach to Display Data in a Table with a Running Total Column

Title	Quantity	Price	Total
The Adventures of Duck and Goose	3	10.99	32.97
The Return of Duck and Goose	5	19.99	99.95
More Fun with Duck and Goose	1	12.99	12.99
Duck and Goose on Holiday	3	11.99	35.97
The Return of Duck and Goose	3	19.99	59.97
The Adventures of Duck and Goose	9	18.99	170.91
My Friend is a Duck	20	14.99	299.80
Annotated Notes on the ‘Duck and Goose’ chronicles	10	8.99	89.90
‘Duck and Goose’ Cheat Sheet for Students	50	5.99	299.50
‘Duck and Goose’: an allegory for modern times?	0	59.99	0.00
Oh No! The Chickens have Escaped!	11	2.0	22.00
			1,123.96

3.8.2.2.5. Editing Rows

Example 92 is an alternative to Example 87 (see §3.8.1.2) that uses \DTLforeach instead of \DTLmapdata to edit the database. With \DTLmapdata, the row aggregates are computed with the `row aggregate` action, but with \DTLforeach the `current row aggregate` action must be used instead.

92

```

\DTLforeach{marks}{}{
  \DTLaction[
    keys={Assign1-},
    options={mean},
    datum={round=1},% round the result
    return={\Mean=mean}
  ]
  {current row aggregate}
  \DTLifnull{\Mean}% test the return value
  {}% row aggregate failed!
  {% average calculated successfully
    \DTLappendtorow{Average}{\Mean}%
  }
}

```

The database can then be sorted by the average mark and displayed:

```
\DTLsortdata{marks}{Average=desc}
\DTLdisplaydb{marks}
```

↑ Example 92: Editing a Database with \DTLforeach

Surname	Forename	StudentNo	Assign1	Assign2	Assign3	Average
Brown	Jane	102647	75	84	80	79.7
Brown	Jane	102646	64	92	79	78.3
Smith, Jr	John	102689	68	57	72	65.7
Brady	Roger	106872	68	60	62	63.3
Adams	Zoë	105987	52	48	57	52.3
Brown	Andy	103569	42	52	54	49.3
Verdon	Clare	104356	45	50	48	47.7

3.9. Loops and Conditionals with tabular-like Environments

It can be problematic using loops and conditionals with the tabular environment or any similar alignment environment, such as `longtable`. Each cell within a row has implicit scoping. This allows you to conveniently use declarations, such as `\bfseries`, at the start of a cell without affecting the rest of the row. For example:

```
\begin{tabular}{cr}
\bfseries A & B \\
\itshape C & D \\
\end{tabular}
```

In the above, only “A” is bold and only “C” is italic because of the implicit scope that localises the effect of the font change.

Suppose now that you want to keep track of the row numbers. Using the older $\text{\TeX}$ route, you could define a count register and increment it at the start of each row. However a local increment will only have an effect within the current scope. For example:



```

\newcount\myrowctr
\begin{tabular}{cr}
A B (0)
C D (0)
\advance\myrowctr by 1\relax
A & B (\the\myrowctr)\\
\advance\myrowctr by 1\relax
C & D (\the\myrowctr)
\end{tabular}

```

The local increment within a cell means that the change is lost. You can prefix `\advance` with `\global` to make the change global, but a higher-level more $\text{\LaTeX}$ solution is to define a counter and increment it with `\stepcounter` (or `\refstepcounter`), which will make the change global. For example:



```

\newcounter{myrowctr}
\begin{tabular}{cr}
A B (1)
C D (2)
\stepcounter{myrowctr}
A & B (\themyrowctr)\\
\stepcounter{myrowctr}
C & D (\themyrowctr)
\end{tabular*}

```

This is what `\DTLforeach` does and, to allow it to be nested, there are three counters (`DTLrowi`, `DTLrowii` and `DTLrowiii`) associated with each level. These counters are incremented globally (with `\refstepcounter`). The placeholder commands also need to be globally defined, as well as various other commands that need to change at each iteration.

This means that `\DTLforeach` can, to a limited extent, be used within a tabular-like context but `\DTLmapdata`, which only locally defines or changes variables, can't. This means that `\DTLmapdata` can be scoped to prevent any local assignments or changes conflicting with other content, whereas `\DTLforeach` can't.

Problems arise when more complex operations are needed at the start of a row or cell that interferes with the column alignment. Suppose now that I want to include a test at the start of each row to insert a horizontal rule above every other row:



```

\newcounter{myrowctr}
\newcommand{\myrowhook}{%
  \stepcounter{myrowctr}%
  \ifthenelse{\isodd{\value{myrowctr}}}{\hline}%
}
\begin{tabular}{cr}

```

3. Databases (*datatool* package)

```
\myrowhook A & B (\themyrowctr) \\
\myrowhook C & D (\themyrowctr)
\end{tabular}
```

This now causes a “misplaced `\noalign`” error.

The best way to avoid this in complex situations where the tabular content requires various conditionals to adjust rows or cells is to construct the content in a temporary command (a token list variable) so that the problematic code is shifted outside of the tabular environment. This is the method now used by `\DTLdisplaydb` and `\DTLdisplaylongdb`.

Using a more traditional $\text{\LaTeX 2}_{\epsilon}$ method, you can define the variable with `\newcommand` and append to it using commands provided by `etoolbox`. For example, first define the counter and the command that will be used to store the content of the tabular environment:

```
\newcounter{myrowctr}
\newcommand{\mytabcontent}{}%
```

Provide a command to initialise the above (in case multiple tables are needed):

```
\newcommand{\InitTabContent}{%
  \renewcommand{\mytabcontent}{\begin{tabular}{cr}}%
  \setcounter{myrowctr}{0}%
}
```

Provide a command to finish off:

```
\newcommand{\FinishTabContent}{%
  \appto\mytabcontent{\end{tabular}}%
}
```

Now a command is needed to add a row. Note that the row separator `\\` needs to be inserted at the end of all but the last row, which is the same as inserting it at the start of all but the first row. So this first checks if the counter is 0 before incrementing it to determine whether or not to insert `\\`. A horizontal line is inserted every even row (after the counter has been incremented). Alternatively, this could be modified to insert `\hline` before the counter is incremented when it's odd. The two arguments provided are appended without expansion to the content (separated by `&`). However, the value of the row counter must be expanded when it's added so `\eappto` is required.

3. Databases (datatool package)

```
\newcommand{\AddRow}[2]{%
  \ifthenelse{\value{myrowctr}=0}{}{\app-
to\mytabcontent{\}}%
  \stepcounter{myrowctr}%
  \ifthenelse{\isodd{\value{myrowctr}}}{}{%
    {\appto\mytabcontent{\hline}}% even row
  \appto\mytabcontent{#1 & #2}%
  \eappto\mytabcontent{ (\themyrowctr)}%
}
```

Once the custom commands have been defined, they can be used:

```
\InitTabContent
\AddRow{A}{B}% add first row
\AddRow{C}{D}% add second row
\FinishTabContent
```

This doesn't actually display the tabular content, but at the end of the above, the replacement text of the custom command `\mytabcontent` now contains the content of the tabular environment without the awkward conditionals. You can confirm this with:

```
\show\mytabcontent
```

This will show the content in the transcript or terminal when you run $\LaTeX$:

```
> \mytabcontent=macro:
->\begin {tabular}{cr}A & B (1)\hline C & D
(2)\end {tabular}.
```

You can now go ahead and put `\mytabcontent` where you want your table.

If you prefer to use $\LaTeX 3$, you can instead use the commands provided by the `l3tl` package (token lists) and the `l3int` package (integers), which are now part of the $\LaTeX$ kernel so they don't need loading. For example, first switch on $\LaTeX 3$ syntax and define the variables:

```
\ExplSyntaxOn
\int_new:N \l_my_row_int % new integer variable
\tl_new:N \l_my_content_tl % new token list variable
```

The custom command to initialise these variables is now provided as a document command, which

3. Databases (*datatool* package)

is robust:

```
\NewDocumentCommand \InitTabContent { }
{
  \tl_set:Nn \l_my_content_tl { \begin{tabular} { cr }
}
  \int_zero:N \l_my_row_int
}
```

Similarly for the command that finishes off:

```
\NewDocumentCommand \FinishTabContent { }
{
  \tl_put_right:Nn \l_my_content_tl { \end{tabular} }
}
```

And for the command that adds values to the first and second columns of the current row:

```
\NewDocumentCommand \AddRow { m m }
{
  \int_if_zero:nF { \l_my_row_int }
  { % row index not zero
    \tl_put_right:Nn \l_my_content_tl { \\ }
  }
  \int_incr:N \l_my_row_int
  \int_if_even:nT { \l_my_row_int }
  { % row index even
    \tl_put_right:Nn \l_my_content_tl { \hline }
  }
  \tl_put_right:Nn \l_my_content_tl { #1 & #2 }
  \tl_put_right:Nx \l_my_content_tl
  { ~ ( \int_use:N \l_my_row_int ) }
}
\ExplSyntaxOff
```

The rest is as before:

```
\InitTabContent
\AddRow{A}{B}% add first row
```

3. Databases (datatool package)

```
\AddRow{C}{D}% add second row
\FinishTabContent
```

This may seem very convoluted, and certainly is for the above trivial case, but when using a loop to construct the contents, it can avoid the headaches of misplaced `\noalign` errors.

Example 93 uses `\DTLmapdata` to iterate over the data in the “marks” database (see §3.2.1). The average mark is calculated using the `row aggregate` action and a row is only appended if the average is 50 or above. Any row that has an average above 70 is highlighted using `\rowcolor` (which requires the `colortbl` package).

93

Example 93 is provided to demonstrate the technique described in this chapter. The same output can be more compactly achieved using the hooks provided with `\DTLdisplaydb` (which internally applies a similar method). See Example 85 in §3.7.3.14, which produces almost the same but has an extra column showing the student number.

First some variables are defined:

```
\ExplSyntaxOn
\int_new:N \l_my_row_int
\tl_new:N \l_my_content_tl
\tl_new:N \l_my_forename_tl
\tl_new:N \l_my_surname_tl
\tl_new:N \l_my_mean_tl
```

The initialisation command is the same as before but now its an internal command:

```
\cs_new:Nn \my_init_content:
{
  \tl_set:Nn \l_my_content_tl { \begin{tabular}
{ lr } }
  \int_zero:N \l_my_row_int
}
```

Similarly for the command to finish off:

```
\cs_new:Nn \my_finish_content:
{
  \tl_put_right:Nn \l_my_content_tl { \end{tabular} }
```



```
}
```

The command that adds a row is different. The first argument will be the student's name and the second argument will be the average score. These will be supplied with placeholder commands so the values will need to be expanded when they are appended to the token list variable. The command starts off with a test to determine if the mean is greater than or equal to 50. This test is actually done in reverse (that is, the code is only done if $\text{mean} < 50$ is false).

```
\cs_new:Nn \my_add_row:nn
{
  \fp_compare:nNnF { #2 } < { 50 }
  {
    \int_if_zero:nF { \l_my_row_int }
    {
      \tl_put_right:Nn \l_my_content_tl { \\ }
    }
    \int_incr:N \l_my_row_int
    \fp_compare:nNnT { #2 } > 70
    {
      \tl_put_right:Nn \l_my_content_tl
        { \rowcolor { yellow } }
    }
    \tl_put_right:Nx \l_my_content_tl { #1 & #2 }
  }
}
```

The document command iterates over the default database. (Alternatively, you can adapt this to provide an argument with the database name.) The surname and forename are fetched using `\DTLmapgetvalues` and the mean is obtained with the `row aggregate` function. Be careful with actions with spaces in their name when you have $\text{\LaTeX}3$ syntax on as spaces are ignored. You will need to use `~` where a space must actually occur.

```
\NewDocumentCommand { \meanscorestab }
{
  \my_init_content:
  \DTLmapdata
  {
    \DTLmapgetvalues
    {
      \l_my_surname_tl = Surname ,
```

```

        \l_my_forename_tl = Forename
    }
\DTLaction
[
    keys={Assign1-},
    datum={round=1},
    return={ \l_my_mean_tl = mean },
    options=mean
]
{ row ~ aggregate }
\my_add_row:nn

{ \l_my_forename_tl \c_space_tl \l_my_surname_tl }
  { \l_my_mean_tl }
}
\my_finish_content:
% expand the content:
\l_my_content_tl
}
\ExplSyntaxOff

```

This custom command can now be used in the document at the point where the table is required.

The “marks” database used by Example 93 has two Jane Browns who both have an average above 70, so both their rows are highlighted in yellow. The students with an average below 50 aren’t listed.

↑ Example 93: Loops and Alignment

John Smith, Jr	65.7
Jane Brown	79.7
Jane Brown	78.3
Zoë Adams	52.3
Roger Brady	63.3

3.10. Null Values

Certain commands, such as `\DTLmapget`, provide a way to fetch a value from a row in a database. Return values for actions can also be fetched using the `return` option or with `\DTLget`. In either case, if you try to fetch a value that *hasn’t been set* then you will get a *null value*.



A null value is not the same as an empty value. Empty values can be tested using etoolbox's `\ifdefempty` or similar.

For example, the “customers” database described in §3.2.3 can either be read from a CSV file or defined in the document. However, there is a slight difference between the two approaches.

When data is loaded from a CSV file, null values can only occur where final columns are missing. When data is loaded from a DTLTEX or DBTEX file (see §§3.15.1.2 & 3.15.1.3) or when the database is constructed in the document (see §§3.3.1.1 & 3.4) then null values can also occur in intermediate columns.

The `customers.csv` data has null values where the final columns have been completely omitted in a row (that is, there are no trailing separators with empty elements between or after them), but the missing mid-column values are empty because there’s no way of completely omitting them without interfering with the CSV syntax. So those values are empty, not null. See the examples in §3.10.1.

If you have assigned a value to a placeholder command, for example with `\DTLmapget-values` or with the `return` option in `\DTLmapget`, then you can test if that command represents a null value with:



```
\DTLifnull{<arg>}{<true>}{<false>}
```

This will do `<true>` if the first argument is a single token that represents null and `<false>` otherwise. Note that if the argument is empty or is a command whose expansion text is empty, this will do `<false>`, because null isn’t the same as empty.

If you want to test for null or empty, use:



```
\DTLifnullorempy{<arg>}{<true>}{<false>}
```

This will do `<true>` if the argument is empty or if the argument is a single token whose value is empty or represents null, otherwise it will do `<false>`.



No trimming is performed in the null or empty tests.

Null values will cause different output to empty values when displaying data (see the examples in §3.10.1), but they can also produce different results in other functions, such as sorting (see Example 101), where a null value can trigger alternative behaviour but an empty value won’t.

3.10.1. Examples

Example 94 loads the customers database (see §3.2.3) from the CSV file `customers.csv`.

94

3. Databases (datatool package)

```
\DTLsetup{default-name=customers}  
\DTLread{customers.csv}
```

This file has some missing final columns but also has some empty values. For example:

```
5,, Duck, Dickie, dd@example.com,
```

This has an *empty* value in the second column (Organisation, a string column) and in the final column (Age, a numeric column). If these values are fetched from the database, they will expand to empty. By way of contrast, the next line is missing the final column:

```
6, Newt Fellowship, Axolotl, Lizzie, la@example.com
```

The difference is subtle (there's no trailing comma) but in this case, if the age is fetched from this row, a null value will be returned.

The data is displayed in tabular form using the `display` action:

```
\DTLaction{display}
```

String null values show as “NULL” and numeric null values show as 0. Whereas empty values show nothing.

↑ Example 94: CSV Data Containing Empty Cells and Missing Final Cells

Empty values show as a blank. Missing values show as NULL in string columns and 0 in numeric columns.

Id	Organisation	Surname	Forename	Email	Age
1		Parrot	Polly	pp@example.com	42
2	University of Somewhere	Canary	Mabel	mc@example.com	0
3	University of Somewhere	Zebra	Zoë	zz@example.com	21
4	Zinnia Florestry	Arara	José	ja@example.com	42
5		Duck	Dickie	dd@example.com	
6	Newt Fellowship	Axolotl	Lizzie	la@example.com	0
7	Avian Emporium	Canary	Fred	fc@example.com	19
8	Newt Fellowship		Molgina	m@example.com	0
9		Mander	Sally	NULL	0
10	Élite Emporium	Fant	Eli	ef@example.com	101

Example 95, in contrast, constructs the database in the document with actions (see §3.2.3). In this case, the syntax does allow for mid-columns to be omitted when setting the values for a row. For example:

95

3. Databases (datatool package)

```
\DTLaction[
  assign={
    Id = 5, Surname = {Duck}, Forename = {Dickie},
    Email = dd@example.com
  }
]{new row}
```

In this case, the Organisation and Age column aren't set for this row. This means that if an attempt is made to fetch those values, null will be returned. Compare this with another row:

```
\DTLaction[
  assign={
    Id = 9, Organisation = {},
    Surname = {Mander}, Forename = {Sally}
  }
]{new row}
```

Here, the Age and Email columns are missing but the Organisation column is set to empty. This means that if an attempt is made to fetch the Age or Email from this row, null will be returned, but an attempt to fetch the Organisation will return an empty value.

↑ Example 95: Constructed Data With Missing (Null) Values

Empty values show as a blank. Missing values show as NULL in string columns and 0 in numeric columns.

Id	Organisation	Surname	Forename	Email	Age
1	NULL	Parrot	Polly	pp@example.com	42
2	University of Somewhere	Canary	Mabel	mc@example.com	0
3	University of Somewhere	Zebra	Zoë	zz@example.com	21
4	Zinnia Florestry	Arara	José	ja@example.com	42
5	NULL	Duck	Dickie	dd@example.com	0
6	Newt Fellowship	Axolotl	Lizzie	la@example.com	0
7	Avian Emporium	Canary	Fred	fc@example.com	19
8	Newt Fellowship	NULL	Molgina	m@example.com	0
9		Mander	Sally	NULL	0
10	Élite Emporium	Fant	Eli	ef@example.com	101

Example 96 modifies Example 95 to provide a custom command that tests if its argument is null and will show a dash if so, otherwise it just does the argument:

96

```
\newcommand{\checkmissing}[1]{\DTLifnull{#1}{---}{#1}}
```

The `display` action uses the same underlying function as `\DTLdisplaydb` which encapsulates strings with `\dtlstringformat`, integers with `\dtlintformat`, decimals with `\dtlrealformat` and currency with `\dtlcurrencyformat`. The numeric formatting commands all internally use `\dtlnumericformat`, so only `\dtlstringformat` and `\dtlnumericformat` need to be redefined to use the custom command:

```
\renewcommand{\dtlstringformat}[1]{\checkmissing{#1}}
\renewcommand{\dtlnumericformat}[1]{\checkmissing{#1}}
```

Note that the empty Organisation field is still shown as empty not as a dash. Use `\DTLifnulloreempty` if you want to check for empty as well.

Example 96: Display Data With Missing (Null) Values Shown as a Dash

Missing values are shown as a dash.

Id	Organisation	Surname	Forename	Email	Age
1	—	Parrot	Polly	pp@example.com	42
2	University of Somewhere	Canary	Mabel	mc@example.com	—
3	University of Somewhere	Zebra	Zoë	zz@example.com	21
4	Zinnia Florestry	Arara	José	ja@example.com	42
5	—	Duck	Dickie	dd@example.com	—
6	Newt Fellowship	Axolotl	Lizzie	la@example.com	—
7	Avian Emporium	Canary	Fred	fc@example.com	19
8	Newt Fellowship	—	Molgina	m@example.com	—
9		Mander	Sally	—	—
10	Élite Emporium	Fant	Eli	ef@example.com	101

Example 97 has a slightly different version of the custom command that also checks for empty and will produce *Missing* instead of a dash:

```
\newcommand{\checkmissing}[1]{%
\DTLifnulloreempty{#1}{\emph{Missing}}{#1}}
```

3. Databases (datatool package)

The database is iterated over using `\DTLmapdata`. The forename, surname and organisation can be shown for each customer:

```
\DTLmapdata{
  \DTLmapgetvalues
  {\Surname=Surname, \Organisation=Organisation}
  Forename: \DTLmapget{key=Forename}.
  Surname: \checkmissing{\Surname}.
  Organisation: \checkmissing{\Organisation}.
  \par
}
```

Note that it won't work if you use `\DTLmapget` in the argument of `\DTLifnull` or `\DTLifnullorepty` as the command `\DTLmapget{<options>}` doesn't represent null. Either use `\DTLmapgetvalues` (as above) or get the value with `return`. The actual code used in Example 97 has a second custom command that both fetches and tests the value:

```
\newcommand{\showvalue}[1]{%
  \DTLmapget{key=#1, return=\myReturnVal}% fetch value
  \checkmissing{\myReturnVal}%
}
\DTLmapdata{
  Forename: \showvalue{Forename}.
  Surname: \showvalue{Surname}.
  Organisation: \showvalue{Organisation}.
  \par
}
```

3.10.2. Advanced Commands

The following commands are actually provided by `datatool-base` rather than `datatool` as they are used in datum tests or by the `person` package (which can be loaded without `datatool`).

The actual null command is:

```
\dtlnovalue
```

This command should not be redefined and is provided for testing purposes. It's not intended to be typeset in the document.

In general, it shouldn't be necessary to use this command, as you can test with `\DTLifnull`, but you can do a simple test against `\dtlnovalue`. For example, with `\ifdefequal`

Example 97: Iterating Through Data with Empty or Missing Values

Polly. Surname: Parrot. Organisation: *Missing*.
 Mabel. Surname: Canary. Organisation: University of Somewhere.
 Zoë. Surname: Zebra. Organisation: University of Somewhere.
 José. Surname: Arara. Organisation: Zinnia Florestry.
 Dickie. Surname: Duck. Organisation: *Missing*.
 Lizzie. Surname: Axolotl. Organisation: Newt Fellowship.
 Fred. Surname: Canary. Organisation: Avian Emporium.
 Molgina. Surname: *Missing*. Organisation: Newt Fellowship.
 Sally. Surname: Mander. Organisation: *Missing*.
 Eli. Surname: Fant. Organisation: Élite Emporium.

(provided by etoolbox). The difference is that `\DTLifnull` will also test for the string null and number null commands, below.

When fetching values from a database, some commands (such as `\DTLdisplaydb`) will set the placeholder that stores the value of the current row and column to a “string null” or “numeric null” according to the column type. So the `\DTLifnull` and `\DTLifnulloreempty` tests for null will also compare their first argument against the following.

`\DTLstringnull`

Represents null for values that are expected to be strings. This command should not be redefined but may be tested against to determine if a placeholder command represents a null string.

`\DTLnumbernull`

Represents null for values that are expected to be numeric. This will fully expand to 0 if used in calculations. This command should not be redefined but may be tested against to determine if a placeholder command represents a null number.

`\datatool_if_null:NTEF <tl var> {<true>} {<false>}`
`\datatool_if_null_p:N <tl var>`

Tests if the given token list variable represents null. This will test `<tl var>` for equality with `\dtlnovalue`, `\DTLstringnull` and `\DTLnumbernull`.

`\datatool_if_null_or_empty:NTEF <tl var> {<true>} {<false>}`
`\datatool_if_null_or_empty_p:N <tl var>`

3. Databases (datatool package)

Tests if the given token list variable is empty or represents null. This will test `<tl var>` for equality with `\dtlnovalue`, `\DTLstringnull` and `\DTLnumbernull` and the empty datum constant `\c_datatool_empty_datum_tl` as well as testing if the token list variable has an empty value.

`\datatool_if_null:nTF {<tl>} {<true>} {<false>}`

If the argument is a single token, this will return the same result as `\datatool_if_null:nTF` otherwise the result will be false. The higher level user command `\DTLifnull` now simply uses `\datatool_if_null:nTF`.

`\datatool_if_null_or_empty:nTF {<tl>} {<true>} {<false>}`

If the argument is a single token, this will return the same result as `\datatool_if_null_or_empty:nTF` otherwise it will be true if the argument is empty and false otherwise. The higher level user command `\DTLifnulloreempty` now simply uses `\datatool_if_null_or_empty:nTF`.

If `<tl>` isn't a single token, `\datatool_if_null_or_empty:nTF` won't test for the string or numeric null values represented by `\DTLstringnull` and `\DTLnumbernull`.

3.11. Special Values

A special value is an entry that's added to a database where the value starts with:

`\dtlspecialvalue{<text>}`

This simply expands to its argument but its presence at the start of a value indicates that special treatment is required for that entry when adding the entry to the database and when writing the database to a DBTEX file. When adding a value that starts with this command to a database, the `new-value-trim`, `new-value-expand` and `store-datum` options will be ignored.

If you use an action to add the value, such as the `new entry` action, the `expand-value` and `expand-once-value` will remove the special command and it won't be considered a special value.

When writing the database to an external DBTEX file with `\DTLwrite`, any entry starting with `\dtlspecialvalue` will be allowed to expand, regardless of the I/O `expand` setting. Note that if there is any trailing content after the argument of `\dtlspecialvalue` that will also expand. This doesn't apply to any of the other file formats. For example, if you export to DTLTEX or CSV then `\dtlspecialvalue` will be included with `expand=none`.

This command was added for the benefit of the `datagidx` package to encapsulate values in internal columns such as `Used` to allow them to be reset for use at the start of the next `LATEX` run when the database saved at the end of the previous run is read in the preamble of the next.



You can't perform a quick lookup by value (for example, with the `select row` action) where the entry has the `\dtlspecialvalue` command.

3.12. Editing Database Rows

Rows can be edited whilst looping through a database with `\DTLmapdata` with the `allow - edits` option set (see §3.8.1.2 and Example 87) or with the unstarred `\DTLforeach` (see §3.8.2.1 and Example 92).

It's also possible to edit a single row of the database outside of those loop commands. In order to do this, it's first necessary to select the required row and set it as the current row. This stores the row information in the `\dtlcurrentrow` token register, the preceding rows in the `\dtlbeforerow` token register, and the following rows in the `\dtlafterrow` token register.

Modifications can then be performed on the `\dtlcurrentrow` register using the commands described in §3.16.1. Once all the required changes have been made, the database contents can then be updated using `\dtlrecombine` or, if the row should be deleted, `\dtlrecombineomitcurrent`.



If you need to modify a number of rows, it's simpler to use `\DTLmapdata` with `allow - edits`.

The current row can be setup using commands such as `\dtlgetrow` or you can use the `select row` action (or the `find` action with the option `select=true`).

Note that although iterative commands such as `\DTLdisplaydb` and `\DTLforeach` set the current row, if you use any of the editing commands described in §3.16.1 within the loop body, they will cause interference. Within `\DTLforeach`, use the designated commands described in §3.8.2.1.

Within `\DTLdisplaydb`, the current row is only set to allow querying values from the hooks while the tabular contents are being constructed (for example, to fetch an entry from the current row or to compute aggregates with the `current row values` or `current row aggregate` actions).

3. Databases (datatool package)

Example 98 selects the row from the “customers” database (see §3.2.3) where the `Id` column is equal to 9. This happens to be the ninth row, so the row could be selected by its row index with `\dtlgetrow`:

98

```
\dtlgetrow{customers}{9}
```

If it’s possible that the row index may not be the same as the `Id` value, `\dtlgetrowforvalue` may be used (since the `Id` column has unique values). The column index (not the key) is needed in the reference. In this case the `Id` column has index 1:

```
\dtlgetrowforvalue{customers}{1}{9}
```

If the index isn’t known, it can be obtained with `\dtlcolumnindex`:

```
\dtlgetrowforvalue{customers}  
{\dtlcolumnindex{customers}{Id}}% column index  
{9}% value
```

Alternatively, the `select row` action may be used:

```
\DTLaction[key=Id,value=9]{select row}
```

If a more complex selection criteria is required, the `find` action can be used. Remember that the `select` option should be set if the `find` action should select the current row.

The customer with `Id` set to 9 has an empty `Organisation` column. This means that the column has actually been set in the current row, so it needs to be replaced rather than set:

```
\dtlreplaceentryincurrentrow  
{Newt Fellowship}% new value  
{\dtlcolumnindex{customers}{Organisation}}  
% column index
```

This row doesn’t have the `Age` and `Email` columns set. These can be appended, but note that in this case the column key rather than column index is supplied. This is because a new column will be created if it hasn’t already been defined for the database.

3. Databases (*datatool* package)

```
\dtlappendentrytocurrentrow  
{Email}% column key  
{s@example.com}% value
```

Alternatively, `\dtlupdateentryincurrentrow` may be used to update an entry if the column is already set or append it otherwise. This again takes the key rather than the column index as the first argument:

```
\dtlupdateentryincurrentrow  
{Age}% column key  
{23}% value
```

Once all modifications have been made, the `\dtlbeforerow`, `\dtlcurrentrow` and `\dtlafterrow` content needs to be recombined in order to finish the updates:

```
\dtlrecombine
```

Suppose now, the customer with Id 2 needs to be remove. The required row again needs to be selected first:

```
\dtlgetrowforvalue{customers}  
{\dtlcolumnindex{customers}{Id}}% column index  
{2}% value
```

The database now needs to be reconstructed without this row:

```
\dtlrecombineomitcurrent
```

Example 98 then redisplay the data after these modifications. (Compare with the original data shown in Example 94.)

Example 98: Editing a Row of Data

Id	Organisation	Surname	Forename	Email	Age
1		Parrot	Polly	pp@example.com	4
3	University of Somewhere	Zebra	Zoë	zz@example.com	2
4	Zinnia Florestry	Arara	José	ja@example.com	4
5		Duck	Dickie	dd@example.com	
6	Newt Fellowship	Axolotl	Lizzie	la@example.com	
7	Avian Emporium	Canary	Fred	fc@example.com	1
8	Newt Fellowship		Molgina	m@example.com	
9	Newt Fellowship	Mander	Sally	sally.mander@example.com	2
10	Élite Emporium	Fant	Eli	ef@example.com	10

3.13. Arithmetical Computations on Database Entries

The `aggregate` action (see §3.3) provides a way of aggregating numeric data in one or two columns of a database. Alternatively, you can use the commands listed here. Aside from `\DTLcomputebounds`, the aggregate commands in this section return formatted numbers. Additionally, rows may be filtered according to a condition. This is different to the `aggregate` action which returns plain numbers with the default `datum=false` action setting (see Example 69).

The commands described in §2.5 may be used on database values. Remember that if your database contains formatted numbers rather than plain numbers, you will need to use the commands in §2.5.2 to parse the formatted number to obtain the numeric value.

If you need to repeatedly parse numeric values in a database, you can increase efficiency by setting `store-datum=true` before loading or creating a database.

The commands described in §2.5 are dependent on the `math` processor. As from version 3.0, the aggregate commands described here directly use the `l3fp` library after converting formatted numbers to reduce the parsing overhead.

```
\DTLsumforkeys [⟨condition⟩] [⟨assign-list⟩] {⟨db list⟩} {⟨key list⟩} {⟨cmd⟩}
```

Sums all numeric values in the columns identified by the keys in the comma-separated `⟨key list⟩` for all databases listed in the comma-separated `⟨db list⟩` and assigns `⟨cmd⟩` to the total as a formatted number. If `store-datum=true`, `⟨cmd⟩` will be a datum control sequence, otherwise `⟨cmd⟩` will simply expand to the formatted number.

3. Databases (datatool package)

The first optional argument *<condition>* may be set to a boolean that's suitable for use in the first argument of `\ifthenelse`. The second optional argument *<assign-list>* may be set to the *<cs>= <key>* assignment list suitable for use in `\DTLmapgetvalues` so that the placeholder commands *<cs>* may be referenced in *<condition>*. If the condition evaluates to false, the row will be omitted from the total. Note that any non-numeric values will automatically be skipped.

A quicker alternative in the case of only one database, one column and no condition is:

```
\DTLsumcolumn{<db>}{<key>}{<cmd>}
```

This sums over all numeric items in the column with the label *<key>* of the database identified by *<db>* and stores the result as a formatted number in the control sequence *<cmd>*. If `store-datum=true`, *<cmd>* will be a datum control sequence, otherwise *<cmd>* will simply expand to the formatted number.

```
\DTLmeanforkeys[<condition>][<assign-list>]{<db list>}{<key list>}{<cmd>}
```

Computes the mean (average) of all numeric values in the columns identified by the keys in the comma-separated *<key list>* for all databases listed in the comma-separated *<db list>* and assigns *<cmd>* to the result as a formatted number. If `store-datum=true`, *<cmd>* will be a datum control sequence, otherwise *<cmd>* will simply expand to the formatted number. The optional arguments are as for `\DTLsumforkeys`.

A quicker alternative in the case of only one database, one column and no condition is:

```
\DTLmeanforcolumn{<db>}{<key>}{<cmd>}
```

This computes the mean over all numeric items in the column with the label *<key>* of the database identified by *<db>* and stores the result as a formatted number in the control sequence *<cmd>*. If `store-datum=true`, *<cmd>* will be a datum control sequence, otherwise *<cmd>* will simply expand to the formatted number.

```
\DTLvarianceforkeys[<condition>][<assign-list>]{<db list>}{<key list>}{<cmd>}
```

Computes the variance of all numeric values in the columns identified by the keys in the comma-separated *<key list>* for all databases listed in the comma-separated *<db list>* and assigns *<cmd>* to the result as a formatted number. If `store-datum=true`, *<cmd>* will be a datum control sequence, otherwise *<cmd>* will simply expand to the formatted number. The optional arguments are as for `\DTLsumforkeys`.

A quicker alternative in the case of only one database, one column and no condition is:

```
\DTLvarianceforcolumn{<db>}{<key>}{<cmd>}
```

3. Databases (datatool package)

This computes the variance of all numeric items in the column with the label $\langle key \rangle$ of the database identified by $\langle db \rangle$ and stores the result as a formatted number in the control sequence $\langle cmd \rangle$. If `store-datum=true`, $\langle cmd \rangle$ will be a datum control sequence, otherwise $\langle cmd \rangle$ will simply expand to the formatted number.

`\DTLsdforkeys [ $\langle condition \rangle$ ] [ $\langle assign-list \rangle$ ] { $\langle db list \rangle$ } { $\langle key list \rangle$ } { $\langle cmd \rangle$ }`

Computes the standard deviation of all numeric values in the columns identified by the keys in the comma-separated $\langle key list \rangle$ for all databases listed in the comma-separated $\langle db list \rangle$ and assigns $\langle cmd \rangle$ to the result as a formatted number. If `store-datum=true`, $\langle cmd \rangle$ will be a datum control sequence, otherwise $\langle cmd \rangle$ will simply expand to the formatted number. The optional arguments are as for `\DTLsumforkeys`.

A quicker alternative in the case of only one database, one column and no condition is:

`\DTLsdforcolumn { $\langle db \rangle$ } { $\langle key \rangle$ } { $\langle cmd \rangle$ }`

This computes the standard deviation of all numeric items in the column with the label $\langle key \rangle$ of the database identified by $\langle db \rangle$ and stores the result as a formatted number in the control sequence $\langle cmd \rangle$. If `store-datum=true`, $\langle cmd \rangle$ will be a datum control sequence, otherwise $\langle cmd \rangle$ will simply expand to the formatted number.

`\DTLminforkeys [ $\langle condition \rangle$ ] [ $\langle assign-list \rangle$ ] { $\langle db list \rangle$ } { $\langle key list \rangle$ } { $\langle cmd \rangle$ }`

Determines the minimum value over all numeric values in the columns identified by the keys in the comma-separated $\langle key list \rangle$ for all databases listed in the comma-separated $\langle db list \rangle$ and assigns $\langle cmd \rangle$ to the result as a formatted number. If `store-datum=true`, $\langle cmd \rangle$ will be a datum control sequence, otherwise $\langle cmd \rangle$ will simply expand to the formatted number. The optional arguments are as for `\DTLsumforkeys`.

A quicker alternative in the case of only one database, one column and no condition is:

`\DTLminforcolumn { $\langle db \rangle$ } { $\langle key \rangle$ } { $\langle cmd \rangle$ }`

This determines the minimum value over all numeric items in the column with the label $\langle key \rangle$ of the database identified by $\langle db \rangle$ and stores the result as a formatted number in the control sequence $\langle cmd \rangle$. If `store-datum=true`, $\langle cmd \rangle$ will be a datum control sequence, otherwise $\langle cmd \rangle$ will simply expand to the formatted number.

`\DTLmaxforkeys [ $\langle condition \rangle$ ] [ $\langle assign-list \rangle$ ] { $\langle db list \rangle$ } { $\langle key list \rangle$ } { $\langle cmd \rangle$ }`

Determines the maximum value over all numeric values in the columns identified by the keys in the comma-separated $\langle key list \rangle$ for all databases listed in the comma-separated $\langle db list \rangle$ and assigns $\langle cmd \rangle$ to the result as a formatted number. If `store-datum=true`, $\langle cmd \rangle$ will be

3. Databases (datatool package)

a datum control sequence, otherwise $\langle cmd \rangle$ will simply expand to the formatted number. The optional arguments are as for `\DTLsumforkeys`.

A quicker alternative in the case of only one database, one column and no condition is:

`\DTLmaxforcolumn{ $\langle db \rangle$ }{ $\langle key \rangle$ }{ $\langle cmd \rangle$ }`

This determines the maximum value over all numeric items in the column with the label $\langle key \rangle$ of the database identified by $\langle db \rangle$ and stores the result as a formatted number in the control sequence $\langle cmd \rangle$. If `store-datum=true`, $\langle cmd \rangle$ will be a datum control sequence, otherwise $\langle cmd \rangle$ will simply expand to the formatted number.

If you need the sum, mean, standard deviation, minimum and maximum values for a column of just one database, it's more efficient to use the `aggregate` action. Note, however, that specifying two columns in the `aggregate` action indicates two separate sets of data, whereas two columns with commands like `\DTLsumforkeys` treats both columns as a single block of data.

`\DTLcomputebounds[ $\langle condition \rangle$ ]{ $\langle db-list \rangle$ }{ $\langle x-key \rangle$ }{ $\langle y-key \rangle$ }{ $\langle minX cmd \rangle$ }{ $\langle minY cmd \rangle$ }{ $\langle maxX cmd \rangle$ }{ $\langle maxY cmd \rangle$ }`

Computes the maximum and minimum x and y values over all the databases listed in the comma-separated $\langle db-list \rangle$, where the x values are in the column identified by the label $\langle x-key \rangle$ and the y values are in the column identified by the label $\langle y-key \rangle$.

The optional $\langle condition \rangle$ may be used to filter rows. If provided $\langle condition \rangle$ should be in the form suitable for use in the first argument of `\ifthenelse`. Only values in rows where the conditional evaluates to true will be referenced.

The results are stored as plain numbers in the control sequences $\langle minX cmd \rangle$ (the minimum x value), $\langle minY cmd \rangle$ (the minimum y value), $\langle maxX cmd \rangle$ (the maximum x value), and $\langle maxY cmd \rangle$ (the maximum y value).

If you only have one database and no condition, you may prefer to use the `aggregate` action:

```
\DTLaction
[ $\langle$ 
  key= $\langle x-key \rangle$ ,
  key2= $\langle y-key \rangle$ ,
  options={min,max}
 $\rangle$ ]{aggregate}
\DTLget[min]{ $\langle minX cmd \rangle$ }
\DTLget[min2]{ $\langle minY cmd \rangle$ }
\DTLget[max]{ $\langle maxX cmd \rangle$ }
\DTLget[max2]{ $\langle maxY cmd \rangle$ }
```


3.14. Sorting a Database

If you have a large database, it's much more efficient to sort using `datatooltk` or some other external tool which can save to a file format that `datatool` can load.

If your database isn't too large and you are unable to include an external tool into your document to perform sorting, then it's possible to sort a database with the commands described in this section.

Version 3.0 has introduced a new command `\DTLsortdata` (described in §3.14.1), which is analogous to `\DTLsortwordlist`. It's more efficient and more flexible than the older `\dtlsort` (described in §3.14.2).

3.14.1. Sorting with `\DTLsortdata`

```
\DTLsortdata[<options>]{<db-name>}{<criteria>}
```

This command sorts the data (in the database identified by *<db-name>*) according to the given criteria. It uses the same underlying methodology as `\DTLsortwordlist`, in that it first converts all the sort values into byte sequences using a handler function, such as `\DTLsortwordhandler`, and then sorts the byte sequences. This makes sorting faster as it doesn't have to repeatedly parse each sort value.

The modification to the database will match the `global` setting.

The *<db-name>* argument may be empty, which indicates that the default database (identified with `default-name` in `\DTLsetup`) should be sorted. Available *<options>* are described in §3.14.1.1 and the *<criteria>* argument is described in §3.14.1.2.

3.14.1.1. `\DTLsortdata` Options

The optional argument *<options>* of `\DTLsortdata` should be a *<key>=<value>* list of any of the following:

```
function=<function>                                initial: \DTLsortwordhandler
```

The handler function to use (see §2.9.5.2).



```
encap=<cs>
```

The value should be a command that takes three arguments: $\{ \langle value \rangle \} \{ \langle col-idx \rangle \} \{ \langle db-name \rangle \}$. If set, all non-null values will be encapsulated with this command and expanded before being passed to the handler function. The first argument $\langle value \rangle$ is the actual value, the second $\langle col-idx \rangle$ is the column index from which the value was obtained, and $\langle db-name \rangle$ is the database name.



The value will expand differently with `encap` set. Be careful of fragile commands in the database if this option is used.

An empty setting `encap={ }` indicates no encapsulation. Example 183 in §7.10.1 uses this option to encapsulate values with `\DTLbibsorthencap`.



```
replace=<value>
```

initial: null or empty

This determines whether or not a missing value should be replaced (if the `replacements` column option is set). The value may be either `null`, which will only replace null values, or `null or empty`, which will replace null or empty values.



```
missing-column-action=<value>
```

initial: error

This option indicates what to do if a column key referenced in the sort criteria doesn't exist. The value may be one of: `error` (trigger an error), `warn` (issue a warning) or `ignore` (ignore the reference).



```
save-group-key=<col-key>
```

initial: empty

If this option is set to a non-empty value, `\DTLsortdata` will obtain the letter group (using `\DTLassignlettergroup`) from the sort value and save it in the column identified by $\langle col-key \rangle$. If the column doesn't exist, it will be created. After obtaining the letter group, the value will be post-processed by:



```
\datatool_post_process_lettergroup:N <tl var>
```

This requires $\LaTeX 3$ syntax and does nothing by default. The argument is the token list variable used to store the letter group.



```
save-group-column=<col-idx>
```

initial: 0

As `save-group-key` but identifies the column by its index. Note that in this case, the

3. Databases (datatool package)

column must either exist or be one more than the database's column count.

`save-group`

A shortcut for `save-group-key=group`.

`save-sort-key=<col-key>`

initial: empty

If this option is set to a non-empty value, `\DTLsortdata` will save the sort value in the column identified by `<col-key>`. If the column doesn't exist, it will be created. This is primarily intended for debugging. If the resulting order is unexpected, this can be used to check that the sort values were correctly set. Alternatively, you can use the `verbose` package option to view the information in the transcript.

`save-sort-column=<col-idx>`

initial: 0

As `save-sort-key` but identifies the column by its index. Note that in this case, the column must either exist or be one more than the database's column count.

`save-sort`

A shortcut for `save-sort-key=sort`.

3.14.1.2. `\DTLsortdata` Column Options

The `<criteria>` argument of `\DTLsortdata` should be a comma-separated list where each item is in the form `<column-key>={ <column-options> }` where `<column-key>` is the label identifying a column to sort and `<column-options>` is a `<key>=<value>` list of options that apply to that column.

The `= { <column-options> }` part may be omitted if the default options should be used. If present, the following options are available:

`ascending=<boolean>`

default: true; initial: true

If this boolean option is true, the sort will be in ascending order. For columns with a numerical data type, this will be in ascending numerical order, otherwise it will be in ascending lexicographic order.

`asc`

A valueless shortcut for `ascending=true`.

`descending=<boolean>`

default: true; initial: false

An antonym of `ascending`. If true, the sort will be in descending order.



```
desc
```

A valueless shortcut for `descending=true`.



```
replacements=<list>
```

initial: *empty*

If set, the value should be a comma-separated list of column keys to use as a replacement if a missing value (as determined by `replace`) is encountered.

The first *<column-key>* in the criteria list is the primary sort column. If there are any identical values in that column, then the sort values from the second *<column-key>* will be used, and so on. For example, the “marks” database (§3.2.1) has three students with the surname “Brown” and two of them have the surname “Jane”. The student number disambiguates them. So the following will first sort by surname, then (for identical surnames) by forename, and finally (for identical surname and forename) by student number:



```
\DTLsortdata{marks}{Surname,Forename,StudentNo}
```

If a column has missing (null) values, then those values will be treated as empty for string columns or 0 for numeric columns. This means that sorting a string column in ascending order will place all the null values at the top with the empty values. The secondary sort columns in the criteria list will then determine their relative order.

If you want to specify an alternative column to use if a value is missing, then you need to identify the replacement column with the `replacements` option in *<column-options>*. Whether or not an empty value (as opposed to a null value) is considered missing is determined by the `replace` option, which may be supplied in the optional argument of `\DTLsortdata`.

3.14.1.3. \DTLsortdata Examples

Example 99 loads the “customers” database from the `customers.csv` file (see §3.2.3), which has some empty values in the Organisation column. This data is then sorted with:

 99



```
\DTLsortdata{customers}
{Organisation,Surname,Forename}
```

In this case, no replacement columns are provided, so the sort value for the Organisation column in the Polly Parrot, Dickie Duck and Sally Mander rows will be empty and those rows will be placed at the start. Their relative order is then determined by their Surname, so the Dickie Duck row comes first. (Compare Example 99 with the near identical Example 102 which has localisation support.)

The null values are shown as “NULL” (for string columns) or 0 (for numeric columns) by `\DTLdisplaydb`. Whereas empty values show as empty. This is why Dickie Duck’s age is

3. Databases (datatool package)

blank in the Age column, because it was set to empty, but the missing ages (where there was no trailing comma at the end of the line in the CSV file) show as 0.

Example 99: Sorting CSV Data Using \DTLsortdata by Organisation, Surname and Forename With No Replacements

Id	Organisation	Surname	Forename	Email	Age
5		Duck	Dickie	dd@example.com	
9		Mander	Sally	NULL	0
1		Parrot	Polly	pp@example.com	42
7	Avian Emporium	Canary	Fred	fc@example.com	19
8	Newt Fellowship		Molgina	m@example.com	0
6	Newt Fellowship	Axolotl	Lizzie	la@example.com	0
2	University of Somewhere	Canary	Mabel	mc@example.com	0
3	University of Somewhere	Zebra	Zoë	zz@example.com	21
4	Zinnia Florestry	Arara	José	ja@example.com	42
10	Elite Emporium	Fant	Eli	ef@example.com	101

Example 100, in contrast, has:

```
\DTLsortdata{customers}
{
  Organisation={replacements={Surname,Forename}},
  Surname={replacements={Forename}},
  Forename
}
```

In this case, if the sort value is missing from the designated column, the first column within the corresponding `replacements` list that doesn't have a missing value will be used.

Primary In this example, the primary sort value is obtained from the Organisation column. If that value is missing, the primary sort value will be obtained from the Surname column, but if that is also missing then the primary sort value will be obtained from the Forename column. If that is also missing, then the primary sort value will be empty.

Secondary The secondary sort value is only used if the primary sort values are identical when comparing two rows. In this case, the secondary sort value is obtained from the Surname column. If that value is missing, the secondary sort value will be obtained from the Forename column. If that value is also missing, the secondary sort value will be empty.

Tertiary The tertiary sort value is only used if both the primary and secondary sort values are identical when comparing two rows. In this case, the tertiary sort value is obtained from the Forename column. However, no replacement columns have been identified, so if the Forename column is empty, the tertiary sort value will be empty.

Example 100: Sorting CSV Data Using `\DTLsortdata` by Organisation, Surname and Forename With Replacements

Id	Organisation	Surname	Forename	Email	Age
7	Avian Emporium	Canary	Fred	fc@example.com	19
5		Duck	Dickie	dd@example.com	
9		Mander	Sally	NULL	0
6	Newt Fellowship	Axolotl	Lizzie	la@example.com	0
8	Newt Fellowship		Molgina	m@example.com	0
1		Parrot	Polly	pp@example.com	42
2	University of Somewhere	Canary	Mabel	mc@example.com	0
3	University of Somewhere	Zebra	Zoë	zz@example.com	21
4	Zinnia Florestry	Arara	José	ja@example.com	42
10	Élite Emporium	Fant	Eli	ef@example.com	101

Example 101 defines the customer data in the document using action commands instead of loading the data from a CSV file (see §3.2.3). This means that some of the rows have a missing Organisation column, which can't occur with the CSV file (except where missing columns occur at the end). The default `replace={null or empty}` setting will treat empty values as missing, so in Example 101

101

```
\DTLsortdata[replace=null or empty]{customers}
{
  Organisation={replacements={Surname,Forename}},
  Surname={replacements={Forename}},
  Forename
}
```

the resulting order is the same as for Example 100. However, changing the setting so that only null (not empty) values are treated as missing results in a different order.

```
\DTLsortdata[replace=null]{customers}
{
  Organisation={replacements={Surname,Forename}},
  Surname={replacements={Forename}},
  Forename
}
```

↑ Example 101: Sorting Data Using \DTLsortdata With Replacements

(Null vs Empty)

Sort replacing null or empty values.

Id	Organisation	Surname	Forename	Email	Age
7	Avian Emporium	Canary	Fred	fc@example.com	19
5	NULL	Duck	Dickie	dd@example.com	0
9		Mander	Sally	NULL	0
6	Newt Fellowship	Axolotl	Lizzie	la@example.com	0
8	Newt Fellowship	NULL	Molgina	m@example.com	0
1	NULL	Parrot	Polly	pp@example.com	42
2	University of Somewhere	Canary	Mabel	mc@example.com	0
3	University of Somewhere	Zebra	Zoë	zz@example.com	21
4	Zinnia Florestry	Arara	José	ja@example.com	42
10	Élite Emporium	Fant	Eli	ef@example.com	101

Sort replacing null (not empty) values.

Id	Organisation	Surname	Forename	Email	Age
9		Mander	Sally	NULL	0
7	Avian Emporium	Canary	Fred	fc@example.com	19
5	NULL	Duck	Dickie	dd@example.com	0
6	Newt Fellowship	Axolotl	Lizzie	la@example.com	0
8	Newt Fellowship	NULL	Molgina	m@example.com	0
1	NULL	Parrot	Polly	pp@example.com	42
2	University of Somewhere	Canary	Mabel	mc@example.com	0
3	University of Somewhere	Zebra	Zoë	zz@example.com	21
4	Zinnia Florestry	Arara	José	ja@example.com	42
10	Élite Emporium	Fant	Eli	ef@example.com	101

In all the above examples, “É” comes after “Z” because the examples haven’t used any localisation support.

Example 102 adapts Example 99 to use the GB English localisation support. This requires `datatool-english` to also be installed. The only difference in the document code between the two examples is the locale identification:

102

```
\usepackage[locales=en-GB]{datatool}
```

3. Databases (datatool package)



↑ Example 102: Sorting CSV Data Using \DTLsortdata With Language Support

Id	Organisation	Surname	Forename	Email	Age
5		Duck	Dickie	dd@example.com	
9		Mander	Sally	NULL	0
1		Parrot	Polly	pp@example.com	42
7	Avian Emporium	Canary	Fred	fc@example.com	19
10	Élite Emporium	Fant	Eli	ef@example.com	101
8	Newt Fellowship		Molgina	m@example.com	0
6	Newt Fellowship	Axolotl	Lizzie	la@example.com	0
2	University of Somewhere	Canary	Mabel	mc@example.com	0
3	University of Somewhere	Zebra	Zoë	zz@example.com	21
4	Zinnia Florestry	Arara	José	ja@example.com	42

Examples 103 & 104 both sort the data first by the Age column (which is numerical) and then by the Surname column:



```
\DTLsortdata{customers}{Age, Surname}
```

The difference between the two is that Example 103 uses data from the CSV file, which has an empty age element as well as missing age elements, whereas Example 104 uses the data created within the document via action commands, which has missing but not empty age elements. (There is an empty Organisation element, but that column isn't contributing to the sort.)

103

104



↑ Example 103: Sorting Data Using \DTLsortdata on Age then Surname (Empty or Null Values)

Id	Organisation	Surname	Forename	Email	Age
8	Newt Fellowship		Molgina	m@example.com	0
6	Newt Fellowship	Axolotl	Lizzie	la@example.com	0
2	University of Somewhere	Canary	Mabel	mc@example.com	0
5		Duck	Dickie	dd@example.com	
9		Mander	Sally	NULL	0
7	Avian Emporium	Canary	Fred	fc@example.com	19
3	University of Somewhere	Zebra	Zoë	zz@example.com	21
4	Zinnia Florestry	Arara	José	ja@example.com	42
1		Parrot	Polly	pp@example.com	42
10	Élite Emporium	Fant	Eli	ef@example.com	101

3. Databases (datatool package)

Example 104: Sorting Data Using \DTLsortdata on Age then Surname (No Empty Sort Values)

Id	Organisation	Surname	Forename	Email	Age
8	Newt Fellowship	NULL	Molgina	m@example.com	0
6	Newt Fellowship	Axolotl	Lizzie	la@example.com	0
2	University of Somewhere	Canary	Mabel	mc@example.com	0
5	NULL	Duck	Dickie	dd@example.com	0
9		Mander	Sally	NULL	0
7	Avian Emporium	Canary	Fred	fc@example.com	19
3	University of Somewhere	Zebra	Zoë	zz@example.com	21
4	Zinnia Florestry	Arara	José	ja@example.com	42
1	NULL	Parrot	Polly	pp@example.com	42
10	Élite Emporium	Fant	Eli	ef@example.com	101

Example 105 sorts the “marks” database (see §3.2.1) in descending order of marks for the first assignment column (`Assign1`). The secondary sort (where the marks are identical) is by the Surname column in ascending order:

105

```
\DTLsortdata{marks}
{
  Assign1={descending=true},
  Surname={ascending=true}
}
```

This means that the students who obtained the same mark for assignment 1 are listed in alphabetical order relative to each other.

Since `ascending=true` is the default, that may be omitted for the Surname column, and `descending=true` may have the value omitted if it's true, so the `Assign1` column criteria can be written as `Assign1={descending}` and since the part after the equals (=) doesn't contain any commas or equals the outer bracing may be omitted. So the above can be written more succinctly as:

```
\DTLsortdata{marks}{Assign1=descending, Surname}
```

Since `desc` is a synonym of `descending=true`, this can also be written as:

```
\DTLsortdata{marks}{Assign1=desc, Surname}
```

Example 105: Sorting Data Using \DTLsortdata by Descending

Numeric and Ascending String Values

Surname	Forename	StudentNo	Assign1	Assign2	Assign3
Brown	Jane	102647	75	84	80
Brady	Roger	106872	68	60	62
Smith, Jr	John	102689	68	57	72
Brown	Jane	102646	64	92	79
Adams	Zoë	105987	52	48	57
Verdon	Clare	104356	45	50	48
Brown	Andy	103569	42	52	54

3.14.2. Sorting with \dtlsort

The older `\dtlsort` command is less efficient than the newer `\DTLsortdata`, although `\dtlsort` has been rewritten in version 3.0 to use L^AT_EX3's sequence sorting.

```
\dtlsort [⟨replacements⟩] {⟨criteria⟩} {⟨db-name⟩} {⟨handler-cs⟩}
```

Sorts the database identified by `⟨db-name⟩` using the given `⟨handler-cs⟩` function for the comparisons, which should be of the type described in §2.9.5.1. Unlike `\DTLsortdata`, the `⟨db-name⟩` argument can't be empty.

The modification to the database will match the `global` setting.

The `⟨criteria⟩` argument should be a comma-separated list of column keys, where each item in the list may be just the key or in the form `⟨col-key⟩=⟨order⟩`. The `⟨order⟩` may be ascending or descending. If omitted, ascending is assumed.

Unlike `\DTLsortdata`, where you may specify more information with `⟨col-key⟩={⟨options⟩}`, with `\dtlsort`, only the keywords “ascending” or “descending” may be used.

The other difference between `\dtlsort` and `\DTLsortdata` is that with `\DTLsortdata`, the list of replacements is set for specific columns, whereas with `\dtlsort`, a single list of replacement columns may be provided in the optional argument, which will be used if any of the columns listed in `⟨criteria⟩` have a missing value. Also with `\dtlsort`, the replacements are only used for null values (see §3.10) not for empty values.

If any listed column in either `⟨criteria⟩` or `⟨replacements⟩` is undefined, a warning will occur and the column will be dropped from the list. (Unlike `\DTLsortdata`, there's no option

3. Databases (datatool package)

to suppress the warning.) The list of replacements may be omitted or empty but the $\langle criteria \rangle$ argument must have at least one defined column.

If any column has been identified as a numeric column, a numerical comparison will be used (with `\DTLnumcompare`). In the event that two numeric values are deemed equivalent, their string value will be compared using the provided handler.

For example,

```
\dtlsort[Editor,Title]{Author}{books}
{\dtlwordindexcompare}
```

This will sort the “books” database on the “Author” column (in ascending order). Any row that doesn’t have the Author column set will have the sort value obtained from the “Editor” column instead. If that column also isn’t set, the sort value will be obtained from the “Title” column.

Compare this with:

```
\dtlsort[Title]{Author, Series, Volume}{books}
{\dtlwordindexcompare}
```

In this case, the database will be sorted by the “Author” column first. If that column isn’t set for a particular row, the sort value will be obtained from the “Title” column. If the sort value (Author, if set, or Editor otherwise) is equivalent to the other row being compared (that is, the sort handler returns 0), then the two rows will be compared on the “Series” column. If that column isn’t set, the Title will be used for the comparison instead. If this secondary comparison is also considered equivalent, the two rows will then be compared on the “Volume” column.

This means that if the Author, Series and Volume columns are all missing for a particular row, then the primary, secondary and tertiary sort values will all be the value from the Title column. Contrast this with the more flexible `\DTLsortdata`:

```
\DTLsortdata{books}{
  Author={replacements={Editor, Organisation}},
  Series={replacements={Title}},
  Volume
}
```

```
\DTLsort[ $\langle replacements \rangle$ ]{ $\langle criteria \rangle$ }{ $\langle db-name \rangle$ }
```

*modifier: **

A shortcut command that uses `\dtlsort`. The starred version uses `\dtlicompare` as the handler function, and the unstarred version uses `\dtlcompare`.

Example 106 is the closest equivalent to Example 100 that uses `\dtlsort` instead of `\DTLsortdata`:

106

```
\dtlsort[Surname,Forename]% replacements
{Organisation,Surname,Forename}% sort criteria
{customers}% database
{\dtlwordindexcompare}% handler
```

Note that this has produced a different result to Example 100 because with `\dtlsort` the replacements are only used for null values not for empty values. Remember that in both examples, localisation support will need to be added to correctly order values that contain non-ASCII characters.

Example 106: Sorting CSV Data Using `\dtlsort` by Organisation, Surname and Forename With Replacements

Id	Organisation	Surname	Forename	Email	Age
5		Duck	Dickie	dd@example.com	
9		Mander	Sally	NULL	0
1		Parrot	Polly	pp@example.com	42
7	Avian Emporium	Canary	Fred	fc@example.com	19
8	Newt Fellowship		Molgina	m@example.com	0
6	Newt Fellowship	Axolotl	Lizzie	la@example.com	0
2	University of Somewhere	Canary	Mabel	mc@example.com	0
3	University of Somewhere	Zebra	Zoë	zz@example.com	21
4	Zinnia Florestry	Arara	José	ja@example.com	42
10	Elite Emporium	Fant	Eli	ef@example.com	101

3.15. Database Files (I/O)

The data stored in the database can be saved to an external file with `\DTLwrite`, described in §3.15.4, or a new database can be created by reading in a file with `\DTLread`, described in §3.15.3.

3.15.1. File Formats

There are essentially two different types of file format according to how that data can be read into a $\LaTeX$ document:

- plain text which needs to be parsed and converted into a datatool database (CSV and TSV);
- $\LaTeX$ code which can simply be input into the document to recreate the database.

In the second case, the file extension would ordinarily be `tex` but there is a danger with `\DTLwrite` of accidentally overwriting a required document file, so the default file extension

3. Databases (*datatool* package)

is either `dtltex` (for files containing user commands, such as `\DTLnewdb`) or `dbtex` (for files containing special internal commands that aren't designed to be edited by hand).

The `dtltex` files are easier to read and edit, but are slower to load. The `dbtex` format has two versions: 2.0, which is very hard to read but is the fastest to load, and 3.0, which is easier to read and still much faster than the `dtltex` file format. Note that `datatooltk` version 1.9 can read `dbtex` v2.0 but not v3.0 (which was only introduced to `datatool` v3.0). The pending (at the time of writing) `datatooltk` version 2.0 will be able to read the newer format. Datum markup is only preserved for `format=dbtex-3`. It will be stripped when writing to all other formats.

Note that, while it is also possible to create a file containing a set of `\DTLaction` commands that define a database, this is slower to load than the `dtltex` formats. If you want to load such a file, just use `\input` or `\InputIfFileExists` as usual. There's no provision to save a file in this form.



Be aware that if a database contains values with characters that don't have their usual category code, this information will be lost when writing the database to an external file. This is particularly a problem for the `dtltex` and `dbtex` files.

The format is specified with the `format` setting in the optional argument of `\DTLread` and `\DTLwrite` or within the `io` value in `\DTLsetup`. The default is `format=csv`.

3.15.1.1. CSV and TSV Files

The default file format for both `\DTLwrite` and `\DTLread` is CSV. The only difference between `format=csv` and `format=tsv` is the default file extension (`csv` or `tsv`), and `format=tsv` will additionally implement `\DTLsettabseparator` to set the separator to a tab character and change the category code of the tab character to 12 (other).

If the file contains `LATEX` code, for example:

```
Product,Price (\$)
Book,\$
```

the use the `csv-content=tex` option. If the file contains characters that are normally special to `LATEX` but need to be treated literally then use the `csv-content=literal` option. For example:

```
Product,Price ($)
Book,$
```

The default separator for the `csv` format is a comma (`,`), and can be changed with the `separator` option. Note that if `separator` comes after `format=tsv` in the same option list, this will change the separator but leave the default extension as `tsv`.

3. Databases (datatool package)

The default separator is a double-quote ("), and can be changed with the `delimiter` option. The delimiter will always be searched for when loading a CSV/TSV file with `\DTLread`, but you can control whether or not the delimiter is written with `\DTLwrite` with the `add-delimiter` option.

Spaces between the separator and delimiter are always trimmed. Leading and trailing spaces inside the delimiter, or when there is no delimiter present, is determined by the general `new-value-trim` option. This needs to be set in `\DTLsetup`, not in the optional argument of `\DTLread`. For example, to switch off trimming:

```
\DTLsetup{new-value-trim=false}
```

Datum markup is not preserved when writing to a CSV or TSV file nor is the database name.

3.15.1.2. DTLTEX Files

The `dtltx` files are simply `LaTeX` files that contain the user level commands required to define a database. These are easier to read and edit by hand, but are slower to load. There are two supported formats:

- 2.0** (`format=dtltx-2`) This has the database name hardcoded in the file and will always try to define the database. This format contains commands like `\DTLnewdb` but `\DTLread` locally sets the `global` option to true (as the internal workings of `\DTLread` need to be scoped). If you need a local definition, you can simply input the file, with a command such as `\input`.
- 3.0** (`format=dtltx-3`) This has the database name identified near the start of the file but it's designed to allow the `name` option to override it and the `load-action=append` option will prevent the database from being created (to allow the content to be appended to another database).

Datum markup is not preserved for either version.

The remainder of this section is for advanced users who want to understand the file format.

When written by `\DTLwrite`, both start with the line:

```
% DTLTEX <version> <encoding>
```

where `<version>` identifies the DTLTEX version number (either 2.0 or 3.0), and `<encoding>` is the document encoding (obtained by expanding `\TrackLangEncodingName`). Note that this will be incorrect if the encoding changes after `datatool` is loaded. There's no way of knowing if the encoding was changed after the database was created. In general it's best to establish the document encoding as early as possible.

3. Databases (datatool package)

The next comment line provides information about file creator and creation date. In the case of a file created by datatool, this will have the datatool version details. The creation date will be obtained by expanding `\DTMnow` (provided by the `datetime2` package), if that command has been defined, otherwise `\today` will be used.

DTLTEX version 2.0 was designed to work with `\input`, so it only contains normal user-level commands that have an argument to identify the database name (see §3.4): `\DTLnewdb`, `\DTLnewrow`, and `\DTLnewdbentry`. The column headers will then be set with `\DTLsetheader` (regardless of the `no-header` option). Finally, `\dtllastloadeddb` is defined to the database name:

```
\def\dtllastloadeddb{<db-name>}
```

(This line was introduced in datatool v2.15, so any file that matches the rest of the format but omits this line should be considered DTLTEX v1.0.) Note that although `\def` is used, `\DTLread` will change this to a global definition. Note that since these are all document-level commands, the file can simply be loaded with `\input`. This method can be used to locally define the database (provided `global=false` is set).

DTLTEX version 3.0 defines the database with the commands listed below. These are only intended for use within `\DTLread`, although the file can simply be `\input`.

```
\DTLdbProvideData
```

If the file is input within `\DTLread`, this command will locally set the `default-name` to either the `name` value, if set, or `<db-name>`, otherwise, and will create the database if required by `load-action`.

If `\DTLdbProvideData` is used outside of the context of `\DTLread`, it will set the `default-name` to `<db-name>` and define the database if it doesn't already exist.

In either case, both `\dtllastloadeddb` is defined to the database name, and the `<default-name>` is also set to the same value to allow the database to be referenced in the following commands:

```
\DTLdbNewRow
```

Equivalent to:

```
\DTLnewrow*{<default-name>}
```

```
\DTLdbNewEntry{<col key>}{<value>}
```

Equivalent to

```
\DTLnewdbentry*{⟨default-name⟩}{⟨col key⟩}{⟨value⟩}
```

Unless suppressed by `no-header`, `\DTLwrite` will also include code to set the header for each column with:

```
\DTLdbSetHeader{⟨col key⟩}{⟨header⟩}
```

This is equivalent to

```
\DTLsetheader*{⟨default-name⟩}{⟨col key⟩}{⟨header⟩}
```

Note that this is different to DTLTEX v2.0, which always has the header code.

3.15.1.3. DBTEX Files

The `dbtex` files are $\text{\LaTeX}$ files that contain internal commands required to define a database. These are harder to read and edit by hand, but are faster to load than the other file formats. There are two supported formats:

2.0 (`format=dbtex-2`) This has the database name hardcoded in the file and will always try to define the database.

Datum markup is not preserved. Any spaces at the start of an element value will be lost.

3.0 (`format=dbtex-3`) This has the database name identified near the start of the file but it's designed to allow the `name` option to override it.

Datum markup is preserved.

The `load-action=append` option is not supported for this format (for either version).

The remainder of this section is for advanced users who want to understand the file format.

As with the DTLTEX format, `\DTLwrite` will start the file with comment lines. The first identifies the format:

```
% DBTEX ⟨version⟩ ⟨encoding⟩
```

where `⟨version⟩` identifies the DTLTEX version number (either 2.0 or 3.0), and `⟨encoding⟩` is the document encoding, as for DTLTEX. The second comment line is the creation information, as for DTLTEX.

The v2.0 format then starts with a check for the existence of the database and exits the file input if it already exists:

3. Databases (*datatool* package)

```
\DTLifdbexists{<db-name>}%  
{\PackageError{datatool}{Database `<db-  
name>' already exists}{}}%  
\aftergroup\endinput}{}}%
```

The rest of the file consists of low-level internal commands that define the underlying registers and commands used to store the database information. This means that @ needs to have its category code set to “letter”. A local scope is introduced to limit the effect of \makeatletter and a message will be written to the transcript if *verbose* mode is on:

```
\bgroup\makeatletter  
\dtl@message{Reconstructing database `<db-name>'}%
```

This means that the internal commands used to store the database information must be globally defined. (Note that \global is redundant as all registers are globally defined, but this is now part of the DBTEX v2.0 file format.) The column meta data is stored in a token register, which needs to be defined and then set:

```
\expandafter\global\expandafter  
\newtoks\csname dtlkeys@<db-name>\endcsname  
\expandafter\global  
\csname dtlkeys@<db-name>\endcsname={<header markup>}
```

Similarly, the database content is stored in a token register, which needs to be defined and then set:

```
\expandafter\global\expandafter  
\newtoks\csname dtldb@<db-name>\endcsname  
\expandafter\global  
\csname dtldb@<db-name>\endcsname={<body markup>}
```

The total number of rows is stored in a count register that needs to be defined and set:

```
\expandafter\global  
\expandafter\newcount\csname dtlrows@<db-  
name>\endcsname  
\expandafter\global  
\csname dtlrows@<db-name>\endcsname=<num-rows>\relax
```

Similarly, the total number of columns is stored in a count register that needs to be defined and set:

3. Databases (datatool package)

```
\expandafter\global
\expandafter\newcount\csname dtlcols@<db-
name>\endcsname
\expandafter\global
\csname dtlcols@<db-name>\endcsname=<num-columns>\relax
```

The column key to index mapping is implemented by defining:

```
\expandafter
\gdef\csname dtl@ci@<db-name>@<key>\endcsname{<column-idx>}%
```

This is done for each column. Finally, the local scope is ended and `\dtllastloadeddb` is defined:

```
\egroup
\def\dtllastloadeddb{<db-name>}%
```

Note that `\dtllastloadeddb` was only introduced in datatool v2.15, so if the last line is omitted, the file should be considered DBTEX v1.0.

The *<header markup>* consists of sub-blocks that contain the meta data for each column in the form:

```
\db@plist@elt@w
\db@col@id@w <column-idx>%
\db@col@id@end@ %
\db@key@id@w <key>%
\db@key@id@end@ %
\db@type@id@w <type>%
\db@type@id@end@ %
\db@header@id@w <header>%
\db@header@id@end@ %
\db@col@id@w <column-idx>%
\db@col@id@end@ %
\db@plist@elt@end@ %
```

These commands are quarks and have no meaning. Unpredictable results can occur in loops if the header blocks aren't in order of the column index. It depends on whether the loop iterates over the column index or maps over the header blocks.

Note that `\DTLwrite` automatically inserts the comment character at the end of most lines, even though they are not always required. If you are writing a tool that reads DBTEX files, remember to discard the comments.

The *<body markup>* is more complicated but has a similar design.³ This consists of sub-blocks

³Thanks to Morten Høgholm for the design.

3. Databases (*datatool* package)

(*row-markup*) that contain the data for each row:

```
\db@row@elt@w %  
\db@row@id@w <row-idx>%  
\db@row@id@end@ %  
\db@row@elt@w %  
<entry markup>  
\db@row@elt@w %  
\db@row@id@w <row-idx>%  
\db@row@id@end@ %
```

Again, these commands are quarks and have no meaning, and unpredictable results can occur if the blocks aren't ordered according to the row index. The *entry markup* consists of sub-blocks (*entry column markup*) that contain the data for each entry in the given row and column:

```
\db@col@id@w <column-idx>%  
\db@col@id@end@ %  
\db@col@elt@w <value>%  
\db@col@elt@end@ %  
\db@col@id@w <column-idx>%  
\db@col@id@end@ %
```

where *column-idx* is the column index and *value* is the value for that column. Again, these commands are quarks and have no meaning, and unpredictable results can occur if the blocks aren't ordered according to the column index.

Note that with DBTEX v2.0, as mentioned earlier, `\makeatletter` is used to allow internal commands, which means that `@` will have a letter category code. This will affect any data contained in the database that includes the `@` character (for example, if the database has an email column). See Example 111. Although the quarks have no meaning, spaces following those commands are ignored. This means that if an entry starts with a space that space will be lost.

It's not possible to switch to L^AT_EX3 syntax for a new file format as the data will likely contain one or more of the characters that have their category code changed by `\ExplSyntaxOn` (most notably the space character). Therefore, the datum items markup is stripped by `\DTLwrite`, since it contains L^AT_EX3 commands. The only way to retain it is via DBTEX v3.0, which hides the L^AT_EX3 syntax within its custom reconstruction commands, which expand the content before setting the underlying token registers. This is faster than constructing the content with `\DTLnewdbentry`, but not as fast as explicitly setting the token register, as is done with DBTEX v2.0.



Although DBTEX v2.0 is the fastest format to load, you need to be aware of its limitations.

The DBTEX v3.0 format, starts with the same command as for DTLTEX v3.0:

3. Databases (datatool package)

```
\DTLdbProvideData{<db-name>}
```

This sets up the *<default-name>* to either the *name* setting, if provided, or *<db-name>*, otherwise. This allows the appropriate name to be used in the subsequent commands. Note that this command also defines `\dtllastloadeddb` to the database name, but within `\DTLread` it doesn't define the database with the DBTEX formats as the database internals are either explicitly defined (`format=dbtex-2`), or they are constructed by the following command:

```
\DTLreconstructdatabase{<num-rows>}{<num-columns>}{<header code>}{<body code>}{<key-index code>}
```

This creates a database with *<num-rows>* rows and *<num-columns>* columns, where the content of the internal token register used to store the header markup is obtained by expanding *<header code>*, and the content of the internal token register used to store the database body is obtained by expanding *<body code>*. The *<key-index code>* is the code required to reconstruct the mapping from column key to column index.

The *<header code>* should only consist of an ordered series of:

```
\dtldbheaderreconstruct{<column-idx>}{<key>}{<type>}{<header>}
```

where *<column-idx>* is the column index (starting from 1), *<key>* is the unique column key, *<type>* is the data type numeric identifier (0: string, 1: integer, 2: decimal, 3: currency), and *<header>*. Note that *<header code>* is designed to expand to the explicit *<header markup>*, describe above for DBTEX v2.0.

The *<index-key code>* is a reverse mapping from column key to column index, which consists of a series of:

```
\dtldbconstructkeyindex{<key>}{<column-idx>}
```

(This corresponds to defining the commands `\dtl@ci@<db-name>@<key>` to expand to *<column-idx>*, described above for DBTEX v2.0.)

Every column in *<header code>* must have a corresponding key to index mapping. For example:

3. Databases (datatool package)

```
\DTLreconstructdatabase
{<num-rows>}{4}%
{% Header
\dtldbheaderreconstruct{1}{Name}{0}{Name}%
\dtldbheaderreconstruct{2}{Age}{1}{Age}%
\dtldbheaderreconstruct{3}{Score}{2}{Score (\%)}%
\dtldbheaderreconstruct{4}{Award}{2}
{Award (\protect \$)}%
}% End of Header
{<body code>}
{% Key to index
\dtldbconstructkeyindex{Name}{1}%
\dtldbconstructkeyindex{Age}{2}%
\dtldbconstructkeyindex{Score}{3}%
\dtldbconstructkeyindex{Award}{4}%
}% End of key to index
```

The *<body code>* is more complicated, but is designed to expand to the explicit *<body markup>*, describe above for DBTEX v2.0.

The *<body code>* consists of an ordered set of row blocks, where each block is identified with:

```
\dtldbrowreconstruct{<row-idx>}{<row code>}
```

(This is designed to expand to the *<row-markup>* sub-block, described above.) The *<row-idx>* argument is the row index (starting from 1) and *<row code>* is the row content, which should consist of an ordered set of column blocks:

```
\dtldbcolreconstruct{<column-idx>}{<content>}
```

(This is designed to expand to the *<entry column markup>*, described above.) The *<column-idx>* argument is the column index and *<content>* is the content for the given column in the given row. This may be the actual content which will be encapsulated with:

```
\dtldbvaluereconstruct{<string>}
```

or it could be a datum item, in which case *<content>* will be in the form:

```
\dtldbdatumreconstruct{<string>}{<numeric>}{<currency>}{<type>}
```

where *<string>* is the original value (such as *\\$12,500*), *<numeric>* is the plain number numeric value (for example, *12500*) or empty if the value is a string, *<currency>* is the currency symbol (for example, *\\$*), and *<type>* is the numeric data type identifier.

3.15.2. I/O Settings

Data can be loaded from an external file using `\DTLread`, described in §3.15.3, and saved to an external file using `\DTLwrite`, described in §3.15.4. Both commands have an optional argument with the settings that govern the format. Some of the settings are only applicable to a particular format or to either reading or writing a file.

When passed to the optional argument of `\DTLread` or `\DTLwrite`, these settings only have a local effect within the read or write action. You can set up defaults with the `io` option in `\DTLsetup`. For example:

```
\DTLsetup{
  io={
    separator = {;},
    delimiter = {''},
    format = {csv}
  }
}
```

If a column key can't be obtained (either from the file or from the `keys` option) when reading a `csv` or `tsv` file, then a default key will be used with the column index prefixed with:

```
\dtldefaultkey
```

So the default key for a column with index $\langle n \rangle$ is obtained by expanding `\dtldefaultkey` $\langle n \rangle$ (for example, the default key for column 4 will be “Column4”).

```
add-delimiter= $\langle value \rangle$  initial: detect
```

This option is only applicable with `\DTLwrite`, and determines whether or not to use the delimiter when writing to a `csv` or `tsv` file. This option has no effect on other formats. The value may be one of the following.

```
add-delimiter=always
```

Always add the delimiter.

```
add-delimiter=never
```

Never add the delimiter.

`add-delimiter=`**detect**

Only add the delimiters if the separator is found in the value.

`auto-keys=`*<boolean>*

initial: **false**

The column keys will be the default key (obtained by expanding `\dtldefaultkey <n>`), regardless of whether or not the file has a header row. This option overrides any previous `keys` setting and is only applicable when reading a `csv` or `tsv` file.

`autokeys=`*<boolean>*

alias: `auto-keys`

Equivalent to `auto-keys`.

`convert-numbers=`*<boolean>*

initial: **false**

Any columns that have had their data type identified as an integer, decimal or currency, will have the value read from the CSV or TSV file converted to the current localisation settings using `\DTLdecimaltolocale` or `\DTLdecimaltocurrency`, respectively. Any other type of column will be parsed as usual (except in the case of `csv-content=no-parse`). See Example 110.

Only has an effect when `\DTLread` parses `format=csv` or `format=tsv` files. This option is ignored by the other formats. If all columns are plain numbers you may want to consider using this option with `csv-content=no-parse` and `store-datum=true`.

This setting should be used with `data-types`, and the values in the indicated columns should be plain numbers. The column data type will be updated if a value in that column is found to be incompatible with the identified type (except in the case of `csv-content=no-parse`).

`csv-blank=`*<value>*

initial: **ignore**

This option is only applicable with `\DTLread` and determines how to act if an empty line is encountered in a `csv` and `tsv` file. This option has no effect with other formats.

Note that a blank line means an empty line in the file unless it occurs within grouped content with `csv-content=tex`.



```
csv-blank=ignore
```

Blank lines are ignored.



```
csv-blank=empty-row
```

Blank lines will be treated as an empty row to be added to the database.



```
csv-blank=end
```

A blank line indicates that parsing should stop. Any following content is ignored.



```
csv-content=<value>
```

initial: **literal**

This option is only applicable with `\DTLread` and determines how the content of `csv` and `tsv` files should be interpreted. This option has no effect with other formats.

If you don't intend to use any numerical operations on the data or if the data consists only of plain numbers or text that either has no `LATEX` special characters or has valid `LATEX` syntax, then the fastest method of loading a CSV file is with `csv-content=no-parse`. The other options will parse the elements in each row to determine the data type.

For example, if the first two lines of the file consist of:

```
Name , Score (\\%), {Award
(\\$) }
```

Then with `csv-content=tex` this will be read in as a single header row. The grouping allows an element value to be split across multiple lines. The first column will have the header `Name`, the second column will have the header `Score (\\%)` and the third column will have the header `Award (\\$)`.

Whereas with `csv-content=literal` this will be read in as a header row on line 1 and a row containing a single column on line 2. The header for the first column is again `Name`, but the header for the second column will be `Score (\\textbackslash \\%)`, and the header for the third column will be `\\{Award`. The entry of the first column in the first row will be `(\\textbackslash \\$) \\}`.

Note that in both case, the column keys would need to be set with `auto-keys` or `keys` as the headers for the second and third columns are inappropriate for keys.

Suppose now that the next line is:

```
"Chlo\\"e", 89, 5.50
```

In this case, the result depends on the `csv-escape-chars` setting, which determines whether or not to strip the backslash in front of the double-quote. With `csv-escape-chars`

3. Databases (datatool package)

=none the backslash isn't removed, so the value with `csv-content=literal` will end up as `Chlo\textbackslashash "e` (`Chlo\"e`). Whereas with `csv-content=tex` the value will end up as `Chlo\"e` (`Chloë`). With the other `csv-escape-chars` settings, the backslash will be removed in both cases, and the value will be `Chlo"e`. Since the category code for the delimiter is automatically set to 12 (other) at the start of `\DTLread`, that's the category code it will have in the value.

`csv-content=tex`

The content includes $\LaTeX$ markup. These means that grouping can also be used to delimit values. The file is read line by line, but any line break occurring within a group will ensure that subsequent lines are appended until the grouping is balanced. Once the line has been read, it will then be split on the `separator` and the `delimiter` pairs will be removed. Any escape backslashes (`\`) will be stripped according to `csv-escape-chars`.

`csv-content=no-parse`

This is similar to `csv-content=tex` but the items are not parsed to determine their data type. If `data-types` has not been set then each item will be assumed a string (even if it's empty). If `data-types` is set, then if the current column type is numerical then the item is expected to be a plain number (even if the data type is currency). If the current column number is greater than the number of items in `data-types` then the final data type listed will be assumed for all remaining columns. See Example 108.

For example, if you have a CSV file with six columns of plain numbers then the file can be read with:

```
\DTLsetup{store-datum}  
\DTLread[  
  format=csv,  
  csv-content=no-parse,  
  data-types=decimal,  
{data.csv}
```

Note that this won't check that the values are actually numerical (unless you have `convert-numbers` on). If they're not and you try performing a numerical operation with them, you will get an error.

If you do intend using the values in a numerical operation that expects formatted numbers (including plotting) then switch on the `store-datum` setting before loading the data, otherwise the values will be parsed during the numerical operation according to the current localisation settings.

If you specify a column as currency, each item in that column must still be a plain number without a currency symbol. In this case, the string part of the element will be set to the given item encapsulated with `\DTLcurrency` unless you also have the `convert-numbers` setting

3. Databases (datatool package)

on, in which case `\DTLdecimaltocurrency` will be used instead.

If `convert-numbers` is on, the plain numbers will be converted according to the data type for the current column.

```
csv-content=literal
```

The content should be interpreted literally. Each line in the file is read in as a detokenized string, which is then split according to the `separator` and `delimiter`. Each element is then processed according to the following steps:

1. strip any backslashes according to `csv-escape-chars`;
2. perform a “replace all cases” regular expression which substitutes the sequences `\n`, `\r` and `\f` with a space character, the sequence `\t` with a tab character and the $\text{\TeX}$ special characters with $\text{\LaTeX}$ commands (see Table 3.1);
3. rescan the value to ensure all tokens have their correct category code according to the current setting;
4. apply user mappings.

The regular expression in the second step uses `\regex_replace_case_all:nN` with the cases provided in the token list variable:

```
\l_datatool_str_csv_regex_cases_tl
```

The default substitutions are listed in Table 3.1. If you want to redefine this token list, remember that the input string will only contain “other” and space tokens when the substitution is performed.

The final user mappings are applied after the value has been rescanned. There are none by default, but mappings can be added with:

```
\DTLrawmap{\langle original \rangle} {\langle replacement \rangle}
```

This indicates that all instances of *\langle original \rangle* should be replaced by *\langle replacement \rangle*. Note that this appends the mapping. It won’t override an existing mapping for *\langle original \rangle*. This command was originally provided for use with `\DTLloadrawdb` and is retained for backward-compatibility.

```
csv-escape-chars=\langle value \rangle initial: double-delim
```

Determines if a literal instance of the delimiter should simply be doubled (the standard for CSV files) or whether or not the backslash (`\`) and `delimiter` characters that occur within a value in a CSV or TSV file should be escaped. (That is, the character should have a backslash inserted in front of it.)

This setting is only applicable with `format=csv` and `format=tsv`. Note that if a value contains the `separator` character, it should be delimited.

Table 3.1.: Mappings Used with `csv-content=literal` Before Re-Scanning

Original	Substituted
#	\#
\$	\\$
%	\%
&	\&
\	\textbackslash
^	\textasciicircum
_	_
{	\{
}	\}
~	\textasciitilde
\f	<i>space</i>
\n	<i>space</i>
\r	<i>space</i>
\t	<i>tab character</i>

`csv-escape-chars=double-delim`

The delimiter should be doubled.

`csv-escape-chars=delim`

Only the delimiter should be escaped. `\DTLwrite` will insert a leading backslash and `\DTLread` will strip a leading backslash from the `delimiter` character but not from a backslash character.

`csv-escape-chars=delim+bksl`

Both the `delimiter` and backslash characters should be escaped. `\DTLwrite` will insert a leading backslash and `\DTLread` will strip a leading backslash from the `delimiter` and backslash characters.

`csv-escape-chars=none`

No escaping. `\DTLwrite` won't insert a leading backslash or double the delimiter and `\DTLread` won't strip a leading backslash or convert a double delimiter to a single instance.

`csv-skip-lines=<value>`

initial: 0

The value may be the keyword `false` (which is equivalent to `csv-skip-lines=0`) or a

3. Databases (datatool package)

non-negative integer. If the value is greater than zero, then `\DTLread` will skip the first $\langle n \rangle$ lines in a `format=csv` or `format=tsv` file where $\langle n \rangle$ is the supplied value. This option has no effect on other formats.

Note that with `csv-content=tex`, a “line” may actually cover multiple lines if a line break occurs within a group. For example, if the file starts with:

```
Name , Score (\%), {Award
(\$) }
"Chlo\"e", 89, 5.50
```

Then with `csv-content=literal` and `csv-skip-lines=1` then the first line to be parsed will be the line:

```
(\%) }
```

whereas with `csv-content=tex` and `csv-skip-lines=1`, the first line to be parsed will be the line:

```
"Chlo\"e", 89, 5.50
```

```
data-types= $\langle list \rangle$ 
```

The value should be a comma-separated list of keywords that identify the corresponding column data type. The keywords may be one of: `unknown`, `string`, `integer`, `decimal`, `currency`, `datetime`, `date`, or `time`. Only has an effect when `\DTLread` parses `format=csv` or `format=tsv` files. This option is ignored by the other formats.

The column data type will be updated if a value in that column is found to be incompatible with the identified type (except in the case of `csv-content=no-parse`).

If there are more columns than are listed in `data-types` then `csv-content=no-parse` will assume all remaining columns have the same type as the last listed (or string if the list is empty). With the other `csv-content` settings, any remaining columns will updated as usual, according to the data type identified from parsing the items in the column. See Example 108.

```
delimiter= $\langle char \rangle$ 
```

initial: "

Sets the delimiter for `format=csv` and `format=tsv` files to $\langle char \rangle$, which must be a single token. This setting is ignored by `format=dbtex` and `format=dtltx` files.

The default delimiter may also be set via the `delimiter` package option or with:



```
\DTLsetdelimiter{<char>}
```

If you don't want a delimiter when saving a file, use the `add-delimiter=never` option.



```
expand=<value>
```

*default: **protected**; initial: **varies***

This option governs element expansion when the data is written to the file for all formats. This option also changes `new-value-expand`, which affects element expansion when data is read in from a file, except for the `dbtex` formats where no expansion is applied to the values. If the value is omitted, `expand=protected` is assumed.

The default setting for `\DTLwrite` is `expand=none`. The default for `\DTLread` is the current `new-value-expand` setting. If the value is omitted, `expand=protected` is assumed. The value may be one of the following.



```
expand=none
```

No expansion. Automatically implements `new-value-expand=false`.



```
expand=protected
```

Protected expansion, except for the `dbtex` formats where this option is equivalent to `expand=none`. Automatically implements `new-value-expand=true`.



```
expand=full
```

Full expansion. Make sure that the database doesn't contain fragile commands with this setting. Automatically implements `new-value-expand=true`.

Bear in mind that if you add content to databases that contain characters that should have non-standard category codes, this information may be lost unless it's hidden inside a robust command that ensures the correct category codes are used. Normally with `expand=none`, `\DTLwrite` will prevent expansion of an element while writing to a file. However, with the DBTEX formats, if a database element starts with the special datum item markup or if the element starts with the command `\dtlspecialvalue` then the datum item will be stripped in DBTEX v2.0 and `\dtlspecialvalue` will be allowed to expand. (The `datagidx` package uses this to clear the "Used" and "Location" columns when writing the index/glossary database to a file for the next run.)



```
format=<value>
```

*initial: **csv***

Indicates the file format for both `\DTLread` and `\DTLwrite`. The default setting is `format=csv` (even if you have separately set the separator to the tab character). The format

3. Databases (datatool package)

may be one of the following values:

```
format=csv
```

The CSV file format with the separator given by the `separator` option and the delimiter given by the `delimiter` option. The default file extension is set to `csv`. Note that the designated separator and delimiter will have their category code set to “other”.

```
format=tsv
```

The TSV file format with a tab character separator and the delimiter given by the `delimiter` option. The default file extension is set to `csv`.

If the `separator` option is specified *after* this option, then that separator will be used instead, but the file extension will still default to `tsv`.

The following formats are all files that contain L^AT_EX code and all use the same underlying function with `\DTLread`. The file is input as per a normal L^AT_EX file with `global=true` and scoping to limit the effect of the options and some local redefinitions. In terms of `\DTLread`, the difference between the specific `format` values simply determines the default file extension.

```
format=dtltx-2
```

This indicates DTLTEX v2.0 format (see §3.15.1.2). The default file extension is set to `dtltx`.

When used with `\DTLwrite`, the data will be written to the file with document level user commands that include the database name, such as `\DTLnewdb`.

```
format=dtltx-3
```

This indicates DTLTEX v3.0 format (see §3.15.1.2). The default file extension is set to `dtltx`.

When used with `\DTLwrite`, the data will be written to the file with v3.0 document level user commands described in §3.15.1.2.

```
format=dtltx
```

The latest DTLTEX format. This is currently equivalent to `format=dtltx-3`.

```
format=dbtex-2
```

This indicates DBTEX v2.0 format (see §3.15.1.3). The default file extension is set to `dbtex`.

When used with `\DTLwrite`, the data will be written to the file in DBTEX v2.0 format,

3. Databases (datatool package)

which uses low-level internal commands to define and set the registers used to store the data. Note that this will cause any instance of @ in the database to have a letter category code (see Example 111) and leading spaces at the start of database elements will be lost. The DBTEX v3.0 format is better.

```
format=dbtex-3
```

This indicates DBTEX v3.0 format (see §3.15.1.3). The default file extension is set to dbtex.

When used with \DTLwrite, the data will be written to the file in DBTEX v3.0 format, which uses higher level internal commands.

```
format=dbtex
```

The latest DBTEX format. This is currently equivalent to format=dbtex-3.

```
headers=<list>
```

Identifies the column headers to be used when \DTLread parses format=csv or format=tsv files. This option is ignored by the other formats.

Leading and trailing spaces and empty elements in <list> will be stripped, regardless of the trim and skip-empty settings. If you specifically want to retain these, you will need to use braces around the item.

If headers is set to empty (the default), or there is no item in the <list> that corresponds to a given column, then that column header will be the same as the column key.

For example:

```
\DTLread{headers={ Name , , Email, {}, Notes, }}  
{<csv-file>}
```

This is equivalent to:

```
\DTLread{headers={Name, Email, {}, Notes}}  
{<csv-file>}
```

The first column will be given the header “Name” and the second column will be given the header “Email”. The third column will have an empty header, and the fourth column will be given the header “Notes”. If a fifth column is found in the file, the header for that column will be the same as the key for that column.

```
keys=<list>
```

Identifies the column keys to be used when `\DTLread` parses `format=csv` or `format=tsv` files. This option is ignored by the other formats.

Leading and trailing spaces and empty elements in `<list>` will be stripped, regardless of the `trim` and `skip-empty` settings. If you specifically want to retain these, you will need to use braces around the item.

If `keys` is set to empty and `auto-keys=false` (the default), then the keys will be the column headers supplied in the file. If there are no column headers (`no-header=true`) or the header is empty for a given column, then the corresponding key in `<list>` will be used. If there is no corresponding key in `<list>`, or if the corresponding key is empty, then the default key is used.

If `keys` is used with a non-empty list (after stripping extraneous spaces and commas) then it will automatically implement `auto-keys=false`. If `auto-keys=true` is subsequently used, it will override the `keys` setting.

For example:

```
\DTLread{keys={ Name , , Email, {}, Notes, }}
{ <csv-file> }
```

This is equivalent to:

```
\DTLread{keys={Name, Email, {}, Notes}}
{ <csv-file> }
```

This will set the key to “Name” for the first column and “Email” for the second. The third column will either pick up the key from the header row in the file or, if that is missing, the key will be `\dtldefaultkey <n>` (“Column3”). The fourth column will have the key “Notes”. If additional columns are found in the file, they will act as for an empty element in the list. So a fifth column will either pick up the key from the header row in the file or, if that is missing, the key will be `\dtldefaultkey <n>` (“Column5”).

```
load-action=<value>
```

initial: old-style

Determines whether or not `\DTLread` should create a new database or append to an existing

3. Databases (datatool package)

one. This append setting does not support `format=dtltex-2` or any dbtex format.



Remember that the database name depends on the `name` setting and the format.



`load-action=detect`

Detects the appropriate action: if a database with the given name exists, it behaves as `load-action=append`, otherwise it behaves as `load-action=create`.



`load-action=create`

A new database will be created. If a database already exists, an error will occur.



`load-action=append`

A new database won't be created. If the database doesn't already exist, an error will occur. Appended data will match on the column key not the index. This may cause null values in the database if there are extra or missing columns in the appended data.



`load-action=overwrite`

If the database with the given name already exists, it will be cleared first instead of attempting to define it.



`load-action=old-style`

For backward-compatibility with old versions of datatool, this setting will test the conditional:



```
\ifDTLnewdbonload <true>\else <false>\fi initial: \iftrueDeprecated
```

If true, this behaves like `load-action=create` otherwise it behaves like `load-action=append`.



`name=<value>`

The database name. This option is supported by `\DTLwrite` for all formats, and identifies the database to save. If omitted, the general `default-name` setting will be used.

Note that the argument is expanded when the option is set. For example:

```
\newcommand{\mydatacmd}{mydata}
\DTLsetup{io={name=\mydatacmd}}
\renewcommand{\mydatacmd}{otherdata}
```

In the above, the database name remains “mydata” after `\mydatacmd` is redefined.

With `\DTLread`, this option identifies the database name but isn’t supported by all formats. This option is ignored by `dtltex-2` and `dbtex-2`. If omitted, the default behaviour depends on the format: `csv` and `tsv` will fallback on the `default-name` setting, but `dbtex-3` and `dtltex-3` will use the name provided in the file.

`no-header`=*<boolean>*

initial: **false**

A boolean option that determines if `\DTLwrite` should omit header information in the file. This option has no effect with `format=dtltex-2`, `format=dbtex-2` and `format=dbtex-3`.

With `\DTLread`, this option only has an effect with the `format=csv` and `format=tsv` formats. If `no-header=true`, then there’s no header row in the file (taking into account any offset introduced with `csv-skip-lines`). The column keys will either be obtained from the `keys` setting or will be set to the default for the given column index. The column headers will either be obtained from the `headers` setting or will be set to the key.

With `dtltex` files, this option will locally redefine the underlying command used by `\DTLsetheader` to do nothing, unless a corresponding value is found in the `headers` option. This means that the header will be the same as the column key unless the `headers` value supplies an alternative.

With `\DTLwrite`, this option will omit the header line for the `format=csv` and `format=tsv` formats. So the database content will start on the first line of the file. With `format=dtltex-3`, this will omit the code that sets the column headers (but the column keys will still be used).

`noheader`=*<boolean>*

alias: `no-header`

A synonym of `no-header`.

`only-reformat-columns`=*<list>*

initial: **empty**

Similar to the `auto-reformat` numeric option and the `auto-reformat` datetime option. The value should be a comma-separated list of column index numbers.

This option identifies the indexes of the columns that should be automatically reformatted (but only for data types identified by `auto-reformat-types`) when `\DTLread` parses `format=csv` or `format=tsv` files. This option is ignored by the other formats.



The option simply locally switches on the corresponding `auto-reformat` numeric option and the `auto-reformat` datetime option when parsing a column that's included in the `only-reformat-columns` list. If the list is empty, then no change will be made, so whatever setting was in effect before the file was opened will be obeyed.



`omitlines=<n>`

initial: 0

Provided for backward compatibility, this option is like `csv-skip-lines` but doesn't allow the keyword `false` and doesn't trigger an error for negative values.



`overwrite=<value>`

initial: **error**

A boolean option that governs whether or not an existing file should be overwritten. Only applicable with `\DTLwrite`. The value may be one of the following.



`overwrite=error`

Trigger an error and don't allow the file to be overwritten.



`overwrite=warn`

Trigger a warning and don't allow the file to be overwritten.



`overwrite=allow`

Allow the file to be overwritten.



`separator=<char>`

initial: ,

Sets the separator for CSV files to `<char>`, which must be a single token.

The default separator may also be set with:



`\DTLsetseparator{<char>}`

The tab character in TSV files is awkward as the tab character is usually treated the same as a space by $\text{\LaTeX}$. This means that in order to read a tab character correctly, the category code first needs to be changed. The following command:



`\DTLsettabseparator`

changes the category code of the tab character to 12 (“other”) and then sets the tab character as the separator. Note that this will affect any tab characters in the document code unless the change is localised. The simplest method is to use the setting `format=tsv`.

```
trim=<boolean>
```

Equivalent to `\DTLsetup{new-value-trim=<boolean>}`. Only applicable with `\DTLread` but not for the `dbtex` formats.

3.15.3. Loading a Database from an External File

```
\DTLread[<options>]{<filename>}
```

Loads the data from the file given by `<filename>` and globally defines a database containing that data. If the file extension is omitted, the default extension associated with the format will be used. The options are as for those listed in §3.15.2 that are identified as working with `\DTLread`.

The CSV and TSV files don’t have the database name included in the file, so the `name` option may be used to specify the database name to override the `default-name` setting. The `name` option may also be used to override the name supplied in DBTEX 3.0 and DTLTEX 3.0 files, but the name is hardcoded in DBTEX 2.0 and DTLTEX 2.0 files, so the `name` option will have no effect (other than to generate a warning).

After the file has been read, the command `\dtllastloadeddb` is defined to expand to the database name.

For example, the “xydata” database described in §3.2.11 is in a CSV file with two columns of numbers with decimal points. Since there are no special characters to worry about, if the current decimal character is a decimal point then either `csv-content=literal` or `csv-content=tex` may be used, but `csv-content=tex` is faster, since it doesn’t need to convert any characters. However, since these are all plain numbers, it’s even faster to load with `csv-content=no-parse`.

Example 107 loads the data with `csv-content=no-parse` but doesn’t identify the data as decimal values:

107

```
\DTLread[
  name=xydata,
  format=csv,
  csv-content=no-parse
]{xydata.csv}
```

This will assume that all content is just text, which becomes evident when the data is sorted and displayed:

3. Databases (datatool package)

```
\DTLsortdata{xydata}{Y}
\DTLdisplaydb{xydata}
```

Because the columns are identified as having the string data type, the sorting uses a string comparison and `\DTLdisplaydb` uses left alignment.

↑ Example 107: Loading Data With No Parsing

X	Y
3.2	-0.4
-2.5	-1
-3.5	-2.75
1	-4.2
-1	1.5
2.6	1.8
-3	3

Example 108 identifies the data as decimal values and also sets `store-datum` beforehand, which means that values won't have to be parsed later:

108

```
\DTLsetup{store-datum}
\DTLread[
  name=xydata,
  format=csv,
  csv-content=no-parse,
  data-types=decimal
]{xydata.csv}
```

Again the data is sorted (on column Y) and displayed as before, but now the sorting is numerical and the columns are right aligned. Note that with `csv-content=no-parse`, the `data-types` setting doesn't need a type for each column. The type for just the first column is set and following columns are assumed to have the same type. This is different from the behaviour of `data-types` when parsing is on.

↑ Example 108: Loading Data With No Parsing and Columns Identified as

Decimal X	Y
1	-4.2
-3.5	-2.75
-2.5	-1
3.2	-0.4
-1	1.5
2.6	1.8
-3	3

Example 109 identifies the data in the first column as decimal and the data in the second column as currency. Note that with `csv-content=no-parse`, columns identified as currency must also be plain numbers without a currency symbol (as is the case here):

109

```
\DTLsetup{store-datum}
\DTLread[
  name=xydata,
  format=csv,
  csv-content=no-parse,
  data-types={decimal,currency}
]{xydata.csv}
```

Again the data is sorted (on column Y) and displayed as before. Note that the second column has the original value encapsulated with `\DTLcurrency`. Since `store-datum` is on, the actual value provided in the CSV file is embedded in the data for easy access.

↑ Example 109: Loading Data With No Parsing and Columns Identified as

Decimal and Currency

X	Y
1	-\$4.2
-3.5	-\$2.75
-2.5	-\$1
3.2	-\$0.4
-1	\$1.5
2.6	\$1.8
-3	\$3

Example 110 has the localisation settings on. (This will require `datatool-regions` and `datatool-english` to also be installed.)

110

3. Databases (datatool package)

```
\usepackage[locales=en-BE]{datatool}
```

This means that the decimal character is now a comma not a decimal point, but this isn't a problem because `csv-content=no-parse` is used. However, the `convert-numbers` option will reformat the values according to the current localisation setting. (Note that the `auto-reformat` option isn't applicable with `csv-content=no-parse` as no parsing takes place.)

```
\DTLsetup{store-datum}  
\DTLread[  
  name=xydata,  
  format=csv,  
  csv-content=no-parse,  
  data-types={decimal,currency},  
  convert-numbers  
{xydata.csv}
```

Again the data is sorted (on column Y) and displayed as before.

Example 110: Loading Data With No Parsing and Columns Identified as Decimal and Currency with Reformatting

X	Y
1	-4,20€
-3,5	-2,75€
-2,5	-1,00€
3,2	-0,40€
-1	1,50€
2,6	1,80€
-3	3,00€

```
\DTLloaddbtex{<cs>}{<filename>}
```

Deprecated

Inputs the file identified by `<filename>` and defines the control sequence `<cs>` to expand to the database name. In datatool v3.0, this command has been rewritten to simply do:

```
\DTLread[name={},format=dbtex]{<filename>}  
\let<cs>\dtllastloadeddb
```

```
\DTLloadadb[⟨options⟩]{⟨db-name⟩}{⟨filename⟩}
```


 Deprecated

This deprecated command now simply does:

```
\DTLread[name={⟨db-name⟩},format=csv, csv-content=
tex,⟨options⟩]{⟨filename⟩}
```

```
\DTLloaddrawdb[⟨options⟩]{⟨db-name⟩}{⟨filename⟩}
```


 Deprecated

This deprecated command now simply does:

```
\DTLread[name={⟨db-name⟩},format=csv, csv-content=
literal,⟨options⟩]{⟨filename⟩}
```

3.15.4. Saving a Database to an External File

```
\DTLwrite[⟨options⟩]{⟨filename⟩}
```



Saves the data from a database to the file given by *⟨filename⟩*. If the file extension is omitted, the default extension associated with the format will be used. The options are as for those listed in §3.15.2 that are identified as working with `\DTLread`. The `name` option identifies the database. If that setting isn't provided, the `default-name` is assumed.

```
\DTLsavedb{⟨db-name⟩}{⟨filename⟩}
```


 Deprecated

This deprecated command now simply does:

```
\DTLwrite[name={⟨db-name⟩},overwrite=warn,format=
csv,expand=none,add-delimiter=detect]{⟨filename⟩}
```

```
\DTLsaverawdb{⟨db-name⟩}{⟨filename⟩}
```


 Deprecated

This deprecated command now simply does:

```
\DTLwrite[name={⟨db-name⟩},overwrite=warn,format=dbtex-
2,expand=full]{⟨filename⟩}
```



```
\DTLprotectedsave rawdb{<db-name>} {<filename>}
```


 Deprecated

This deprecated command now simply does:

```
\DTLwrite[name={<db-name>}, overwrite=warn, format=dbtex-2, expand=none] {<filename>}
```

```
\DTLsave texdb{<db-name>} {<filename>}
```


 Deprecated

This deprecated command now simply does:

```
\DTLwrite[name={<db-name>}, overwrite=warn, format=dtl tex-2, expand=full] {<filename>}
```

3.15.5. I/O Examples

The “customers” database (see §3.2.3) may be loaded from the CSV file `customers.csv`:

 111

```
\DTLread[name=customers, format=csv] {customers.csv}
```

Alternatively, you can setup the default database name first, to avoid having to repeatedly specify it. The file extension may also be omitted, as can `format=csv` which is the default:

```
\DTLsetup{default-name=customers}
\DTLread{customers}% parse customers.csv
```

Example 111 does this in the preamble. Setting the default name makes it easier to use actions without having to repeatedly write the database name.

The `select row` action can be used to find the row where the `Email` column is set to the email address `fc@example.com`. If successful, the row index can be accessed with `\dtlrownum`.

```
\DTLaction[key=Email, value=fc@example.com]
{select row}
Row: \number\dtlrownum.
```

The database hasn’t been modified, but it can be saved to the DBTEX v3.0 format with:

3. Databases (datatool package)

```
\DTLwrite[format=dbtex-3,overwrite=allow]
{customers-v3}
```

(The `overwrite` setting allows the test document to be re-compiled afterwards without triggering an error.) This will create a file called `customers-v3.dbtex`. (If you try this example, compare the DBTEX v3.0 file with and without the `store-datum` setting on.)

Example 111 then reads this new file back in with:

```
\DTLread[format=dbtex,name=customers-v3]
{customers-v3}
```

Note that although the DBTEX v3.0 file format includes the database name (which will be “customers” in this example), this can be overridden with the `name` option (but not with `default-name`, which can’t be used to override the database name if it’s hard-coded in the file).

This has created a second identical database called “customers-v3”. The `select row` action is again used to look up the row with the email `fc@example.com` but note that the database name now needs to be specified, since it’s not the default:

```
\DTLaction[
  name=customers-v3,
  key=Email,value=fc@example.com
]{select row}
Row: \number\dtlrownum.
```

Example 111 then re-saves the original “customers” database in the DBTEX v2.0 format. (The default name is still set to “customers”.)

```
\DTLwrite[format=dbtex-2,overwrite=allow]
{customers-v2}
```

This creates a file called `customers-v2.dbtex`.

The DBTEX v2.0 format has the database name hard-coded in the file and doesn’t allow it to be changed (even if the `name` option is used) nor does it support any load action other than `load-action={create}`. This means that the “customers” database must be deleted before this new file can be loaded:

```
\DTLaction{delete}
\DTLread[format=dbtex]{customers-v2}
```

3. Databases (datatool package)

This should in theory still be identical to the original but it's not because the DBTEX v2.0 file format requires the category code of @ to be set to “letter”. This means that the row look up will now fail.

```
\DTLaction[key=Email,value=fc@example.com]
{select row}
Row: \number\dtlrownum.
```

This results in “Row: 0” (without an error) indicating that no match was found because the @ character in `value=fc@example.com` has its usual “other” category code, but the `fc@example.com` within the database has the letter category code.

↑ Example 111: Loading and Saving Data (Be Careful of Category Codes)

Row: 7.
Row: 7.
Row: 0.

As described in §3.2.10, the “growthdata” database can be obtained by parsing the example file `growth.tsv` as follows:

```
\DTLsetup{store-datum,default-name=growthdata}
\DTLread[
  format=tsv, csv-skip-lines=1,
  keys={Exp1Time, Exp1Count, Exp2Time, Exp2Count},
  csv-content=no-parse,
  data-types=decimal
]{growth}
```

This is done in Example 112 and the data is then displayed using the `display` action:

```
\DTLaction{display}
```

112

↑ Example 112: Loading a TSV File

Time	Log Count	Time	Log Count
0	2.9	0	3.31
15	3.14	10	3.45
30	3.26	25	3.61
45	4.01	40	3.76
60	4.2	55	3.89

As described in §3.2.8, the “profits” database can be obtained by parsing the example file `profits.csv`. This has three columns, “Year” (integers), “Profit” (currency) and “Units” (integers). For this example, I have the localisation support set to “en-US”, which matches the dollar symbol used in the “Profit” column.

113

```
\usepackage[locales={en-US}]{datatool}
```

Some of the later examples that use this database (see §5) need the numerical values, so it’s useful to switch on the `store-datum` setting to prevent repeated parsing, and I’ve also set the default name:

```
\DTLsetup{store-datum,default-name=profits}
```

The second column is a little untidy as it has a mix of some negative currency with the negative sign before the currency symbol and some with the sign after the symbol. The third column is also missing the number group characters.

The `auto-reformat` numeric option can be set before reading the CSV file, but that will also reformat the first column, which contains integers, but as they represent years they shouldn’t be converted to formatted numbers.

The `auto-reformat-types` option could be used to omit integers from being reformatted, but that would prevent the Unit column (which also contains integers) from being reformatted. In this case, the `only-reformat-columns` option can be used to indicate that only columns 2 and 3 should be reformatted:

```
\DTLread[
  format=csv,
  csv-content=tex,
  only-reformat-columns={2,3},
]{profits.csv}
```

3. Databases (datatool package)

With localisation support, this reformatting will use the formatting commands that are sensitive to the region's currency settings.



The number group character and decimal character need to be established before parsing. Reformatting will insert the relevant characters, and they won't be affected by any later change.

This means that the currency style can be modified:



```
\DTLsetLocaleOptions{US}{currency-symbol-sep=thin-space}
```

The data is then displayed using the `display` action:



```
\DTLaction{display}
```



↑ Example 113: Automatically Reformatting Data While Loading a CSV file



Year	Profit	Units
1999	−\$ 4,673.00	12,467
2000	\$ 2,525.49	8,965
2001	\$ 1,673.52	14,750
2002	−\$ 1,320.01	14,572
2003	\$ 5,694.83	13,312
2004	−\$ 451.67	9,764
2005	\$ 6,785.20	11,235

3.16. Advanced Database Commands



```
\dtlgetrowindex{<row-cs>}{<db-name>}{<col idx>}{<value>}
```

Gets the row index for the first row in the database identified by `<db-name>` where the entry in the column given by its index `<col idx>` exactly matches the given `<value>`. If successful, `<row-cs>` will be defined to expand to the row index, otherwise it will be set to the null value.

3. Databases (datatool package)

```
\DTLgetrowindex{<row-cs>}{<db-name>}{<col idx>}{<value>} modifier: *
```

The starred version simply does `\dtlgetrowindex`. The unstarred version triggers an error if not found. (That is, if `<row-cs>` is set to null.)

```
\xdtlgetrowindex{<row-cs>}{<db-name>}{<col idx>}{<value>}
```

As `\dtlgetrowindex` but expands the value.

If `store-datum=true` was set when the database was constructed, remember that the elements in the database will be datum items which means that the value must be in the same format.

```
\DTLgetvalue{<cs>}{<db-name>}{<row idx>}{<col idx>}
```

Gets the element in the row identified by `<row idx>` for the column identified by its index `<col idx>` in the database identified by `<db-name>` and defines the control sequence `<cs>` to expand to the element's value.

```
\DTLgetlocation{<row-cs>}{<col-cs>}{<db-name>}{<value>}
```

Defines the control sequences `<row-cs>` and `<col-cs>` to expand to the row and column indexes, respectively, of the first entry in the database identified by `<db-name>` that matches the given `<value>`.

```
\DTLassignfirstmatch{<db-name>}{<col key>}{<value>}{<assign-list>}
```

This internally uses `\dtlgetrowindex` to find the row where the entry in the column identified by its label `<col key>` exactly matches the given value and (globally) performs the placeholder assignments in `<assign-list>`, which should be a comma-separated `<cs>=<key>` assignment list. Unlike `\DTLassign`, this command doesn't change the current row.

For historical reasons, some of the older commands, such as `\DTLassignfirstmatch`, perform global assignments when setting the placeholder commands. This is because the underlying code is shared by `\DTLforeach`, which was designed to work within tabular-like environments and so had to make global assignments to avoid the scoping created by the cells within the tabular body (see §3.9). Be aware that this can

cause problems.

If you need to fetch more than one entry for a particular row, you can use the `find` action to find the required row and make the assignments. Alternatively, you can select a row to identify it as the current row and then use the “current row” functions. For example, the `current row aggregate` may be used to aggregate the data in the current row, or the row editing commands may be used to edit the current row (see §3.12).

```
\dtlswaprows { <db-name> } { <row1-idx> } { <row2-idx> }
```

Swaps the rows identified by their row index. No check is performed to determine if the database exists or if the indexes are in range.

3.16.1. Operating on Current Row

Some iterative commands, such as `\DTLdisplaydb`, fetch the content of the current row and store it in the token register:

```
\dtlcurrentrow
```

The content is in a special format that makes it easier to access elements within the row. The preceding rows are stored in the token register:

```
\dtlbeforerow
```

and the following rows are stored in the token register:

```
\dtlafterrow
```

The row index will be stored in the count register:

```
\dtlrownum
```

Commands like `\DTLdisplaydb` set `\dtlrownum` to the filtered row index. If rows have been omitted, `\dtlrownum` will be less than the actual database row index.

The placeholder command:

3. Databases (datatool package)

```
\dtldbname
```

is defined to expand to the current database name.

Individual entries in the current row can be fetched with:

```
\dtlgetentryfromcurrentrow{<cs>}{<col num>}
```

This gets the value from the `\dtlcurrentrow` register for the column with the index `<col num>` and stores the value in the control sequence `<cs>`. If you only know the column key, you can obtain the column index with `\dtlcolumnindex`.

Multiple entries in the current row can be fetched with:

```
\DTLassignfromcurrentrow{<assign-list>}
```

The `<assign-list>` argument should be a comma-separated `<cs>=<key>` list, where `<cs>` is a placeholder command (token list variable) and `<key>` is the column key. Each `<cs>` will be defined to the entry in the column identified by `<key>` or to null if the column isn't set in the current row.

It's also possible to fetch the row information using `\DTLaction` with the `current row values` or `current row aggregate` actions.

If you're not within one of the iterative commands that sets up the `\dtlcurrentrow` and associated registers, you can select the current row by its index with:

```
\dtlgetrow{<db-name>}{<row idx>}
```

Before using `\dtlgetrow`, check that the database exists and contains the required row number. This can be done with `\DTLifdbexists` (to test for existence) and test that the row index is between 1 and the row count (which can be obtained with `\DTLrowcount`).

If you don't know the applicable row index, you can also set the current row with:

```
\dtlgetrowforvalue{<db-name>}{<col idx>}{<value>}
```

This is similar to `\dtlgetrow`, but gets the row where the entry in the column with the given `<column index>` exactly matches the given `<value>`. Internally `\dtlgetrowforvalue` uses

3. Databases (datatool package)

`\dtlgetrowindex` to first fetch the row index. If the result is null then there was no match and an error will occur, otherwise the row index is then used to select the current row with `\dtlgetrow`.

No expansion is performed on `<value>`. If you want `<value>` to be expanded before comparison, use: `\edtlgetrowforvalue` This performs a protected expansion and then uses `\dtlgetrowforvalue` on the expanded value.



If `store-datum=true` was set when the database was constructed, remember that the elements in the database will be datum items which means that the value must be in the same format.



```
\DTLassign{<db-name>}{<row idx>}{<assign-list>}
```

This essentially performs:

```
\dtlgetrow{<db-name>}{<row-idx>}  
\DTLassignfromcurrentrow{<assign-list>}
```

but the assignments are global. This first selects the current row by its row index and then performs the assignments. If you prefer local assignments, then select the row and use `\DTLassignfromcurrentrow` instead. Alternatively, consider using the `select row` action.

The following commands alter the contents of the `\dtlcurrentrow` token register. You will then need to recombine `\dtlbeforerow`, `\dtlcurrentrow` and `\dtlafterrow` in order to reconstruct the database with the amendments.



```
\dtlrecombine
```

This will update the database by merging the contents of `\dtlbeforerow`, `\dtlcurrentrow` and `\dtlafterrow`. The `global` option is checked to determine if the change should be global.



```
\dtlrecombineomitcurrent
```

This will update the database by merging the contents of `\dtlbeforerow` with the rows in `\dtlafterrow`, where the row indexes are adjusted to account for the omitted current row.



Recombining the database while iterating over it can cause problems if the row count or row indexes change. If you are using `\DTLforeach`, use the commands described in

§3.8.2 to alter the current row of the database, not these ones.

`\dtlreplaceentryincurrentrow{ $\langle new\ value \rangle$ }{ $\langle col\ idx \rangle$ }`

Replaces the entry for the column identified by the index $\langle col\ idx \rangle$ in `\dtlcurrentrow` with the given value $\langle new\ value \rangle$. The column data is updated according to the new value honouring the `global` option (but `\dtlcurrentrow` will only be locally changed).

`\dtlappendentrytocurrentrow{ $\langle col\ key \rangle$ }{ $\langle value \rangle$ }`

Appends an entry with the given $\langle value \rangle$ to `\dtlcurrentrow` for the column identified by $\langle col\ key \rangle$. The row must not already contain an element in the given column. The column data is updated according to $\langle value \rangle$.

`\dtlupdateentryincurrentrow{ $\langle col\ key \rangle$ }{ $\langle value \rangle$ }`

Appends the entry, as per `\dtlappendentrytocurrentrow`, if there is no entry for the given column in the current row, otherwise updates the entry.

`\dtlremoveentryincurrentrow{ $\langle col\ idx \rangle$ }`

Removes the entry for the column identified by the index $\langle col\ idx \rangle$ from `\dtlcurrentrow`.

`\dtlswapentriesincurrentrow{ $\langle col1\ num \rangle$ }{ $\langle col2\ num \rangle$ }`

Swaps the values in the columns identified by their index, $\langle col1\ num \rangle$ and $\langle col2\ num \rangle$, in `\dtlcurrentrow`.

`\dtlmapcurrentrow{ $\langle cs \rangle$ }{ $\langle body \rangle$ }`

Iterates over the columns of the current row. At each iteration, $\langle cs \rangle$ is defined to the current column value and $\langle body \rangle$ is performed.

3.16.2. Advanced Iteration

In addition to the iteration commands described in §3.8, the following are also available.

```
\dtlforeachkey (<key-cs>, <col-cs>, <type-cs>, <header-cs>) \in{<db-name>}
\do{<body>}
```

Iterates over each column in the database identified by $\langle db-name \rangle$, and at each iteration defines $\langle key-cs \rangle$ to expand to the column key, $\langle col-cs \rangle$ to expand to the column index, $\langle type-cs \rangle$ to expand to the data type, and $\langle header-cs \rangle$ to expand to the column header, and then does $\langle body \rangle$. Note that this globally defines the loop variables. The loop may be broken with `\dtlbreak`.

If you have L^AT_EX3 syntax enabled, you may prefer the following instead.

```
\datatool_map_keys_function:nn{<db-name>}{function}
```

Maps over all the columns in the given database, applying the function to each set of column meta data. The function should have four arguments $\{ \langle key \rangle \} \{ \langle col-idx \rangle \} \{ \langle type \rangle \} \{ \langle header \rangle \}$ where $\langle key \rangle$ is the column key, $\langle col-idx \rangle$ is the column index, $\langle type \rangle$ is the data type (−1 for unknown) and $\langle header \rangle$ is the column header.

```
\datatool_map_keys_inline:nn{<db-name>}{<def>}
```

Maps over all the columns in the given database, applying the inline function to each set of column meta data. Within $\langle def \rangle$, #1 references the column key, #2 references the column index, #3 references the data type (−1 for unknown) and #4 references the column header.

```
\dtlforcolumn{<cs>}{<db-name>}{<key>}{<body>}
```

*modifier: **

Loops through the column identified by $\langle key \rangle$ in the database identified by $\langle db-name \rangle$ and, for each iteration, defines $\langle cs \rangle$ to the value of the entry in the row. The starred version doesn't check if the database exists. The loop may be broken with `\dtlbreak`.

```
\dtlforcolumnidx{<cs>}{<db-name>}{<col-idx>}{<body>}
```

*modifier: **

Loops through the column with index $\langle col-idx \rangle$ in the database identified by $\langle db-name \rangle$ and, for each iteration, defines $\langle cs \rangle$ to the value of the entry in the row. The starred version doesn't check if the database exists. The loop may be broken with `\dtlbreak`.

The following commands are actually provided by `datatool-base`, but are listed here for completeness.

```
\dtlforint<count-reg>=<start>\to<end>\step<inc>\do{<body>}
```

3. Databases (*datatool* package)

Integer iteration from $\langle start \rangle$ to $\langle end \rangle$, incrementing by $\langle incr \rangle$ using $\langle count-reg \rangle$ as the loop variable. This command is retained for backward-compatibility but L^AT_EX3 now provides integer step functions. The loop may be broken with `\dtlbreak`.

```
\dtlgforint $\langle count-reg \rangle$ = $\langle start \rangle$ \to $\langle end \rangle$ \step $\langle inc \rangle$ \do{ $\langle body \rangle$ }
```

As `\dtlgforint` but globally sets the count register. There are environment versions, but note that the environment body can't contain any verbatim.

```
\begin{dtlenvgforint}{ $\langle count-reg \rangle$ = $\langle start \rangle$ \to $\langle end \rangle$ \step $\langle inc \rangle$ }{ $\langle content \rangle$ }  
\end{dtlenvgforint}
```

Performs `\dtlgforint` where the iteration body is obtained from the environment content with leading and trailing spaces trimmed.

```
\begin{dtlenvgforint*}{ $\langle count-reg \rangle$ = $\langle start \rangle$ \to $\langle end \rangle$ \step $\langle inc \rangle$ }{ $\langle content \rangle$ }  
\end{dtlenvgforint*}
```

As `dtlenvgforint` but doesn't trim the environment body.

4. Pie Charts (datapie package)

```
\usepackage[<options>]{datapie}
```

The datapie package can be used to draw a pie chart obtained from a column in a database. This package automatically loads the datatool package.

Any package options provided when loading datapie will be passed to datatool. If datatool has already been loaded, any options will be passed to `\DTLsetup` instead (which means that you would only be able to use options that can be set after datatool has been loaded).

The datapie package additionally loads the tikz package.

The datapie package was rewritten in version 3.0 to use L^AT_EX3 commands. The xkeyval package has been dropped and the use of `\DTLforeach` has been replaced with `\DTLmapdata`. A number of bugs have also been fixed, which may cause some differences in the output.

Rollback to version 2.32 is available:

```
\usepackage{datapie}[=2.32]
```

Note that if datatool hasn't already been loaded, this will also apply rollback to datatool. Problems may occur if a newer release of datatool has already been loaded.

The principle command provided by datapie is:

```
\DTLpiechart[<condition>]{<settings-list>}{<db-name>}{<assign-list>}
```

This draws a pie chart using data provided in the database identified by *<db-name>*. The *<db-name>* argument may be empty to indicate the default database. The *<settings list>* argument may be a *<key>=<value>* list of options to override the default. See §4.2 for available settings.

The `variable` setting must be provided to identify the column of data to plot.

The *<assign-list>* argument should be a *<cs>=<key>* list of placeholder command assignments suitable for use in `\DTLmapgetvalues`. The `variable` setting must be set to the

applicable `<cs>`. Additional assignments may be added if required for the segment inner or outer labels.

The `<condition>` optional argument allows filtering to be applied. This should be in the syntax allowed for the first argument of `\ifthenelse` (see §2.4.2). Note that if this option is provided, it will override the `include-if/include-if-fn` setting.

There is also a corresponding action, which has optional settings: `name` (for the database name), `assign` (for the assignment list corresponding to the `<assign-list>` argument of `\DTLpiechart`), and `options` (for the pie chart settings corresponding to the `<settings-list>` argument).

```
\DTLaction[<settings>]{pie chart}
```

The `pie chart` action is equivalent to using `\DTLpiechart`. If you include the `key` or `column` action option to identify the column to use for the variable then you can omit `variable` in the `options` setting. See Example 115 for a simple example.

If `variable` is included in `options`, it will override the action `key` or `column` setting. However, if `variable` has been previously set within the `pie` option in `\DTLsetup` and does not occur in the action `options` then the action `key` or `column` setting will be used (if provided).

If you identify the required column of data with `key` or `column` then a temporary placeholder command will be created and added to the assignment list. If you don't need to use any of the database values in hooks or options (such as the segment labels) then you may omit the `assign` action option or just assign the placeholders required for the labels. You will still be able to use `\DTLpievariable` or `\DTLpiepercent`.

The examples in this chapter use the “fruit” database (see §3.2.7), which has two columns identified by the labels “Name” and “Quantity”. The Quantity column is numeric and so can be used to create a pie chart.

Example 114 uses the following code to create a simple pie chart:

```
\DTLpiechart
{variable=\Quantity}% variable required
{fruit}% database
{\Quantity=Quantity}% assignment list
```

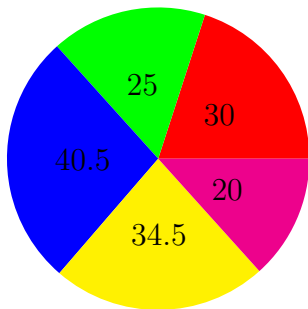
The segment colours are the default. See Example 124 for an example that changes the default colours. Note that in this case, only the Quantity column needs to be referenced so only one assignment is made in the final argument. See Example 120 for an example that changes the inner and outer labels, with an extra assignment to allow the corresponding Name item to be shown.

Example 115 creates the same chart as Example 114 using the `pie chart` action instead

114

115

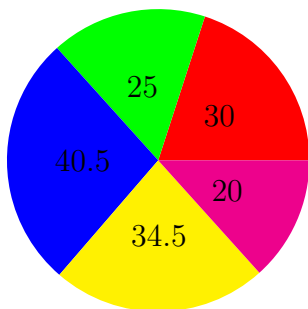
↑ Example 114: Pie Chart



of \DTLpiechart:

```
\DTLaction
[
  key=Quantity % variable
]
{pie chart}
```

↑ Example 115: Pie Chart (Action 'pie chart')



4.1. Package Options

The options listed here may be passed as package options when loading *datapie*. Unless otherwise stated, these options can't be used in `\DTLsetup`. Additional options that may be set with `\DTLsetup` are listed in §4.2.

color

Initialises the default colour list to: red, green, blue, yellow, magenta, cyan, orange, white. This package option is equivalent to `segment-default-colors` and is the default.

gray

Initialises the default colour list to eight shades of grey. This package option is equivalent to `segment-default-gray`.

rotateinner

Indicates that inner labels should be rotated. This package option is equivalent to the `rotate-inner=true` setting.

norotateinner

Indicates that inner labels should not be rotated. This package option is equivalent to the `rotate-inner=false` setting and is the default.

rotateouter

Indicates that outer labels should be rotated. This package option is equivalent to the `rotate-outer=true` setting.

norotateouter

Indicates that outer labels should not be rotated. This package option is equivalent to the `rotate-outer=false` setting and is the default.

4.2. Settings

These settings may be set with the `pie` option in `\DTLsetup`. For example:

```
\DTLsetup{pie={rotate-inner, rotate-outer}}
```


4.2.1. Pie Chart Data

variable= $\langle cs \rangle$

This specifies the control sequence used to construct the pie chart. The control sequence $\langle cs \rangle$ must be included in the assignment list provided in the final argument of `\DTLpiechart`. This setting is required by `\DTLpiechart`.

include-if= $\langle definition \rangle$

The value should be the definition (expansion text) for a command that takes a single argument. The definition should do `#1` for any row that should have its `variable` value included in the pie chart, and do nothing otherwise. See Example 116.

include-if-fn= $\langle cs \rangle$

An alternative to `include-if` where a function (identified by $\langle cs \rangle$) is provided instead of providing an inline definition.

The `include-if` and `include-if-fn` override each other, but both will be ignored if the $\langle condition \rangle$ optional argument is provided with `\DTLpiechart`. Either use one form or the other. Don't use both.

4.2.2. Pie Chart Style

These settings determine the pie chart's radius, outline and segment colours. See examples 117, 118 & 119.

cutaway= $\{ \langle list \rangle \}$

initial: empty

A list of cutaway segments. Each listed segment will be offset from the centre of the pie chart (the cutaway offset). The value should be a comma-separated list of individual numbers or number ranges (separated by a hyphen). The numbers refer to the segment index, starting from 1. For example, `cutaway={1, 3}` will separated the first and third segments, whereas `cutaway={1-3}` will separate the first three segments. An empty list, indicates no cutaway segments.

cutawayratio= $\langle value \rangle$

initial: 0.2

The cutaway offset is calculated as the given $\langle value \rangle$ multiplied by the radius. Alternatively, use `cutawayoffset` to set the offset explicitly.

cutaway-ratio=*<value>*

alias: **cutawayratio**

A synonym of `cutawayratio`.

cutawayoffset=*<dimension>*

The cutaway offset (not dependent on the radius). This is the distance from the centre of the pie chart to the point of the cutaway segments. The default value is obtained from the `cutaway-ratio` multiplied by the `radius`. Note that this option overrides `cutawayratio` but will itself be overridden by `radius`. If you also need to change the radius, either set `radius` first or use `radius*`.

cutaway-offset=*<dimension>*

alias: **cutawayoffset**

A synonym of `cutawayoffset`.

outline-width=*<dimension>*

initial: **0pt**

Sets the width of the line drawn around the segments. The outline will only be drawn if the given dimension is greater than 0pt.

outline-color=*<colour>*

initial: **black**

Sets the colour to draw the segment outlines. (To omit the outline, set the outline width to 0pt.) The given value should be a valid colour suitable for use in the mandatory argument of `\color`.

radius=*<dimension>*

initial: **2cm**

Sets the radius of the pie chart. This will also set the inner, outer and cutaway offsets according to the current `innerratio`, `outerratio` and `cutawayratio`.

radius*=*<dimension>*

Sets the radius of the pie chart without altering the offsets.

segment-colors=*{ <list> }*

Sets the segment colours. The supplied value should be a comma-separated list of colours suitable for use in the mandatory argument of `\color`. Note that if any segment colours have previously been assigned and this list is smaller, this will only override the given subset.

For example,

```
\DTLsetup{pie={
  segment-colors={pink, red, orange, green},
  segment-colors={cyan, magenta},
}}
```

This will assign cyan and magenta to the first two segments, but the third segment will be orange and the fourth green. Given the default settings, the fifth segment will also be magenta, the sixth segment will also be cyan, the seventh segment will be orange and the eighth segment will be white. Alternatively, you can use `\DTLsetpiesegmentcolor` to set the colour of a specific segment.

segment-default-colors

Resets the first eight segment colours to the default colour set: red, green, blue, yellow, magenta, cyan, orange, white. Any additional segment colours will be unchanged.

segment-default-gray

Resets the first eight segment colours to the default set of eight shades of grey. Any additional segment colours will be unchanged.

start=*<value>*

initial: 0

Sets the starting angle of the first segment (in degrees).

4.2.3. Pie Chart Labels

The inner labels are drawn inside each segment, offset from the segment point by the *inner offset*. The outer labels are drawn outside each segment, offset by the *outer offset*. The labels will typically include the placeholder commands, which will change with each segment. If the placeholder commands aren't used, the labels will be the same for every segment.

The default behaviour is to show the variable value in the inner label and nothing in the outer label. See examples 120 & 121.

innerlabel=*<value>*

initial: `\DTLpievariable`

The inner label, which should be displayed inside each segment, offset from the segment point by the inner offset. The default inner label is obtained from the `\DTLpievariable` placeholder, which shows the actual value. You can change this to `\DTLpiepercent` to show the percentage value instead, as in Example 120.

inner-label= $\langle value \rangle$

alias: **innerlabel**

A synonym of `innerlabel`.

innerratio= $\langle value \rangle$

initial: 0.5

The inner offset is calculated as the given $\langle value \rangle$ multiplied by the radius. Alternatively, use `inneroffset` to set the offset explicitly.

inner-ratio= $\langle value \rangle$

alias: **innerratio**

A synonym of `innerratio`.

inneroffset= $\langle dimension \rangle$

The inner offset (not dependent on the radius). This is the distance from the centre of the pie chart (before the cutaway shift) to the point where the inner label is placed. Note that this option overrides `innerratio` but will itself be overridden by `radius`. If you also need to change the radius, either set `radius` first or use `radius*`.

inner-offset= $\langle dimension \rangle$

alias: **inneroffset**

A synonym of `inneroffset`.

outerlabel= $\langle value \rangle$

initial: empty

The outer label, which should be displayed inside each segment, offset from the segment point by the outer offset.

outer-label= $\langle value \rangle$

alias: **outerlabel**

A synonym of `outerlabel`.

outeroffset= $\langle dimension \rangle$

The outer offset (not dependent on the radius). This is the distance from the centre of the pie chart (before the cutaway shift) to the point where the outer label is placed. Note that this option overrides `outerratio` but will itself be overridden by `radius`. If you also need to change the radius, either set `radius` first or use `radius*`.

outer-offset= $\langle dimension \rangle$ *alias:* **outeroffset**A synonym of `outeroffset`.**rotateinner**= $\langle boolean \rangle$ *default:* **true**; *initial:* **false**

If true, this indicates that inner labels should be rotated. Otherwise the inner labels will be horizontal.

rotate-inner= $\langle boolean \rangle$
initial: **false***alias:* **rotateinner**; *default:* **true**;A synonym of `rotateinner`.**rotateouter**= $\langle boolean \rangle$ *default:* **true**; *initial:* **false**

If true, this indicates that outer labels should be rotated. Otherwise the outer labels will be horizontal.

rotate-outer= $\langle boolean \rangle$
initial: **false***alias:* **rotateouter**; *default:* **true**;A synonym of `rotateouter`.**round**= $\langle n \rangle$ *initial:* **1**Sets the number of digits used to round the percentage value in `\DTLpiepercent`.**outerratio**= $\langle value \rangle$ *initial:* **1.25**The outer offset is calculated as the given $\langle value \rangle$ multiplied by the radius. Alternatively, use `outeroffset` to set the offset explicitly.**outer-ratio**= $\langle value \rangle$ *alias:* **outerratio**A synonym of `outerratio`.

4.3. Pie Chart Examples

These examples use the “fruit” database (see §3.2.7).

4.3.1. Pie Chart Filtering Examples

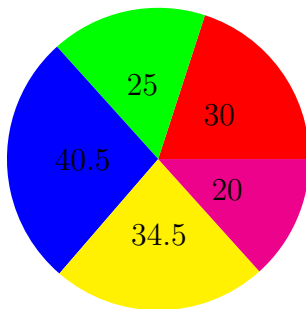
Example 116 modifies Example 114 to skip the “Pears” row using the `include-if` setting:

116

```
\DTLpiechart
{
  variable=\Quantity,% variable required
  include-if={\DTLifstringeq{\Name}{Pears}}{\#1}}
{fruit}% database
{\Quantity=Quantity,\Name=Name}% assignment list
```

This uses `\DTLifstringeq`, which expands its arguments, to test value of the placeholder command. Note that while you can use `etoolbox`’s `\ifdefstring` with the default `store-datum=false`, it won’t work with `store-datum=true` as then `\Name` will be a datum control sequence.

↑ Example 116: Pie Chart (Filtering)



The same result can also be achieved with the optional argument, but bear in mind that you can’t use both the optional argument and the `include-if` setting. The optional argument, if set, will always override the `include-if`/`include-if-fn` setting.

```
\DTLpiechart
[\equal{\Name}{Pears}]
{
  variable=\Quantity% variable required
}
{fruit}% database
```

```
{\Quantity=Quantity,\Name=Name}% assignment list
```

You can provide your own custom command for use with the `include-if` or `include-if-fn` settings. For example:

```
\ExplSyntaxOn
\NewDocumentCommand \PieChartNameFilter { m m }
{
  \datatool_if_value_eq:NnF \Name { #1 } { #2 }
}
\ExplSyntaxOff

\DTLpiechart
{
  variable=\Quantity,% variable required
  include-if={\PieChartNameFilter{Pears}{#1}}
}
{fruit}% database
{\Quantity=Quantity,\Name=Name}% assignment list
```

Alternatively:

```
\ExplSyntaxOn
\NewDocumentCommand \PearFilter { m }
{
  \datatool_if_value_eq:NnF \Name { Pears } { #1 }
}
\ExplSyntaxOff

\DTLpiechart
{
  variable=\Quantity,% variable required
  include-if-fn={\PearFilter}
}
{fruit}% database
{\Quantity=Quantity,\Name=Name}% assignment list
```

4.3.2. Pie Chart Styles Examples

Example 117 adapts Example 114 so that the first and third segments are offset from the centre

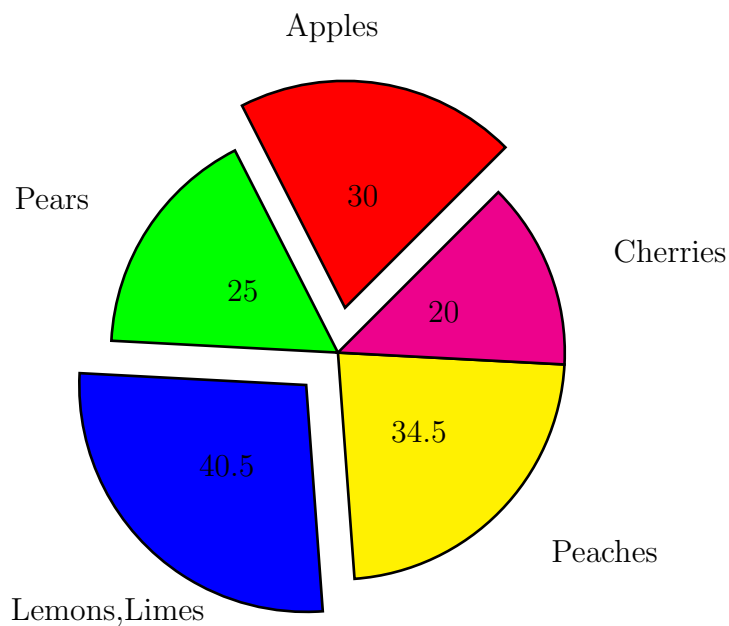
117

4. Pie Charts (datapie package)

of the pie chart. The starting angle has also been changed to 45 degrees and the radius to 3cm. Each segment is outlined.

```
\DTLpiechart
{
  variable=\Quantity,
  outer-label=\Name,
  cutaway={1,3},
  start=45,
  radius=3cm,
  outline-width=1pt
}
{fruit}% database
{\Quantity=Quantity,\Name=Name}% assignment list
```

↑ Example 117: Separating Segments from a Pie Chart



Note the difference between Example 118 which has a range:

```
\DTLpiechart
{
```

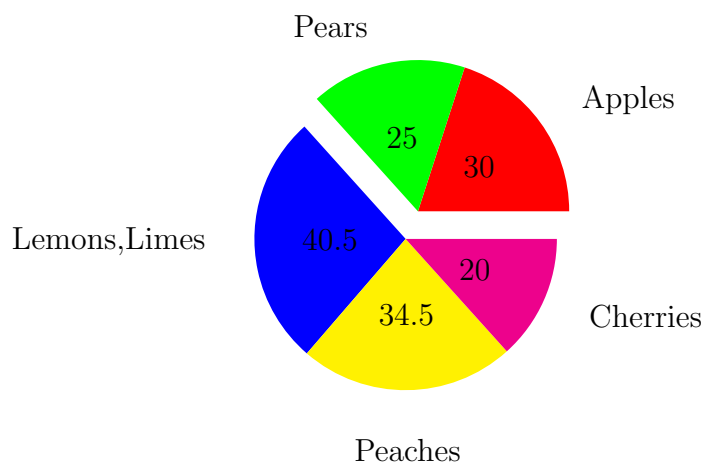

4. Pie Charts (datapie package)

```
variable=\Quantity,  
outer-label=\Name,  
cutaway={1-2},  
}  
{fruit}{\Quantity=Quantity,\Name=Name}
```

and Example 119 which has separate consecutive segments:

```
\DTLpiechart  
{  
  variable=\Quantity,  
  outer-label=\Name,  
  cutaway={1,2}  
}  
{fruit}{\Quantity=Quantity,\Name=Name}
```

Example 118: Separating a Range of Segments from a Pie Chart



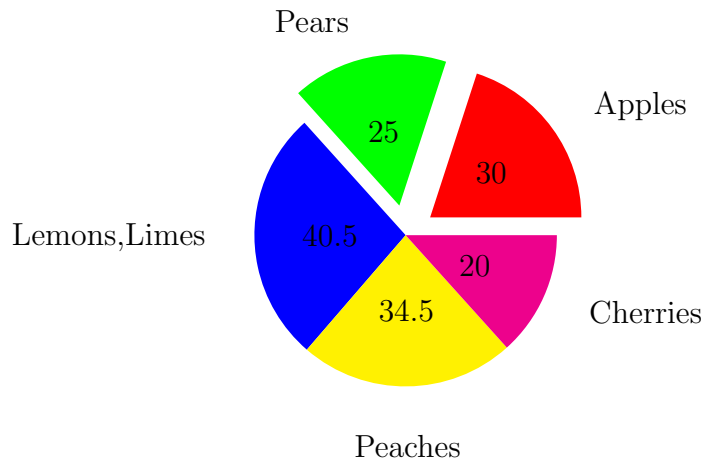
4.3.3. Pie Chart Labels Examples

The default behaviour is to show the variable value in the inner label and nothing in the outer label. Example 120 changes the inner label to the percentage and the outer label to the corresponding value of the “Name” column. This is achieved with a slight modification to Example 114:

120

4. Pie Charts (datapie package)

Example 119: Separating Individual Consecutive Segments from a Pie Chart



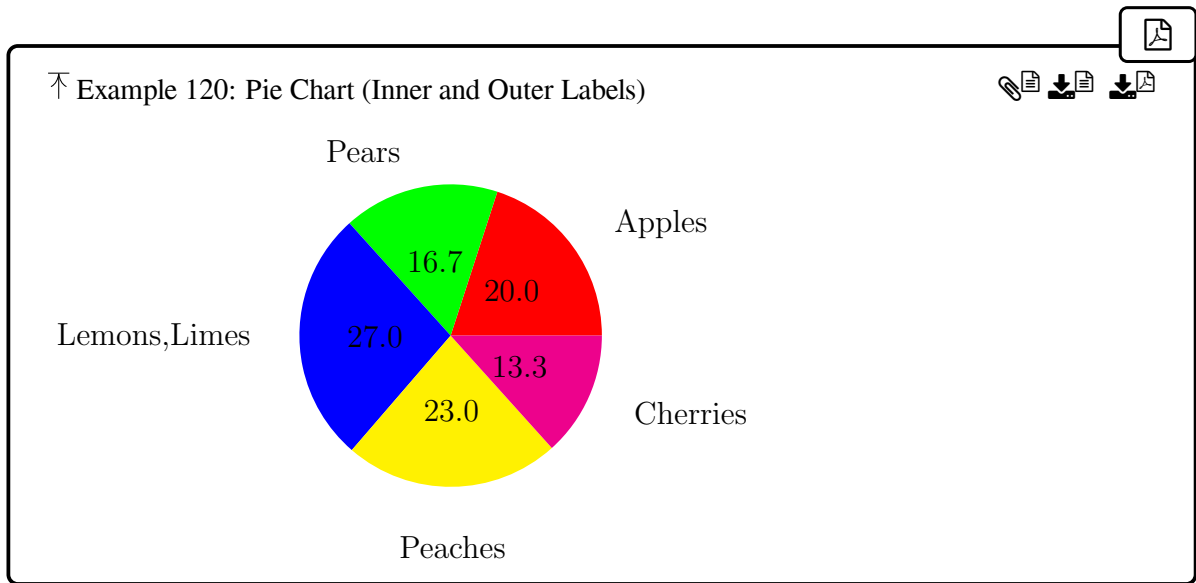
```
\DTLpiechart
{
  variable=\Quantity,
  inner-label=\DTLpiepercent,
  outer-label=\Name
}
{fruit}% database
{\Quantity=Quantity,\Name=Name}
```

Note that this has added an extra assignment in the final argument to provide a placeholder command for the corresponding item in the “Name” column. See also Example 122 which includes the percentage symbol in the inner label and appends the actual value to the outer label.

Example 121 adapts Example 120 to rotate the inner and outer labels:

```
\DTLpiechart
{
  variable=\Quantity,
  rotate-inner,
  rotate-outer,
  inner-label=\DTLpiepercent,
  outer-label=\Name
}
{fruit}% database
```

121



```
{\Quantity=Quantity,\Name=Name}
```

4.3.4. Pie Chart Placeholder Example

Example 122 adapts Example 120 to append the actual value in the outer label. The inner label is adjusted to include the percent symbol and the percentage is rounded to the nearest whole number.

122

```
\DTLpiechart
{
  variable=\Quantity,
  round=0,
  inner-label=\DTLpiepercent\%,
  outer-label=\Name\ (\DTLpievariable)
}
{fruit}% database
{\Quantity=Quantity,\Name=Name}
```

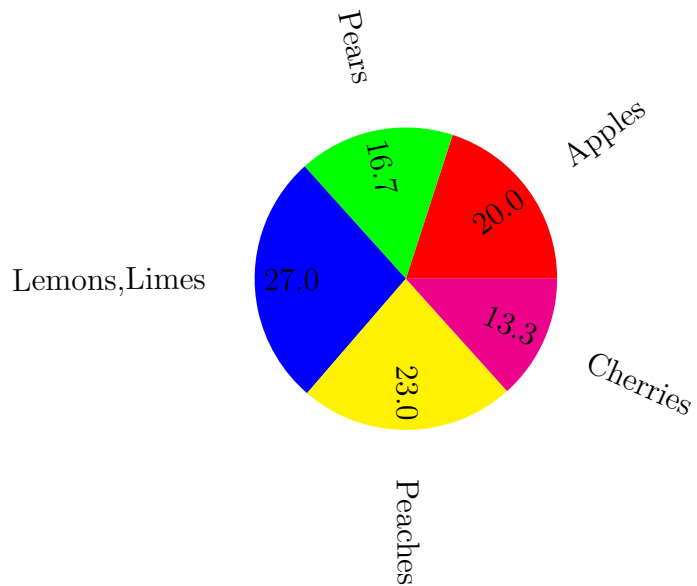
4.3.5. Pie Chart Label Formatting Example

Example 123 adapts Example 122 to format the outer labels in sans-serif and the inner labels in a pale grey bold font:

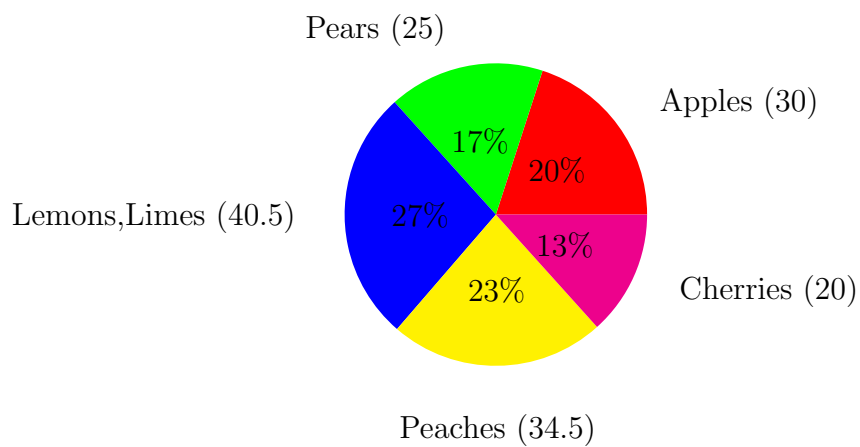
123

4. Pie Charts (datapie package)

Example 121: Pie Chart (Labels Rotated)



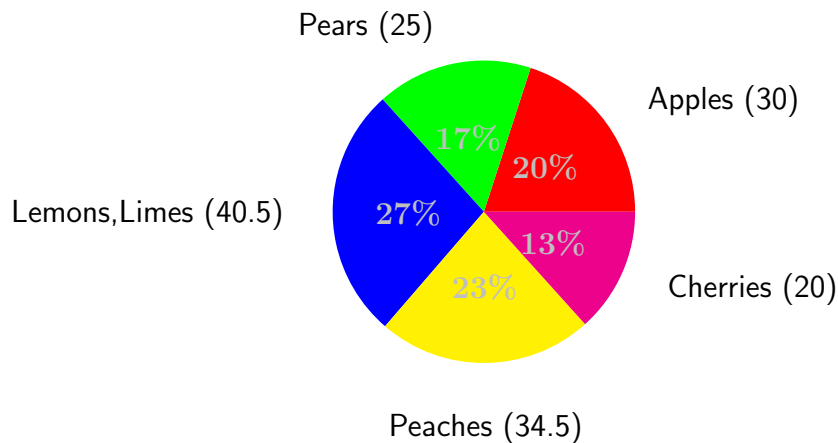
Example 122: Pie Chart (Percentage Rounding)



4. Pie Charts (datapie package)

```
\renewcommand{\DTLdisplayinnerlabel}[1]{%  
  \textcolor{lightgray}{\bfseries #1}}%  
}  
\renewcommand{\DTLdisplayouterlabel}[1]{\textsf{#1}}
```

↑ Example 123: Pie Chart (Changing the Label Format)



4.3.6. Pie Chart Colour Example

Example 124 adapts Example 123. The third and fourth segment colours are set to yellow and pink:

124

```
\DTLsetpiesegmentcolor{3}{yellow}  
\DTLsetpiesegmentcolor{4}{pink}
```

The outline width is set to 2pt. This can either be done by explicitly changing `\DTLpieoutlinewidth`:

```
\setlength{\DTLpieoutlinewidth}{2pt}
```

or implicitly via the `outline-width` option:

```
\DTLsetup{pie={outline-width=2pt}}
```

4. Pie Charts (*datapie* package)

Alternatively, `outline-width` may be set in the settings argument of `\DTLpiechart` if it's only applicable for that chart.

The outer label format is changed to match the text to the segment colour:

```
\renewcommand{\DTLdisplayouterlabel}[1]{%
  \DTLdocurrentpiesegmentcolor
  \textsf{\shortstack{#1}}}
```

I've used `\shortstack` to compact the outer label by putting the value below the name:

```
outer-label=\Name\\(\DTLpievariable)
```

After the pie chart, a key is created with the tabular environment. Note the limitations of `\DTLmapdata` within a tabular context, so `\DTLforeach` is used instead:

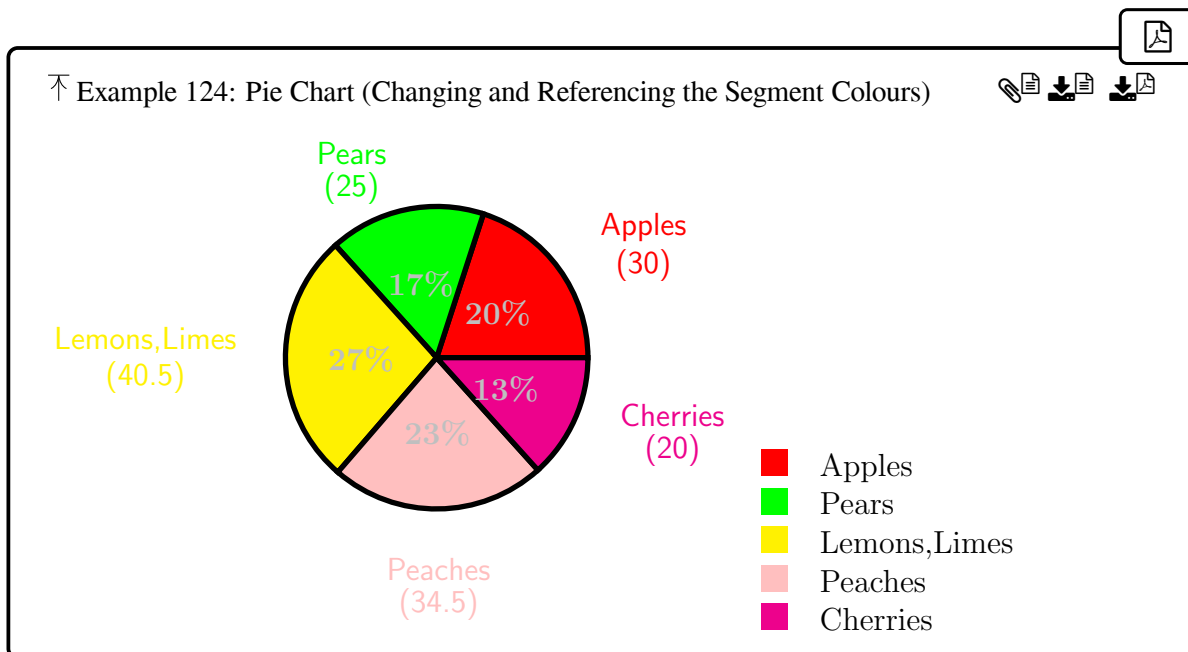
```
\begin{tabular}[b]{l}
\DTLforeach{fruit}{\Name=Name}{%
  \DTLiffirstrow}{\\}%
  \DTLdocurrentpiesegmentcolor
  \rule{10pt}{10pt} &
  \Name
}
\end{tabular}
```

4.4. Pie Chart Variables

```
\DTLpievariable
```

The expansion text of this command should be the control sequence used to identify the column of data used to compute the size of each segment. The `variable=<cs>` option simply defines `\DTLpievariable` to `<cs>`. The `\DTLpievariable` command is simply a convenient wrapper that allows the definition of the inner or outer label to reference the segment value without needing to know the assignment list used in the final argument of `\DTLpiechart`, and also identifies the required column to use for the numeric value if multiple fields are assignment.

```
\DTLpiepercent
```



When each segment is drawn, this command is defined to the percentage value of the variable, rounded to the number of digits given by the following counter:

`\DTLpiepercent`

No

This placeholder command may then be used in the inner or outer label to show the percentage value instead of the actual numeric value. See Example 122.

`\DTLpiepercent` will expand to a plain number without a trailing percent symbol.

4.5. Pie Chart Label Formatting

`\DTLdisplayinnerlabel{<text>}`

This governs how the inner label is formatted, where `<text>` is the inner label text. The default definition simply expands to `<text>`.

`\DTLdisplayouterlabel{<text>}`

This governs how the outer label is formatted, where `<text>` is the outer label text. The default definition simply expands to `<text>`. See Example 123.

4.6. Pie Chart Colours

The *datapie* package predefines colours for the first eight segments of the pie chart. If you require more than eight segments or if you want to change the default colours, you can either use the `segment-colors` option to supply a list or use:

```
\DTLsetpiesegmentcolor{<n>}{<colour>}
```

This will set the colour to the $\langle n \rangle$ th segment (starting from 1 for the first segment). The second argument should be a valid colour that can be used in the mandatory argument of `\color`. See Example 124.

It's useful to set the colours so that each segment colour is somehow relevant to whatever the segment represents. For example, in the previous example pie charts depicting quantities of fruit, some of the default colours were inappropriate. Whilst red is reasonable for apples and green for pears, blue doesn't really correspond to lemons or limes.

With the default `color` package option, the first eight segments are initialised to the colours: red, green, blue, yellow, magenta, cyan, orange, and white. With the `gray` package option, the first eight segments are initialised to eight different shades of grey. In both cases, only the first eight segments have an associated colour. Bear in mind that the `gray` package option simply sets the first eight segments. It doesn't enforce greyscale for any subsequent segment colour changes.

For example:

```
\usepackage[gray]{datapie}
\DTLsetup{pie={segment-colors={pink,,green}}
\DTLsetpiesegmentcolor{6}{purple}
\DTLsetpiesegmentcolor{9}{teal}
```

This starts by initialising the first eight segment colours to shades of grey, then the first two segment colours are changed to pink and green (empty items are skipped when the CSV list is parsed), then the sixth segment colour is changed to purple, and the ninth segment colour (which hasn't yet been set) is set to teal. Segments 3, 4, 5, 7, and 8 are still shades of grey.

```
\DTLdopiesegmentcolor{<n>}
```

This sets the current colour (with `\color`) to that of the $\langle n \rangle$ th segment. Issues a warning and does nothing if no colour has been set for that segment.

```
\DTLgetpiesegmentcolor{<n>}
```


Expands to the colour for the $\langle n \rangle$ th segment or to `white` if not set. (This ensures that no error occurs if used in an expandable context when no colour has been assigned to the given segment.) Since the page colour is typically white, this will give the appearance of an unfilled segment but bear in mind that it will obscure anything drawn in the same position first (for example, by the `\DTLpieatbegintikz` hook).

`\DTLdocurrentpiesegmentcolor`

This command is intended for use within `\DTLpiechart` as a shortcut that will set the current text colour to that of the current segment. However, it may also be used within `\DTLmapdata` or `\DTLforeach` and will use the current row index as the segment index but, to avoid ambiguity, use `\DTLdocurrentpiesegmentcolor` in the inner or outer labels and `\DTLdopiesegmentcolor` elsewhere.

`\DTLpieoutlinecolor`

initial: black

This command should expand to the colour specification for the segment outline. The `outline-color` setting simply redefines `\DTLpieoutlinecolor` to the supplied value.

`\DTLpieoutlinewidth`

initial: 0pt

This length register stores the line width of the segment outline. The `outline-width` setting sets this register to the supplied value. The outline is only draw if the dimension is greater than 0pt.

4.7. Pie Chart Hooks

`\DTLpieatsegment { $\langle index \rangle$ } { $\langle total \rangle$ } { $\langle start\ angle \rangle$ } { $\langle mid\ angle \rangle$ } { $\langle end\ angle \rangle$ } { $\langle shift\ angle \rangle$ } { $\langle shift\ offset \rangle$ } { $\langle inner\ offset \rangle$ } { $\langle outer\ offset \rangle$ }`

This hook is used after each segment and its associated inner and outer labels are drawn, before the end of the current scope (which applies the shift transformation for cutaway segments). Note that the shift means that (0,0) corresponds to the segment point, which won't be the centre of the pie chart for cutaway segments.

The arguments are as follows:

- $\langle index \rangle$: the segment index, which may be used to reference the segment colour with `\DTLdopiesegmentcolor`.
- $\langle total \rangle$: the decimal (plain number) total for the column used as the pie chart variable (omitting any rows skipped by the filter condition).

4. Pie Charts (*datapie* package)

- $\langle start\ angle \rangle$: the starting angle of the segment.
- $\langle mid\ angle \rangle$: the mid way angle.
- $\langle end\ angle \rangle$: the end angle of the segment.
- $\langle shift\ angle \rangle$: the angle part of the segment offset polar co-ordinate.
- $\langle shift\ offset \rangle$: the radius part of the segment offset polar co-ordinate.
- $\langle inner\ offset \rangle$: the offset to the inner label.
- $\langle outer\ offset \rangle$: the offset to the outer label.

`\DTLpieatbegintikz`

This hook is used at the start of the `tikzpicture` environment, allowing you to change the `tikzpicture` settings. Does nothing by default. For example, this hook may be redefined to scale the pie chart. This hook may also be used to insert additional content but note that it may be obscured by the chart if it overlaps.

`\DTLpieatendtikz`

This hook is used at the end of the `tikzpicture` environment, allowing you to add additional content, which may overlap the chart. Does nothing by default.

5. Bar Charts (databar package)

```
\usepackage[<options>]{databar}
```

The `databar` package can be used to draw bar charts from data obtain from a database. This package automatically loads the `dataplot` package as it shares some functions (such as the default tick point generation).

Any package options provided when loading `databar` will be passed to `datatool`. If `datatool` has already been loaded, any options will be passed to `\DTLsetup` instead (which means that you would only be able to use options that can be set after `datatool` has been loaded). Note that the `dataplot` package will then be loaded afterwards without passing any options to it.

The `databar` package was rewritten in version 3.0 to use $\text{\LaTeX}3$ commands. The `xkeyval` package has been dropped and the use of `\DTLforeach` has been replaced with `\DTLmapdata`. A number of bugs have also been fixed, which may cause some differences in the output.

Rollback to version 2.32 is available:

```
\usepackage{databar}[=2.32]
```

Note that if `datatool` hasn't already been loaded, this will also apply rollback to `datatool` and `dataplot`. Problems may occur if a newer release of `datatool` or `dataplot` has already been loaded.

In this chapter, the terms “upper”, “lower”, “width”, “height”, “x” and “y” are relative to the chart's orientation. With the default `verticalbars=true` setting, the x axis is horizontal (vertical bars ordered from left to right) and the y axis is vertical (increasing values from bottom to top). With `verticalbars=false`, the x axis is vertical (horizontal bars ordered from bottom to top) and the y axis is horizontal (increasing values from left to right).

The “width” is the fixed distance of the bar along the x axis (`barwidth`) which is equal to 1 x unit, and the “height” is the variable length along the y axis (indicating the value). “Lower” is the $y = 0$ end of the bar, and “upper” is the other end of the bar, which

5. Bar Charts (*databar* package)

will be above (vertical) or to the right (horizontal) for positive values or below (vertical) or to the left (horizontal) for negative values.

The *databar* package provides two commands to draw bar charts.

`\DTLbarchart [<condition>] {<settings-list>} {<db-name>} {<assign-list>}`

Draws a bar chart from the data given in the database identified by *<db-name>*. The *<db-name>* argument may be empty to indicate the default database. The *<settings list>* argument may be a *<key>=<value>* list of options to override the default. See §5.2 for available settings and Example 125 for a simple example.

The `variable` setting must be provided with `\DTLbarchart` to identify the column of data to plot. This should be set to the applicable placeholder command in *<assign-list>*.

`\DTLmultibarchart [<condition>] {<settings-list>} {<db-name>} {<assign-list>}`

This command is similar to `\DTLbarchart` but plots groups of bars, where each bar in the group has its numeric value obtained from the corresponding placeholder command in the `variables` list. See Example 128 for a simple example.

The `variables` setting must be provided with `\DTLmultibarchart` to identify the columns of data to plot. This should be set to a comma-separated list of the applicable placeholder commands in *<assign-list>*.

For both `\DTLbarchart` and `\DTLmultibarchart`, the *<assign-list>* argument should be a *<cs>=<key>* list of placeholder command assignments suitable for use in `\DTLmap-getvalues`. Additional assignments may be added if required for the bar lower or upper labels.

The *<condition>* optional argument allows filtering to be applied. This should be in the syntax allowed for the first argument of `\ifthenelse` (see §2.4.2). Note that if this option is provided, it will override the `include-if/include-if-fn` setting.

There are also corresponding actions. Both have optional settings: `name` (for the database name), `assign` (for the assignment list corresponding to the *<assign-list>* argument of `\DTLbarchart` and `\DTLmultibarchart`), and `options` (for the bar chart settings corresponding to the *<settings-list>* argument).

```
\DTLaction[⟨settings⟩]{bar chart}
```

The `bar chart` action is equivalent to using `\DTLbarchart`. If you include the `key` or `column` action option to identify the column to use for the y values then you can omit variable in the `options` setting. See Example 126 for a simple example.

If variable is included in `options`, it will override the action `key` or `column` setting. However, if variable has been previous set within the `bar` option in `\DTL-`setup and does not occur in the action `options` then the action `key` or `column` setting will be used (if provided).

If you identify the required column of data with `key` or `column` then a temporary placeholder command will be created and added to the assignment list. If you don't need to use any of the database values in hooks or options (such as the bar labels) then you may omit the `assign` action option or just assign the placeholders required for the labels. You will still be able to use `\DTLbarvariable` or `\DTLbarvalue`.

```
\DTLaction[⟨settings⟩]{multibar chart}
```

The `multibar chart` action is equivalent to using `\DTLmultibarchart`. You can either set the list of bar chart variable placeholder commands in the `variables` option within the action `options` setting or you can use the action settings `keys` or `columns` (or a combination of both) to identify the required columns. See Example 129 for a simple example.

As with the `bar chart` action, if you select the columns using the action settings rather than the `variables` option, you may omit the corresponding placeholder commands in the `assign` list.

For example, the “fruit” database (see §3.2.7) has two columns identified by the labels `Name` and `Quantity`. The `Quantity` column is numeric and so can be used to create a bar chart.

Example 125 creates a simple bar chart using the fruit database:

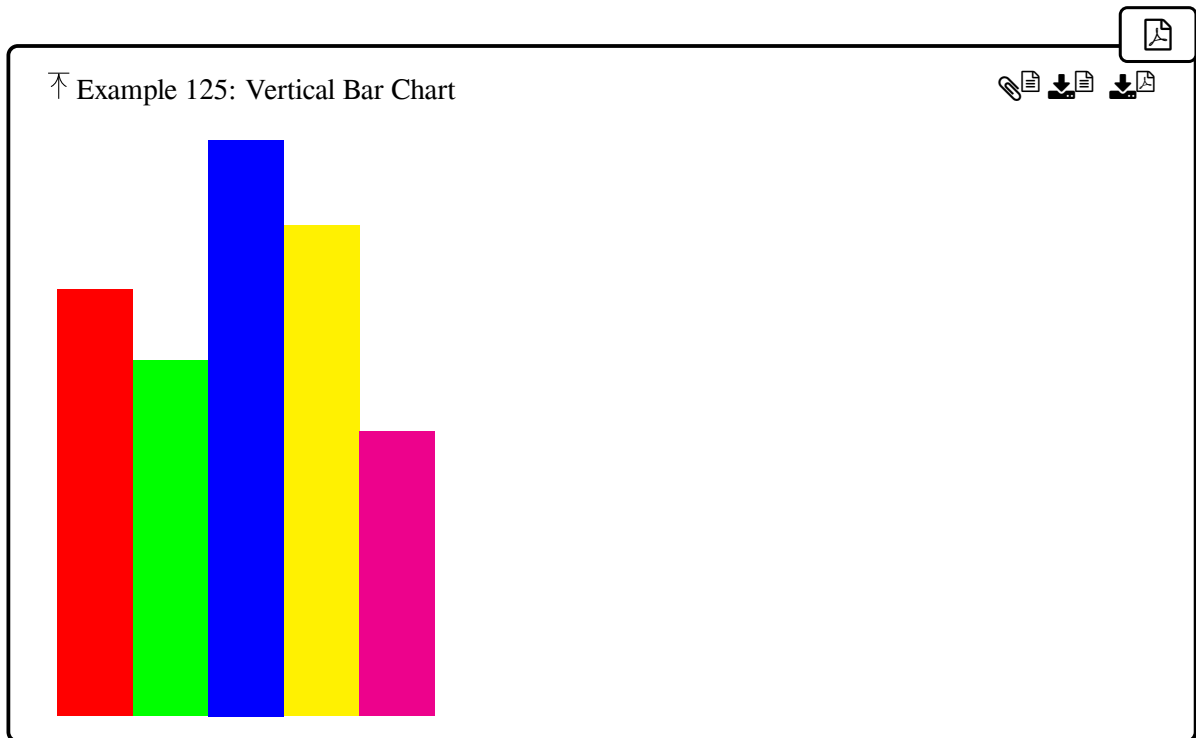
```
\DTLbarchart
{variable=\Quantity}% variable required
{fruit}% database name
{\Quantity=Quantity}% assignment list
```

Note that this has the same syntax as `\DTLpiechart` (see Example 114) which makes it simple to swap between a bar and pie chart if you change your mind about the chart type.

Example 126 is the same as Example 125 but uses the `bar chart` action. Note that this doesn't need to reference the database name.

125

126



```
% Load data from fruit.csv file:
\DTLsetup{default-name=fruit}
\DTLread{fruit.csv}
\DTLaction[key=Quantity]{bar chart}
```

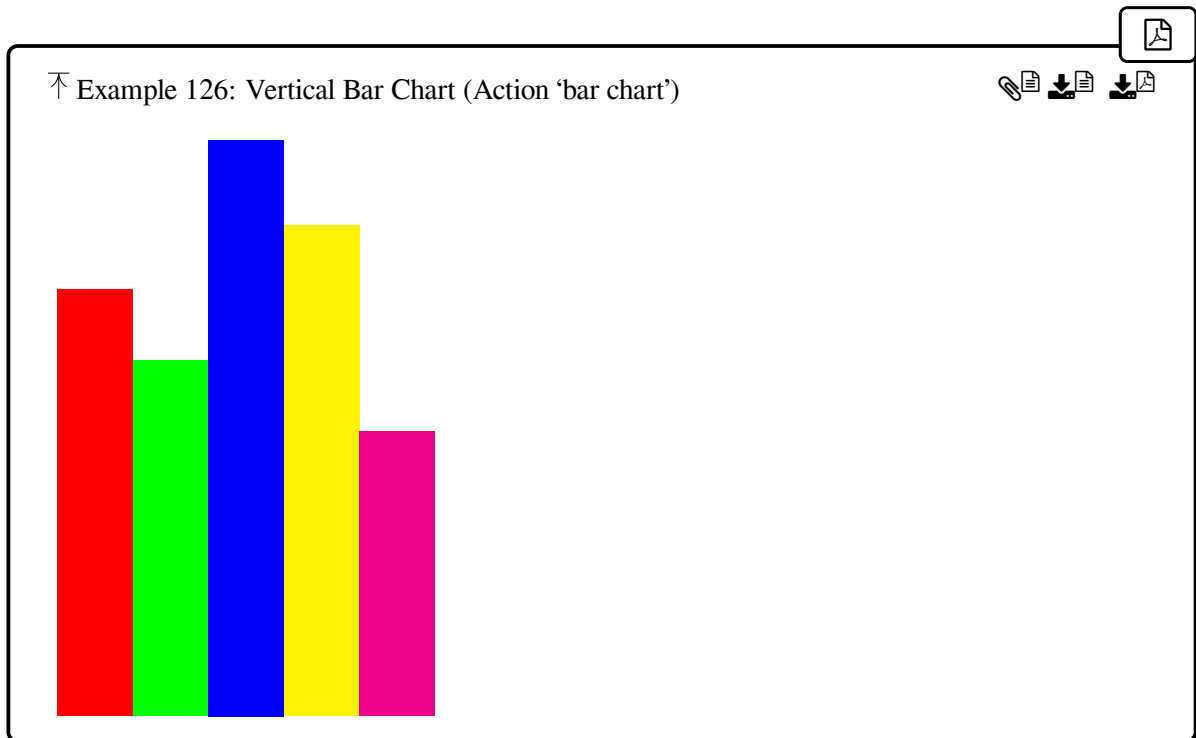
The “profits” database (see §3.2.8) has columns identified by the labels `Year` and `Profit`. The `Profit` column has some negative values, which will result in downward bars, with the default vertical setting, or leftward bars with the horizontal setting.

Example 127 creates a bar chart which has horizontal bars. Note that the bar width now refers to the vertical length.

127

```
\DTLbarchart
{
  variable=\theProfit,
  horizontal,
  bar-width=20pt
}
{profits}% database name
{% assignment list
\theYear=Year,
```

5. Bar Charts (*databar package*)



```
\theProfit=Profit  
}
```

(See Example 143 to make all positive bars blue and all negative bars red.)

The “marks” database (see §3.2.1) has data containing marks for three assignments given in columns labelled Assign1, Assign2 and Assign3. Example 128 displays this data as a multi bar chart with three-bar groups for each student:

128

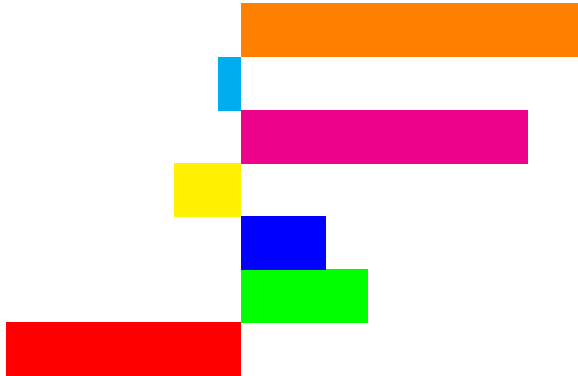
```
\DTLmultibarchart  
{  
  variables={\assignI,\assignII,\assignIII},  
  barwidth=10pt  
}  
{marks}% database name  
{  
  \assignI=Assign1,  
  \assignII=Assign2,  
  \assignIII=Assign3  
}
```

Example 129 is the same as Example 128 but uses the `multibar chart` action.

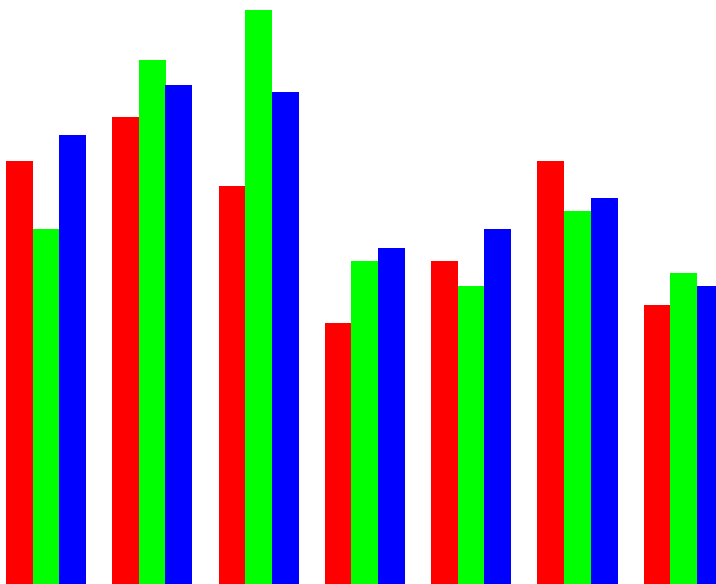
129

5. Bar Charts (*databar* package)

↑ Example 127: Horizontal Bar Chart



↑ Example 128: Multi Bar Chart



5. Bar Charts (*databar package*)

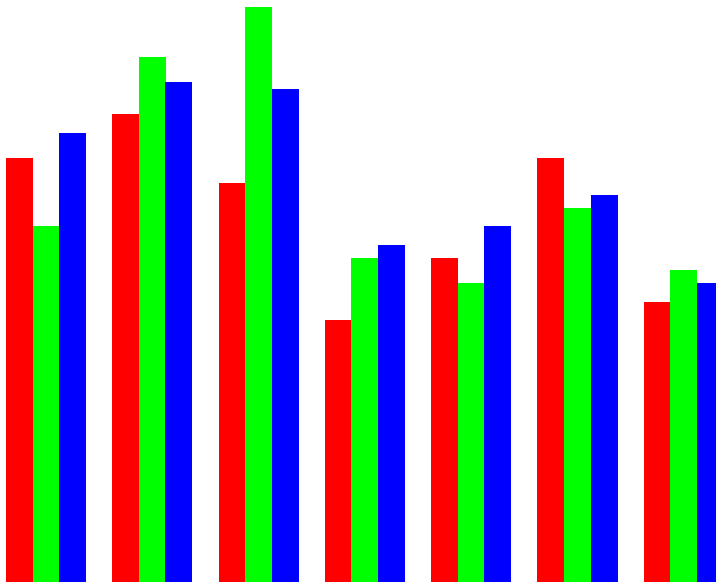
```
\DTLaction
[
  keys={Assign1,Assign2,Assign3},
  options={
    barwidth=10pt
  }
]
{multibar chart}
```

Alternatively, rather than listing every column key, a column range may be used:

```
\DTLaction
[
  columns={4-}, % column 4 onwards
  options={
    barwidth=10pt
  }
]
{multibar chart}
```

This identifies column 4 onwards.

↑ Example 129: Multi Bar Chart (Action 'multibar chart')



5.1. Package Options

The options listed here may be passed as package options when loading `databar`. Unless otherwise stated, these options can't be used in `\DTLsetup`. Additional options that may be set with `\DTLsetup` are listed in §5.2.

color

Initialises the default colour list to: red, green, blue, yellow, magenta, cyan, orange, white. This package option is equivalent to `bar-default-colors` and is the default.

gray

Initialises the default colour list to eight shades of grey. This package option is equivalent to `bar-default-gray`.

verticalbars=*<boolean>*

default: true; initial: true

If set to true, the bars will be vertical, otherwise they will be horizontal.

vertical

Equivalent to `verticalbars=true`.

horizontal

Equivalent to `verticalbars=false`.

5.2. Settings

These settings may be set with the `bar` option in `\DTLsetup`. For example:

```
\DTLsetup{bar={horizontal,round=0}}
```

Most of the settings can be used instead of redefining or setting associated commands (such as `\DTLbarXlabelalign`) or registers (such as `\DTLbarlabeloffset` or `DTLbar-roundvar`). For other commands, such as `\DTLbardisplayYticklabel`, you can put the redefinition code within `init={code}` to localise the effect.

init=*{code}*

The value should be code to be performed at the start of `\DTLbarchart` or `\DTLmulti-`

5. Bar Charts (*databar package*)

`barchart` after the `<settings>` argument has been parsed. This code will be scoped. Example 146 uses `init` to locally redefine `\DTLeverybarhook`.

```
pre-init={\code}
```

This is similar to `init` but is processed before all other options in the same set (see Example 141).

Unlike some options, such as `bar-colors`, the `init` and `pre-init` options are processed within the local scope of `\DTLbarchart` or `\DTLmultibarchart`. This means that if you set the options in `\DTLsetup` rather than in the applicable bar chart command, the effect of `pre-init` may occur too late.

For example, to set the colour list to exactly cyan, yellow, magenta for a particular bar chart (as in Example 141):

```
\DTLbarchart
{
  variable=\Quantity,
  bar-label=\Name,
  pre-init={\DTLclearbarcolors},
  outline-width=1pt,
  bar-colors={cyan,magenta,yellow}
}
{fruit}% database
{\Quantity=Quantity,\Name=Name}% assignment list
```

However, if the settings are moved into `\DTLsetup`:

```
\DTLsetup{
  bar={
    pre-init={\DTLclearbarcolors},
    outline-width=1pt,
    bar-colors={cyan,magenta,yellow}
  }
}
\DTLbarchart
{
  variable=\Quantity,
  bar-label=\Name
```

```

}
{fruit}% database
{\Quantity=Quantity,\Name=Name}% assignment list

```

This will result in white bars as the colour list is set in `\DTLsetup` but the `pre-init` code isn't done until `\DTLbarchart`, at which point it will clear the colour list. In this case, the solution is simply to put `\DTLclearbarcolors` before `\DTLsetup`:

```

\DTLclearbarcolors
\DTLsetup{
  bar={
    outline-width=1pt,
    bar-colors={cyan,magenta,yellow}
  }
}

```

5.2.1. Bar Chart Data

variable=*<cs>*

This specifies the placeholder command used to construct the bar chart with `\DTLbarchart`. The command *<cs>* must be included in the assignment list provided in the final argument of `\DTLbarchart`. This setting is required by `\DTLbarchart`.

variables=*<cs-list>*

This specifies a comma-separated list of placeholder commands used to construct the bar chart with `\DTLmultibarchart`. Each command in the list must be included in the assignment list provided in the final argument of `\DTLmultibarchart`. This setting is required by `\DTLmultibarchart`.

Certain rows in the database may be omitted from the bar chart either with the optional *<condition>* argument (of `\DTLbarchart` or `\DTLmultibarchart`) or by using one of the filtering options below.

If `\dtlrownum` is used in `\DTLeverybarhook` or `\DTLeverybargroup-hook` it will be set to the current database row index, which doesn't take the filtering into account (see Example 146).

include-if=*<definition>*

The value should be the definition (expansion text) for a command that takes a single argument. The command should do #1 for any row that should be included in the bar chart, and do nothing otherwise. Example 132 demonstrates filtering with the `include-if` setting.

include-if-fn=*<cs>*

An alternative to `include-if` where a function (identified by *<cs>*) is provided instead of providing an inline definition.

The `include-if` and `include-if-fn` override each other, but both will be ignored if the *<condition>* optional argument is provided with `\DTLbarchart` or `\DTLmultibarchart`. Either use one form or the other. Don't use both.

5.2.2. Bar Chart Style

These settings determine the bar chart's orientation, size, outline and bar colours. Examples in §§5.3.3, 5.3.4 & 5.3.7 demonstrate the chart upper and lower labels.

verticalbars=*<boolean>*

default: true; initial: true

If set to true, the bars will be vertical, otherwise they will be horizontal.

vertical

Equivalent to `verticalbars=true`. This is the default setting (see Example 125).

horizontal

Equivalent to `verticalbars=false`. Example 127 uses this setting to create a chart with horizontal bars.

lower-label-style=*<value>*

initial: opposite

Specifies the positioning and alignment of the lower bar label. This option will redefine `\DTLbarXlabelalign` and `\DTLbarXneglabelalign`.

lower-label-style=opposite

Lower labels will be on the opposite side of the x -axis to the bar (that is, above the axis for negative values and below the axis for positive values). Example 133 illustrates this style.

lower-label-style=same

Lower labels will be on the same side of the x -axis to the bar (that is, above the axis for positive values and below the axis for negative values). This means that the labels will appear inside the bar. Note that this will result in protrusion over the far end of the bar if the label text is longer than the length of the bar, which can collide with the upper label, if present. Example 134 illustrates this style.

lower-label-style=above

Lower labels will be above the x -axis regardless of the bar value. Again, this may result in protrusion over the far end of the bar, where the bar is also above the axis. Example 136 illustrates this style.

lower-label-style=below

Lower labels will be below the x -axis regardless of the bar value. Again, this may result in protrusion over the far end of the bar, where the bar is also below the axis. Example 135 illustrates this style.

upper-label-align= [$\langle -ve align \rangle$] { $\langle +ve align \rangle$ }

Sets the alignment for the bar upper labels where $\langle +ve align \rangle$ is the alignment for bars with positive values and $\langle -ve align \rangle$, if provided, is the alignment for bars with negative values. The default is:

```
upper-label-align=
  [\ifDTLverticalbars bottom,center\else left\fi]
  {\ifDTLverticalbars top,center\else right\fi}
```

If $\langle -ve align \rangle$ is omitted, only the alignment for bars with positive values will be changed. The grouping around $\langle +ve align \rangle$ is optional. For example, `upper-label-align=[left]right` is equivalent to `upper-label-align=[left]{right}`. See §5.3.4 for an example.

length= $\langle dim \rangle$ *initial: 3in*

The length (as a dimension) of the bar chart. (That is, the total length of the y axis.) The length of the x -axis is governed by the number of bars, the bar width, and (for multi bar charts) the group gap.

barwidth= $\langle dim \rangle$ *alias: bar-width; initial: 1cm*

Sets the width (as a dimension) of each bar.

bar-width= $\langle dim \rangle$ *initial: 1cm*

A synonym of `barwidth`.

bargap= $\langle value \rangle$ *alias: bar-gap; initial: 0*

The gap between bars in terms of multiples of a bar width. (For example, a value of 1 indicates a gap of one bar whereas a value of 0.5 indicates a gap of half a bar.) Note that with `\DTLmultibarchart` this gap is only between bars within a group and does not affect the gap between groups.

bar-gap= $\langle value \rangle$ *initial: 0*

A synonym of `bargap`.

groupgap= $\langle value \rangle$ *alias: group-gap; initial: 1*

For use with `\DTLmultibarchart`, this sets the gap between each group of bars for `\DTLmultibarchart`. The default value of 1 indicates a gap of one bar width so, for example, `groupgap=0.5` indicates a gap of half a bar.

group-gap= $\langle value \rangle$ *initial: 1*

A synonym of `groupgap`.

color-style

Determines which colour in the bar colour list should be used to fill the bar with the corresponding index.

color-style=default

The default setting will set the $\langle i \rangle$ th bar fill colour to the $\langle i \rangle$ th element of the bar colour list. If there is no such element or if $\langle i \rangle$ exceeds the maximum list index then `white` will be used. Note that if the background is also white, the white bars will only be visible if they have an outline (see Example 141).

color-style=single

The first colour in the colour list will be used for all bars.

color-style=cycle

This setting will cycle through the elements of the bar colour list. If an element is missing, `white` will be used. For example, suppose the colour list is cleared, and then only colours are set for index 1, 2 and 4:

```
\DTLclearbarcolors
\DTLsetbarcolor{1}{blue}
\DTLsetbarcolor{2}{red}
\DTLsetbarcolor{4}{green}
```

Then the first bar will be blue, the second red, the third white, and the fourth green. If there are additional bars, they will cycle back to the beginning with the fifth bar blue, the sixth red, the seventh white and the eighth green, etc. See Example 142.

bar-colors={ $\langle list \rangle$ }

Sets the bar colours. Each item in the list is passed to `\DTLsetbarcolor` with the item's index within the list as the bar index. Remember that the $\langle key \rangle = \langle value \rangle$ parser trims leading and trailing spaces and the CSV parser used for the argument removes empty items. This means that if you intend an empty item (to indicate no fill for that bar) then you need `{ }` to indicate this. Bear in mind that if you don't provide a fill colour you will need to set `outline-width` to ensure the outline is drawn (or redefine `\DTLBarStyle`).

If any bar colours have previously been assigned and this list is smaller, this will only override the given subset. Multiple instances of `bar-colors` within the same option list will only override the previous instance if the list is the same length or longer.

For example,


```
\DTLsetup{bar={
  bar-colors={pink, red, orange, green},
  bar-colors={cyan, magenta},
}}
```

This will assign cyan and magenta to the first two bars, but the third bar will be orange and the fourth green. Given the default settings, the fifth bar will also be magenta, the sixth bar will also be cyan, the seventh bar will be orange and the eighth bar will be white. Alternatively, you can use `\DTLsetbarcolor` to set the colour of a specific segment.

You can clear the list with `\DTLclearbarcolors` or reset the list with `bar-default-colors` or `bar-default-gray`. See Example 141.

bar-default-colors

Resets the first eight bar colours to the default colour set: red, green, blue, yellow, magenta, cyan, orange, white. Any additional bar colours will be unchanged.

bar-default-gray

Resets the first eight bar colours to the default set of eight shades of grey. Any additional bar colours will be unchanged.

negative-bar-colors = { *list* }

initial: empty

Sets the negative bar colour list. Any bar with a negative value will first consult the negative colour list. If there's no element for the required index, the default colour list will be consulted instead. See Example 143 for a bar chart with a different colour set for negative bars.

negative-color-style

Determines which colour in the negative bar colour list should be used to fill the bar with the corresponding index.

negative-color-style=single

The first colour in the negative colour list will be used for all bars. If the negative colour list is empty, the first colour in the default list will be used instead.

negative-color-style=cycle

This setting will cycle through the elements of the negative bar colour list.

negative-color-style=default

This setting will follow the same style as `color-style`.

outline-width= $\langle dimension \rangle$

initial: 0pt

Sets the width of the outline draw around each bar. The outline will only be drawn if the value is greater than 0pt. See Example 141.

outline-color= $\langle colour \rangle$

initial: black

Sets the bar outline colour. If the value is empty then the outline won't be drawn.

If you prefer, you can redefine `\DTLBarStyle` rather than setting `outline-width` and `outline-color`. For example, for a dotted outline that will be drawn even if `outline-width=0pt`:

```
\renewcommand{\DTLBarStyle}{draw,dotted}
```

5.2.3. Bar Chart Labels

Each bar may have a label at either end. The lower label is the label along the x -axis. The upper label is at the opposite end of the bar. For y tick labels, see §5.2.4.

barlabel= $\langle value \rangle$

alias: **bar-label**

This sets the lower bar label to $\langle value \rangle$, which will typically include a placeholder command provided in the $\langle assign-list \rangle$ argument of `\DTLbarchart` or the *databar* placeholders `\DTLbarvariable` or `\DTLbarvalue`. See examples in §5.3.3.

With `\DTLmultibarchart`, this setting provides the group label. Use `multibar-labels` for the individual lower bar labels in a multi-bar chart.

bar-label= $\langle value \rangle$

Synonym for `barlabel`.

upperbarlabel= $\langle value \rangle$

alias: **upper-bar-label**

For use with `\DTLbarchart`, this sets the upper bar label to $\langle value \rangle$, which will typically include a placeholder command provided in the $\langle assign-list \rangle$ argument of `\DTLbarchart` or

5. Bar Charts (*databar* package)

the `databar` placeholders `\DTLbarvariable` or `\DTLbarvalue`.



The upper bar label is ignored by `\DTLmultibarchart`.



upper-bar-label= $\langle value \rangle$

Synonym for `upperbarlabel`.



multibarlabels= $\langle value \rangle$ *alias: multi-bar-labels*

For use with `\DTLmultibarchart`, the value should be a comma-separated list, where each item will be used as the lower label for the corresponding bar within a bar group.



multi-bar-labels= $\langle list \rangle$

Synonym for `multibarlabels`.



The multi-bar labels are ignored by `\DTLbarchart`.



uppermultibarlabels= $\langle value \rangle$ *alias: upper-multi-bar-labels*

For use with `\DTLmultibarchart`, the value should be a comma-separated list, where each item will be used as the upper label for the corresponding bar within a bar group.



upper-multi-bar-labels= $\langle list \rangle$

Synonym for `uppermultibarlabels`.



group-label-align= $\langle value \rangle$

For use with `\DTLmultibarchart`, the value should be the `\pgftext` alignment for the bar group labels. The default is to use the same alignment as for the lower bar labels.



label-offset= $\langle dim \rangle$ *initial: 10pt*

The distance from the x -axis to the lower bar label. The value should be a dimension. (This option sets the `\DTLbarlabeloffset` length register.) Note that the direction is dependent on

lower-label-style.

upper-label-offset=*<value>* *initial: \DTLbarlabeloffset*

The distance from the outer edge of the bar (that is, the edge furthest from the x -axis) to the upper bar label. This may be given in terms of `\DTLbarlabeloffset`. If you change this to a negative value (which will cause the upper bar label to shift inside the end of the bar) then you will also need to change the alignment. For example:

```
upper-label-offset=-\DTLbarlabeloffset,
upper-label-align=[left]right
```

5.2.4. Bar Chart Axes

The bar chart x axis is horizontal for vertical bars and vertical for horizontal bars. The y axis corresponds to the bar data value. The y -axis may have tick marks. The x -axis doesn't (but may have labels at either end of each bar, see §5.2.3). The y tick marks follow the same algorithm as those for `dataplot` (see §6.1.4).

axes=*<value>* *default: both; initial: none*

Determines whether or not to display the axes.

axes=both

Switches on x and y axes and y ticks.

axes=none

Switches off x and y axes and y ticks.

axes=x

Switches on x axis, and switches off y axis and y ticks.

axes=y

Switches on y axis and y ticks, and switches off x axis.

y-axis= $\langle$ boolean $\rangle$

default: **true**; initial: **false**

Switches the y axis on ($y\text{-axis}=\text{true}$) or off ($y\text{-axis}=\text{false}$) without affecting the x axis setting.

yaxis= $\langle$ boolean $\rangle$

alias: **y-axis**; default: **true**; initial: **false**

Synonym of $y\text{-axis}$.

y-ticks= $\langle$ boolean $\rangle$

default: **true**; initial: **false**

Indicates whether or not to draw y ticks. If true, this option will automatically set $y\text{-axis}=\text{true}$. If false, no change will be made to the y axis setting. Tick marks will only be drawn if the axis is also drawn so, for example, $y\text{-ticks}=\text{true}$, $y\text{-axis}=\text{false}$ won't show any tick marks.

ytics= $\langle$ boolean $\rangle$

alias: **y-ticks**; default: **true**; initial: **false**

Synonym of $y\text{-ticks}$.

round= $\langle$ n $\rangle$

initial: **1**

Sets the DTLbarroundvar counter, which is used to round the variable value when setting the $\backslash$ DTLbarvalue placeholder command. It will also be used to round the y tick labels unless $y\text{-tick-round}$ is set.

y-tick-round= $\langle$ n $\rangle$

default: *empty*; initial: *empty*

Sets the rounding used in default y tick labels. If not set, this will match the `round` option. If the supplied value is empty or missing, it will revert back to using the DTLbarroundvar counter.

ytic-round

alias: **y-tick-round**

Synonym of $y\text{-tick-round}$.

ylabel= $\langle$ value $\rangle$

The label for the y axis.

ylabel-position= $\langle value \rangle$

initial: center

Sets the x position of the y label where the value may be one of the following keywords.

ylabel-position=center

Sets the x position of the y label to the centre of the x axis.

ylabel-position=zero

Sets the x position of the y label to 0 (see Example 140).

ylabel-position=min

Sets the x position of the y label to the minimum x axis value.

ylabel-position=max

Sets the x position of the y label to the maximum x axis value.

max= $\langle value \rangle$

default: empty; initial: empty

The maximum extent of the y axis (as a plain number in data units). If an empty $\langle value \rangle$ is supplied, the maximum extent will be the maximum non-negative value found in the data (or 0 if no non-negative values found).

maxdepth= $\langle value \rangle$

alias: max-depth; default: empty; initial: empty

The negative extent of the bar chart. The value must be a plain number in data co-ordinates less than or equal to 0. If an empty $\langle value \rangle$ is supplied, the negative extent will be the lowest (closest to $-\infty$) negative value found in the data or 0 if there are no negative values.

max-depth= $\langle value \rangle$

default: empty; initial: empty

Synonym of maxdepth.

y-tick-gap= $\langle value \rangle$

Sets the gap between y tick marks. This option automatically sets `y-ticks=true` and `y-axis=true`. If both `y-tick-gap` and `y-tick-points` are omitted, the y tick

5. Bar Charts (*databar package*)

marks (if `y-ticks=true`) will be calculated according to the minimum and maximum values and the minimum tick gap length `\DTLmintickgap`.



You can't set both `y-tick-gap` and `y-tick-points`.



ycticgap=*<value>*

alias: **y-tick-gap**

Synonym of `y-tick-gap`.



y-tick-points=*<list>*

A comma-separated list of points for the *y* ticks. This option automatically sets `y-ticks=true` and `y-axis=true`. Tick marks outside the *y* range (from `maxdepth` to `max`) will be omitted.



ycticpoints=*<value>*

alias: **y-tick-points**

Synonym of `y-tick-points`.



y-tick-labels=*<list>*

A comma-separated list of labels for the *y* ticks. This option automatically sets `y-ticks=true` and `y-axis=true`. If omitted and `y-ticks=true`, the *y* tick value will be used. Note that if there are more items in `y-tick-points` than in `y-tick-labels` then the default tick labels will be used for the missing items.



yticlabels=*<list>*

alias: **y-tick-labels**

Synonym of `y-tick-labels`.



y-tic-label-align=*<value>*

alias: **y-tick-label-align**

Sets the *y*-tick label alignment. The value should be able to expand to a `\pgftext` alignment specifier. (The expansion occurs within `\DTLbarchart` and `\DTLmultibarchart` not when the option is set.)



ytic-label-align=*<value>*

alias: **y-tick-label-align**

Synonym of `y-tic-label-align`.



```
y-tick-label-align=<value>
```

Another synonym of `y-tic-label-align`.

5.3. Bar Chart Examples

5.3.1. Labels

Example 130 adapts Example 125 to label each bar of the chart. The placeholder commands may be used within the labels. In this case, the fruit name is placed in the lower label and the quantity is placed in the upper label, which can be done by adding an extra assignment in the final argument:

 130



```
\DTLbarchart
{
  variable=\Quantity,% variable required
  bar-label=\Name,
  upper-bar-label=\Quantity,
}
{fruit}% database
{\Quantity=Quantity,\Name=Name}% assignment list
```

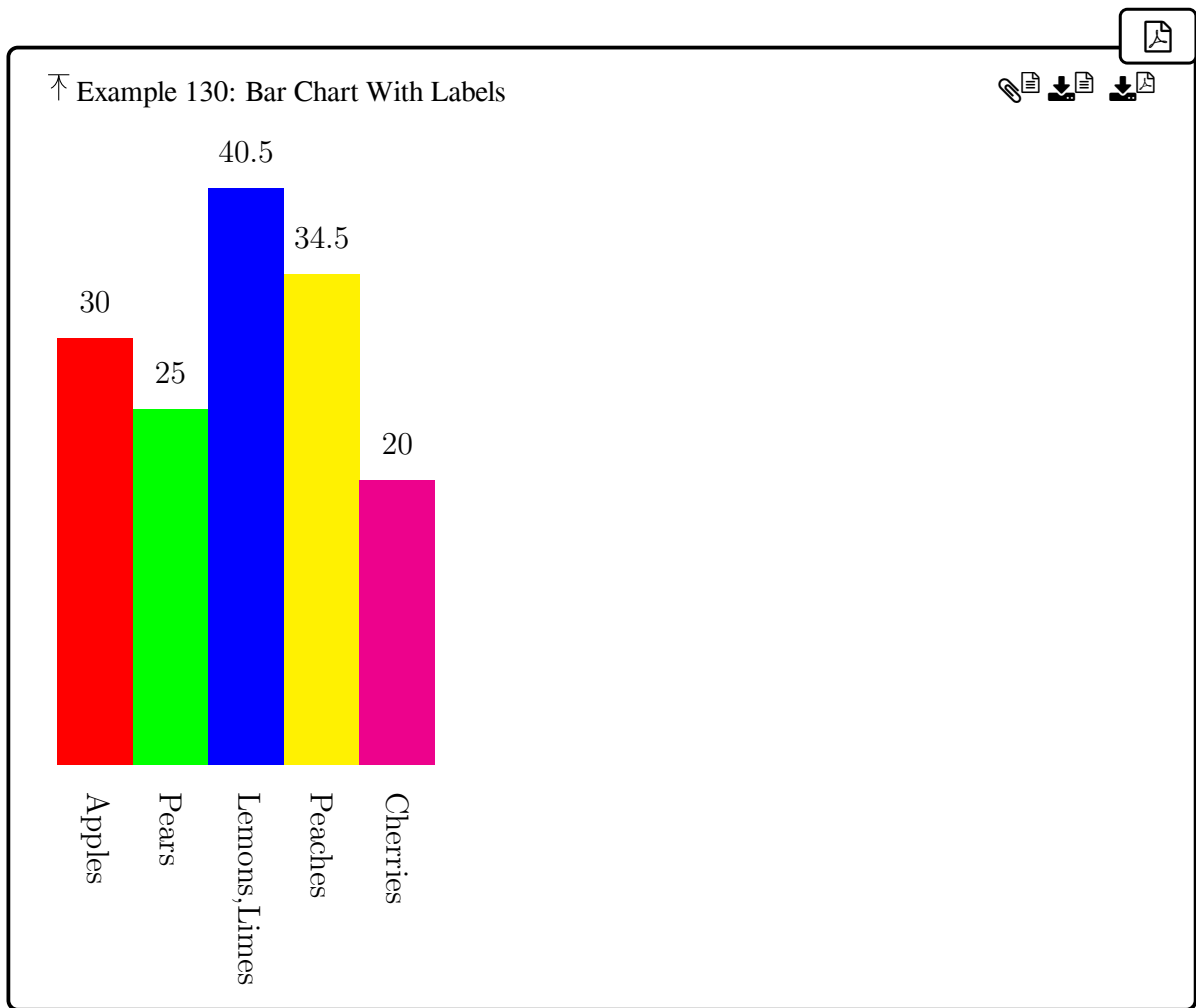
The placeholder commands are assigned using `\DTLmapgetvalues` (since the drawing code for each bar is constructed within the body of `\DTLmapdata`) so they will expand as they would ordinarily if `\DTLmapgetvalues` is used explicitly within `\DTLmapdata`. If you want the value to be rounded, then use `\DTLbarvalue` with the `round` option instead. For example:



```
\DTLbarchart
{
  variable=\Quantity,% variable required
  bar-label=\Name,
  round=0,
  upper-bar-label=\DTLbarvalue,
}
{fruit}% database
{\Quantity=Quantity,\Name=Name}% assignment list
```

The `\DTLbarvalue` command is also useful with `bar chart` and `multibar chart` actions where the column (or columns, for `multibar chart`) can simply be identified by

5. Bar Charts (*databar* package)



5. Bar Charts (databar package)

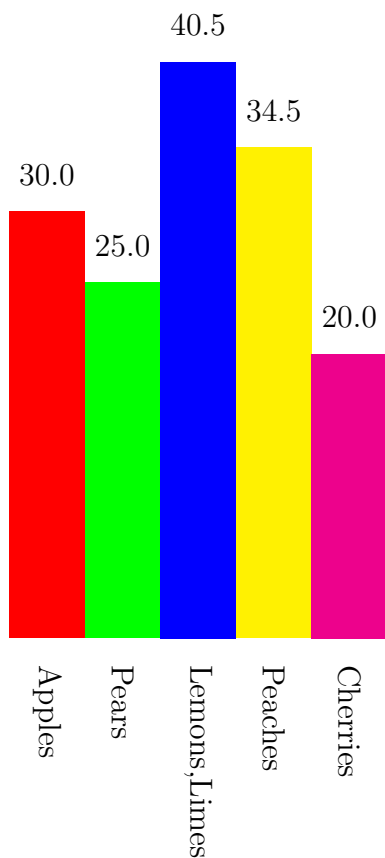
key or index rather than assigning a placeholder command.

Example 131 adapts Example 126 and uses `\DTLmapget` and `\DTLbarvalue` to avoid the need to assign any placeholder commands.

131

```
\DTLaction
[
  key=Quantity,
  options={
    bar-label={\DTLmapget{key=Name}},
    upper-bar-label=\DTLbarvalue
  }
]
{bar chart}
```

Example 131: Bar Chart With Labels (Action 'bar chart')



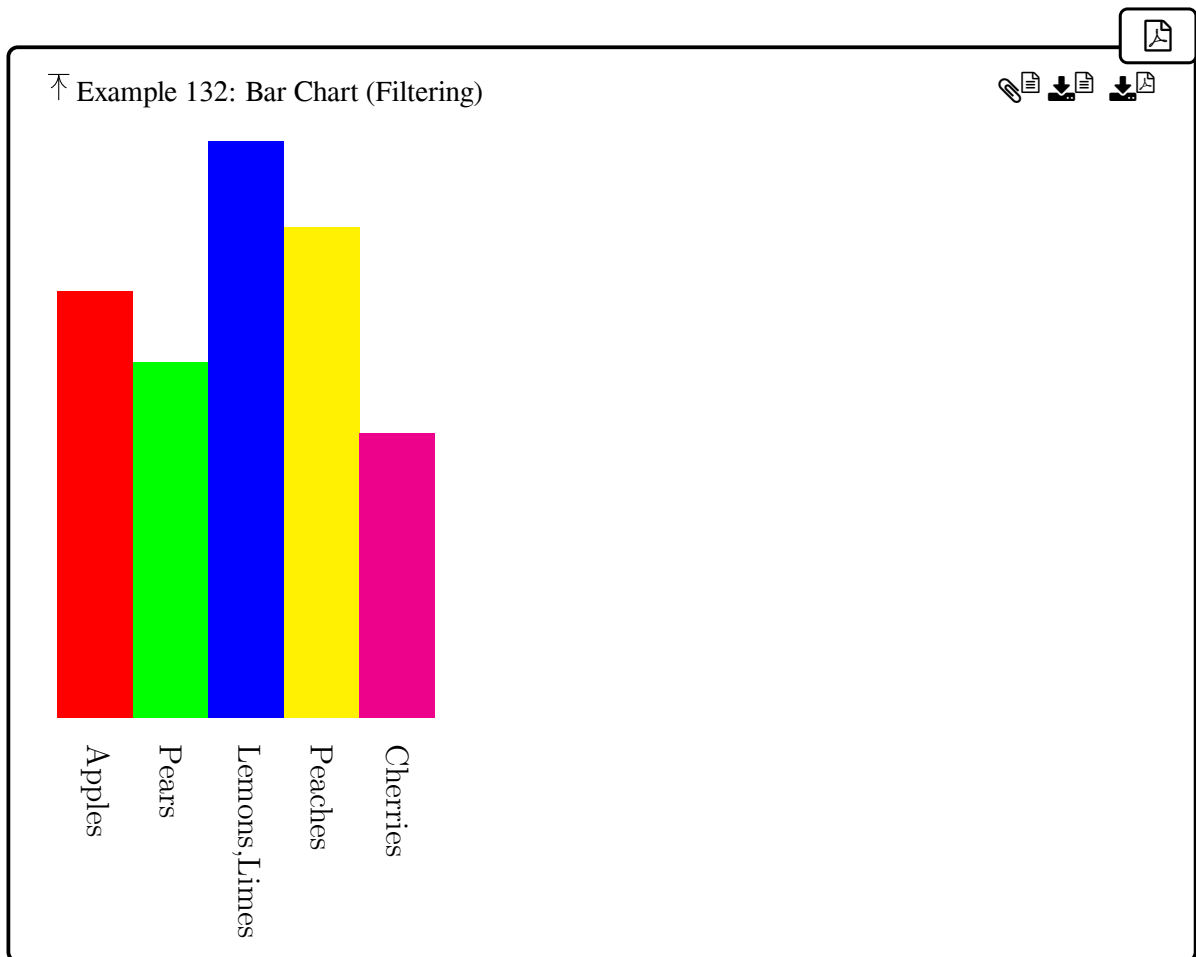
5.3.2. Filtering

Example 132 modifies Example 125 to skip the “Pears” row using the `include-if` setting:

132

```
\DTLbarchart
{
  variable=\Quantity,% variable required
  bar-label=\Name,
  include-if={\DTLifstringeq{\Name}{Pears}}{\#1}
}
{fruit}% database
{\Quantity=Quantity,\Name=Name}% assignment list
```

This uses `\DTLifstringeq` to test value of the placeholder command.



Compare Example 132 with Example 116 which is identical except that one uses `\DTLbarchart` and the other one uses `\DTLpiechart`. As with Example 116, the same result can also be achieved with the optional argument instead of `include-if` or you can use

`include-if-fn`, but bear in mind that you can't use both the optional argument and the `include-if/include-if-fn` setting.

5.3.3. Lower Label Alignment

Example 133 adapts Example 127 to include upper and lower labels. This uses the default settings for `lower-label-style` and `upper-label-align`. The lower bar label is the year (obtained from the `Year` column which is assigned to the `\theYear` placeholder command). The upper bar label is the value (profit or loss). This can be referenced with the placeholder command `\theProfit` or with `\DTLbarvariable`, but this will show the value as given in the database (which includes the currency symbol and has inconsistently formatted negative values, see §3.2.8). With `\DTLbarvalue`, the value will be shown as a formatted number without the currency symbol, rounded according to the `round` setting.

133

```
\DTLbarchart
{
  variable=\theProfit,
  horizontal,
  bar-width=20pt,
  bar-label=\theYear,
  round=0,
  upper-bar-label=\DTLbarvalue,
}
{profits}% database name
{% assignment list
\theYear=Year,
\theProfit=Profit
}
```

An alternative is to reformat the values while the data is read from the file, as is done in Example 113.

Example 134 has a minor modification to Example 133 that sets `lower-label-style=same`, which puts the lower label on the same side of the x axis as the bar. Note that this causes a clash with the upper bar labels for short bars.

134

Example 135 has a minor modification to Example 133 that sets `lower-label-style=below`, which puts the lower label below the x axis as the bar (where below means $y < 0$, which is to the left for horizontal charts). This means that it overlaps bars with negative values which again causes a clash with the upper bar labels for short bars.

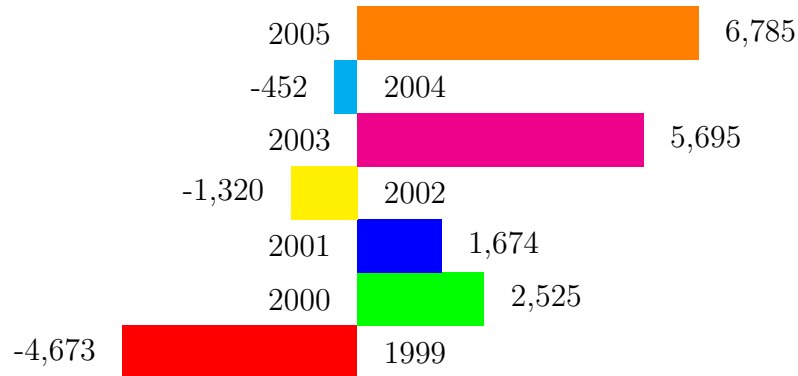
135

Example 136 has a minor modification to Example 133 that sets `lower-label-style=above`, which puts the lower label above the x axis as the bar (where above means $y > 0$, which is to the right for horizontal charts). This means that it overlaps bars with positive values which again causes a clash with the upper bar labels for short bars.

136

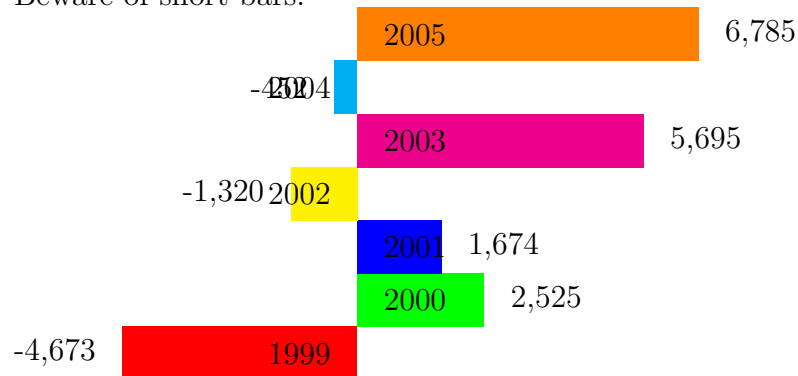
5. Bar Charts (*databar* package)

↑ Example 133: Horizontal Bar Chart with Labels (Default Alignment)



↑ Example 134: Horizontal Bar Chart with Labels (`lower-label-style=same`)

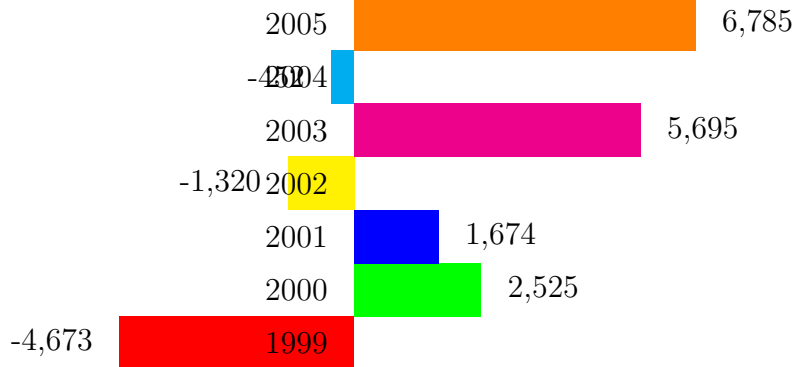
Beware of short bars!



5. Bar Charts (*databar* package)

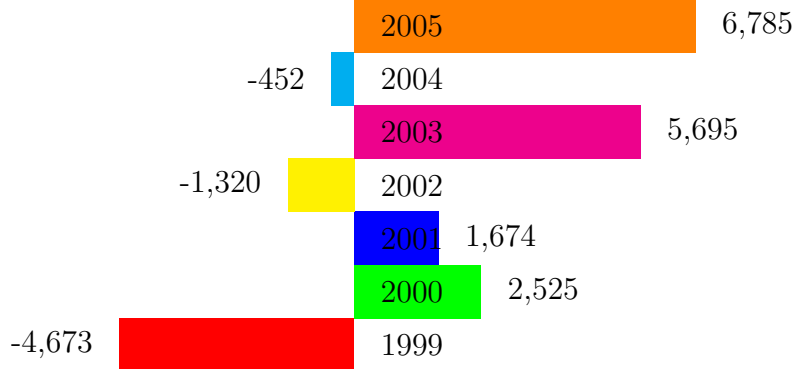
↑ Example 135: Horizontal Bar Chart with Labels (`lower-label-style=below`)

Beware of short bars!



↑ Example 136: Horizontal Bar Chart with Labels (`lower-label-style=above`)

Beware of short bars!



5.3.4. Upper Label Alignment

The examples in §5.3.3 demonstrate the default style for upper labels. Example 137 modifies Example 133 to shift the upper labels inwards so that they overlap the bars:

137

```
\DTLbarchart
{
  variable=\theProfit,
  horizontal,
  bar-width=20pt,
  bar-label=\theYear,
  round=0,
  upper-bar-label=\DTLbarvalue,
  upper-label-offset={ -\DTLbarlabeloffset },
  upper-label-align=[left]right
}
{profits}% database name
{% assignment list
\theYear=Year,
\theProfit=Profit
}
```

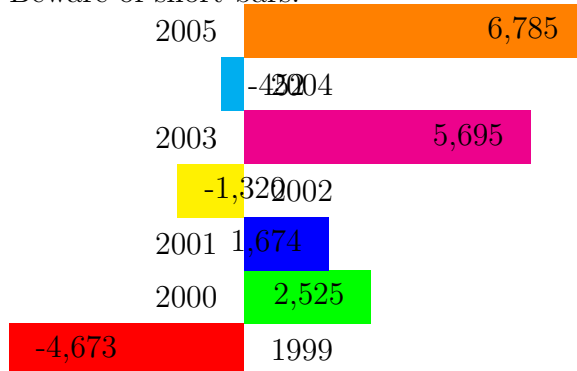
This sets `upper-label-offset` to a negative dimension to shift the upper label inwards (towards the x -axis). Note that this shifts the upper labels for negative bars to the right and for the positive bars to the left. This now means that the node alignment has to be changed for the upper labels: `left` for the negative upper labels and `right` for the positive upper labels. Again, this can cause a problem for short bars.

Bear in mind that if you switch to vertical bars the node alignment will need to be changed to `upper-label-align=[above]below`. (Alternatively, include the `\ifDTLvertical-bars` conditional in the value to allow for either orientation.)

5.3.5. Multi Bar Chart Labels

In addition to lower and upper bar labels, multi bar charts created with `\DTLmultibar-chart` also have group labels. The upper labels are positioned at the bar end, as for `\DTLbarchart`, and the lower labels are positioned at the bar start (near the x -axis). However, the lower labels should now be specified with `multibarlabels` (or `multi-bar-labels`) and the upper labels with `uppermultibarlabels` (or `upper-multi-bar-labels`). These options take comma-separated lists for their values, where each item in the list corresponds to the bar within the group.

↑ Example 137: Horizontal Bar Chart with Upper Labels Over the Bars
(negative upper-label-offset)
Beware of short bars!



Each set of bars within a group corresponds to one row of the database. This means that the placeholder commands set in the *<assign-list>* final argument of `\DTLmultibar-chart` will be the same for each bar label within the group.

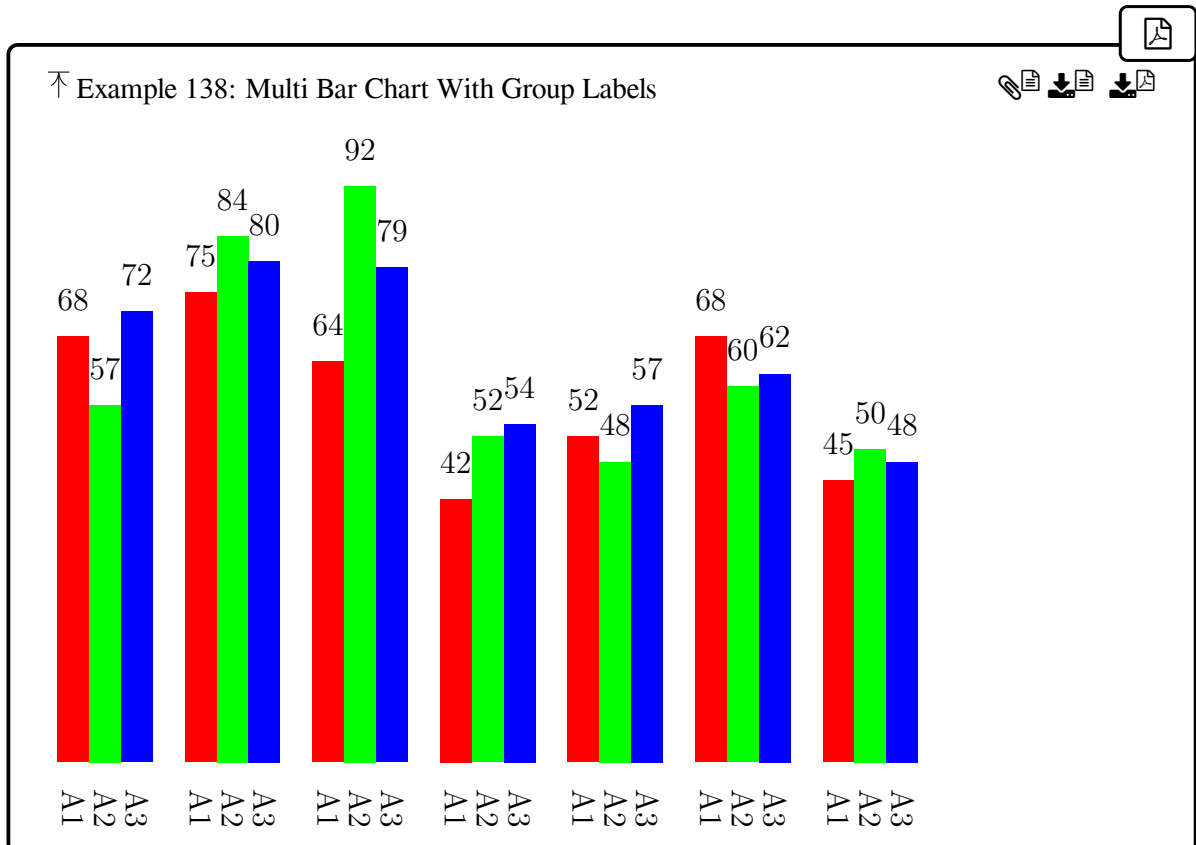
The group label normally is positioned at the mid point of the group below the lower bar labels (if present). Example 138 adapts Example 128 so that the student marks are shown at the end of the bar (the upper label). The lower label indicates the assignment (A1 for assignment 1, A2 for assignment 2, and A3 for assignment 3) and uses the default alignment (rotated for vertical bars). Below that, for each group, is the group label, which is set to the student's initials. The group label alignment is changed so that the group labels aren't rotated.

138

```
\DTLmultibarchart
{
  variables={\assignI,\assignII,\assignIII},
  round=0,
  bar-width=12pt,
  group-label-align={center,top},
  bar-label={\xDTLinitials{\Forename}\xDTLinitials
{\Surname}},
  multi-bar-labels={A1,A2,A3},
  upper-multi-bar-labels={\DTLbarvalue,\DTLbar-
value,\DTLbarvalue},
}
{marks}% database name
```


5. Bar Charts (*databar package*)

```
{
  \assignI=Assign1,
  \assignII=Assign2,
  \assignIII=Assign3,
  \Surname=Surname,
  \Forename=Forename
}
```



This leads to a cluttered chart. A better solution would be to add a y -axis with tick marks to show the values and a legend associating each bar colour with the corresponding assignment (see Example 148).

5.3.6. Axes

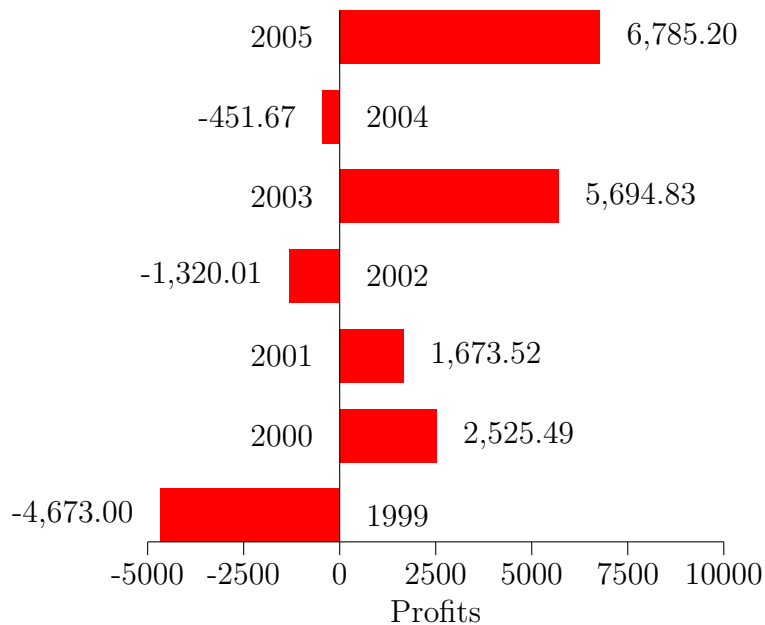
Example 139 adapts Example 127 to include axes and y tick labels. Note that the rounding for the y tick labels and in the definition of the `\DTLbarvalue` placeholder are set differently. The bar width (which for a horizontal bar chart is in fact the height) is set to 20pt and the bar gap is set to 0.5, which is half a bar width (and therefore 10pt).

139

5. Bar Charts (databar package)

```
\DTLbarchart
{
  variable=\theProfit,
  horizontal,
  color-style=single,
  max-depth=-5000,
  max=10000,
  bar-width=20pt,bar-gap=0.5,
  axes,y-ticks,y-tick-gap=2500,
  ylabel=Profits,
  bar-label=\theYear,
  upper-bar-label=\DTLbarvalue,
  round=2,
  y-tick-round=0
}
{profits}% database
{% assignment list
\theYear=Year,
\theProfit=Profit
}
```

↑ Example 139: Bar Chart With Axes



Note that the y axis label (“Profits”) is aligned at the mid point of the x axis, below the y tick labels. This can look strangely off-centre if the origin isn’t in the middle of the x axis (as in this case, since the absolute value of the negative extent is smaller than the maximum positive extent.)

Example 140 adapts Example 139 to align the y label at $x = 0$ (using `ylabel-position=zero`). The tick gap has also been made smaller, which would cause the tick labels to overlap, so `\DTLbardisplayYticklabel` has been redefined to rotate the tick label. The alignment for the `\pgftext` command is also modified to allow for the rotation:

140

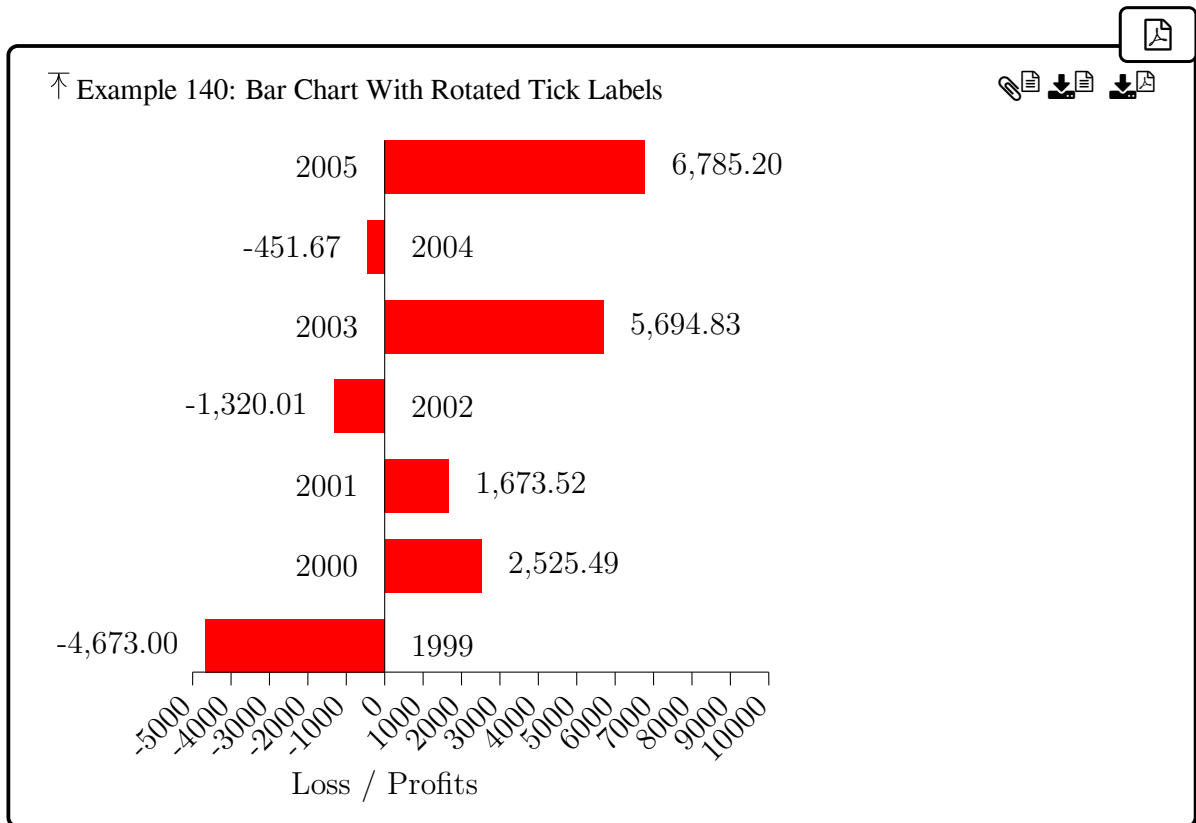
```
\renewcommand{\DTLbardisplayYticklabel}[1]
{\rotatebox{45}{#1}}
\DTLbarchart
{
  variable=\theProfit,
  horizontal,
  color-style=single,
  max-depth=-5000,
  max=10000,
  bar-width=20pt,bar-gap=0.5,
  axes,y-ticks,y-tick-gap=1000,
  ylabel=Loss / Profits,
  ylabel-position=zero,
  bar-label=\theYear,
  upper-bar-label=\DTLbarvalue,
  round=2,
  y-tick-round=0,
  ytic-label-align={right,top}
}
{profits}% database
{% assignment list
\theYear=Year,
\theProfit=Profit
}
```

5.3.7. Bar Colours

Example 141 adapts Example 125 so that bar colour list is locally cleared and set to just three colours. Since there are more than three bars, the remaining bars are white with the default setting. Note that without the outline those bars won’t be visible.

141

5. Bar Charts (*databar* package)

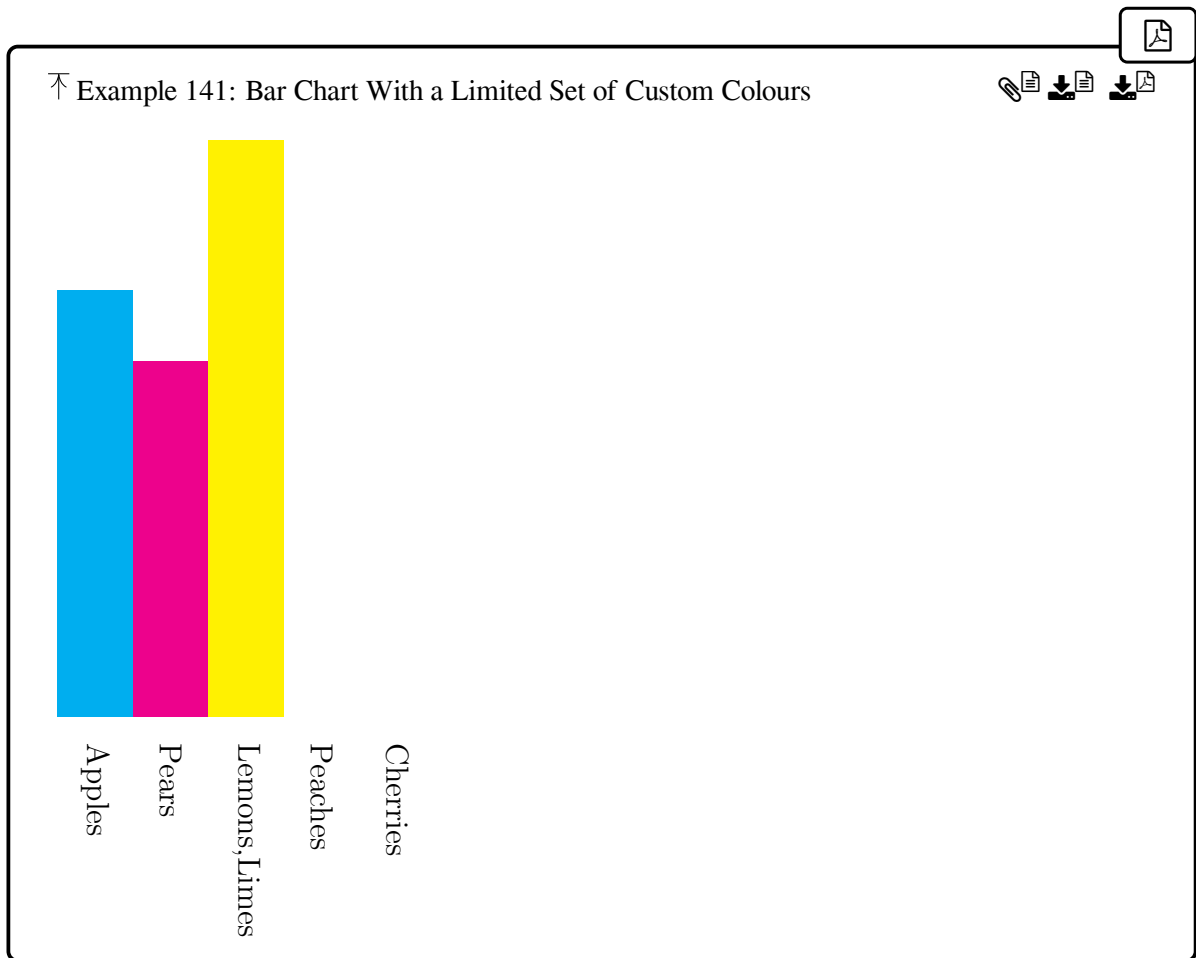


```
\DTLbarchart
{
  variable=\Quantity,
  bar-label=\Name,
  pre-init={\DTLclearbarcolors},
  outline-width=1pt,
  bar-colors={cyan,magenta,yellow}
}
{fruit}% database
{\Quantity=Quantity,\Name=Name}% assignment list
```

Remember to use `pre-init` rather than `init` in this case to ensure that the colour list is cleared before the `bar-colors` option is processed.

If there are insufficient colours, you can cycle round the set. Example 142 adapts Example 141 so that the fourth bar uses the first colour in the set, and the fifth bar uses the second colour.

142



5. Bar Charts (*databar* package)

```
\DTLbarchart
{
  variable=\Quantity,
  bar-label=\Name,
  pre-init={\DTLclearbarcolors},
  bar-colors={cyan,magenta,yellow},
  color-style=cycle,
}
{fruit}% database
{\Quantity=Quantity,\Name=Name}% assignment list
```

↑ Example 142: Bar Chart Cycling through the Colour Set



By default, the negative bars will use the same colour set as the positive bars (see Example 127). Example 143 adapts Example 127 so that bars with positive values are blue and bars with negative values are red:

143

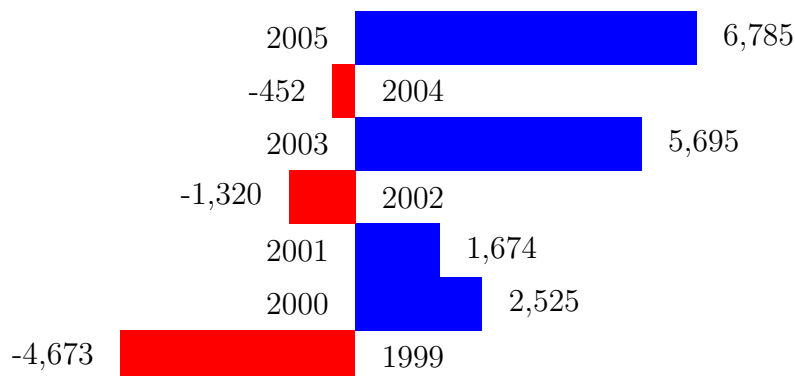
```

\DTLbarchart
{
  variable=\theProfit,
  horizontal=,
  bar-width=20pt,
  bar-label=\theYear,
  upper-bar-label=\DTLbarvalue,
  round=0,
  color-style=single,
  bar-colors=blue,
  negative-bar-colors=red
}
{profits}% database
{% assignment list
\theYear=Year,
\theProfit=Profit
}

```

Note that without `negative-bar-colors=red`, all bars would be blue with `color-style=single`, `bar-colors=blue`.

↑ Example 143: Single Colours for Positive and Negative Bars



Each bar is drawn using `tikz's \path` command with the option list including `fill=<colour>` if a non-empty colour specification has been set for the bar colour and `draw=<colour>` if the `outline-color` is non-empty and the `outline-width` is greater than `0pt`. This will then be followed by the full expansion of `\DTLBarStyle`, which can be used to make further adjustments.

Example 144 adapts Example 139 to shade each bar with a gradient from red on the left to blue on the right. The normal fill action is disabled by setting the fill colour for the first bar to empty

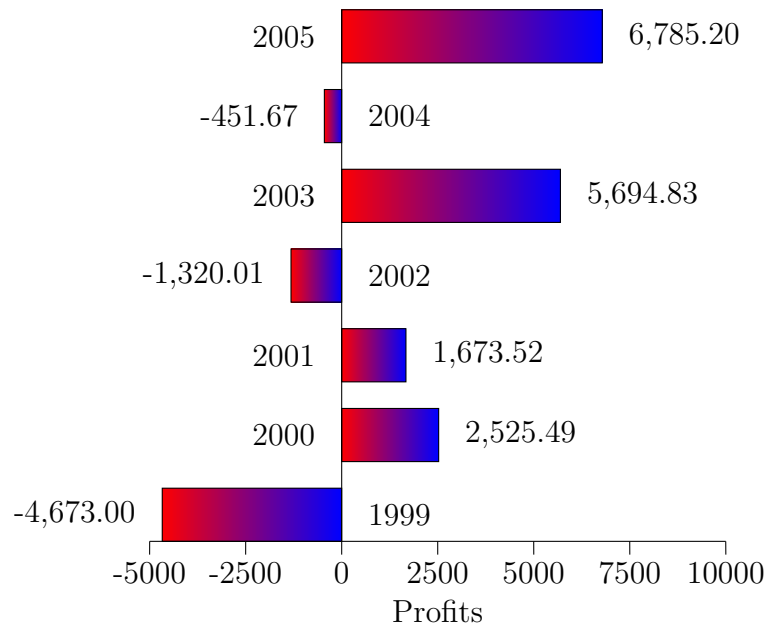
144

5. Bar Charts (*databar package*)

and using `color-style=single` to use that setting for all bars. The actual bar style is then obtained from `\DTLBarStyle` which has the shading settings to be passed to `\path`:

```
\renewcommand\DTLBarStyle{%  
  draw,shade,shading=axis,left color=red,right color=blue  
}
```

↑ Example 144: Shaded Bar



If the `\DTLBarStyle` content needs to be constructed according to each bar variable, then you can redefine `\DTLeveryprebarhook` to redefine `\DTLBarStyle` as applicable.

5.3.8. Hooks

Example 145 adapts Example 127 to set all bars to a single colour (pink) and uses the after every bar hook to draw a line from the start point (`\DTLstartpt`) to the end point (`\DTLendpt`) and draw a circle at the start, middle (`\DTLmidpt`), and end points:

145

```
\renewcommand\DTLeveryprebarhook{%  
  \pgfpathmoveto{\DTLstartpt}  
  \pgfpathlineto{\DTLendpt}
```

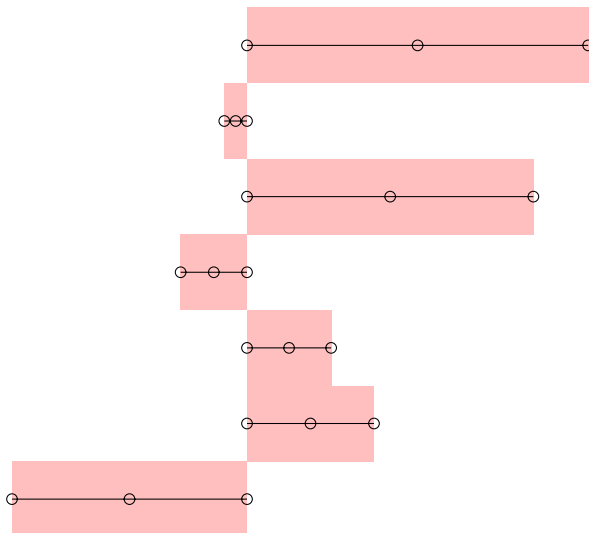

5. Bar Charts (*databar package*)

```

\pgfpathcircle{\DTLstartpt}2pt
\pgfpathcircle{\DTLmidpt}2pt
\pgfpathcircle{\DTLendpt}2pt
\pgfusepath{draw}
}%
\DTLbarchart
{
  variable=\theProfit,
  horizontal,
  color-style=single,
  bar-colors=pink,
}
{profits}% database
{% assignment list
\theYear=Year,
\theProfit=Profit
}

```

Example 145: Hook at Every Bar



Example 146 modifies Example 132 to show the bar index (`\DTLbarindex`) and the row index number (`\dtlrownum`) in the centre of each bar. The second row (Pears) has been skipped so the row index numbering is: 1, 3, 4 and 5. Compare this with the bar index numbering: 1, 2, 3, and 4.

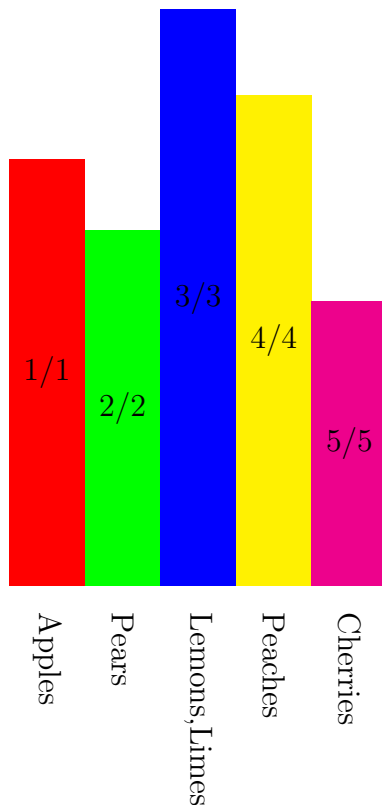
146

```

\DTLbarchart
{
  init={\renewcommand{\DTLeverybarhook}{%
    \pgftext[at=\DTLmidpt]{\DTLbarindex/\number\dtlrow-
num}}},
  variable=\Quantity,% variable required
  bar-label=\Name,
  include-if={\DTLifstringeq{\Name}{Pears}}{\#1}}
}
{fruit}% database
{\Quantity=Quantity,\Name=Name}% assignment list

```

↑ Example 146: Every Bar Hook (Filtering)



`\DTLeverybarhook` can be used to gather information for use in the end hook `\DTLbaratendtikz`. This is done in Example 147 which modifies Example 125 to move the labels into a legend.

147

```

\DTLbarchart
{
  init={%
    \def\mychartlegend{ }%
    \renewcommand{\DTLeverybarhook}{%
      \ifdefempty{\mychartlegend}{ }
    }%
    {\appto\mychartlegend\ }%
    \eappto\mychartlegend{%
      {\noexpand\DTLdobarcolor{\DTLbarindex}
        \noexpand\rule{\noexpand\DTLbarwidth}
      }%
      {\noexpand\DTLbarwidth}}
      \expandonce\Name
    }%
  }%
  \renewcommand{\DTLbaratendtikz}{%
    \node[at=(\DTLbarchart-
width,0pt),anchor=south west]
    {
      \begin{tabular}{l}
        \mychartlegend
      \end{tabular}
    };
  }%
},
variable=\Quantity,% variable required
y-ticks,ylabel=Quantity
}
{fruit}% database
{\Quantity=Quantity,\Name=Name}% assignment list

```

For a multi bar chart, the colours change for each bar within the group. If the above `\DTLeverybarhook` code is used for the student scores database multi bar chart then the legend would end up with the number of students times the number of assignments bars.

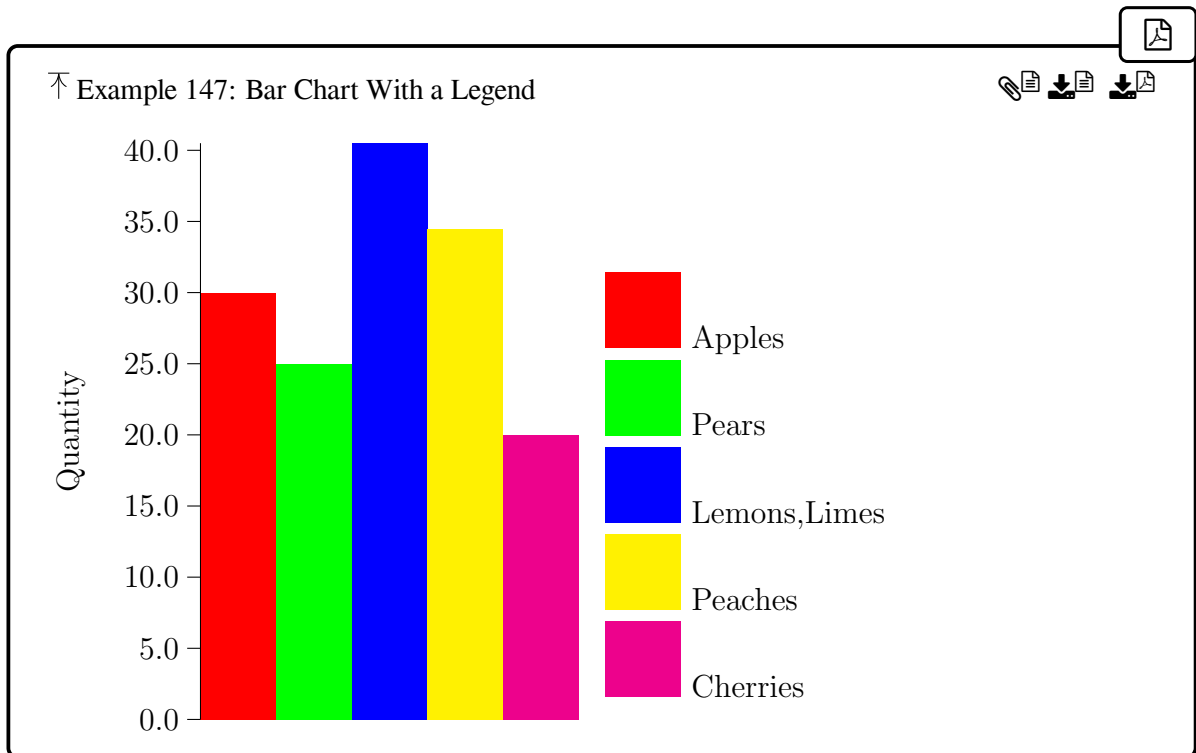
Example 148 modifies Example 138 to include a legend rather than labelling every bar. In this case, the number of bars is known as the information is required for the `variables` list.

148

```

\DTLmultibarchart
{
  init={
    \renewcommand{\DTLbaratendtikz}{%

```



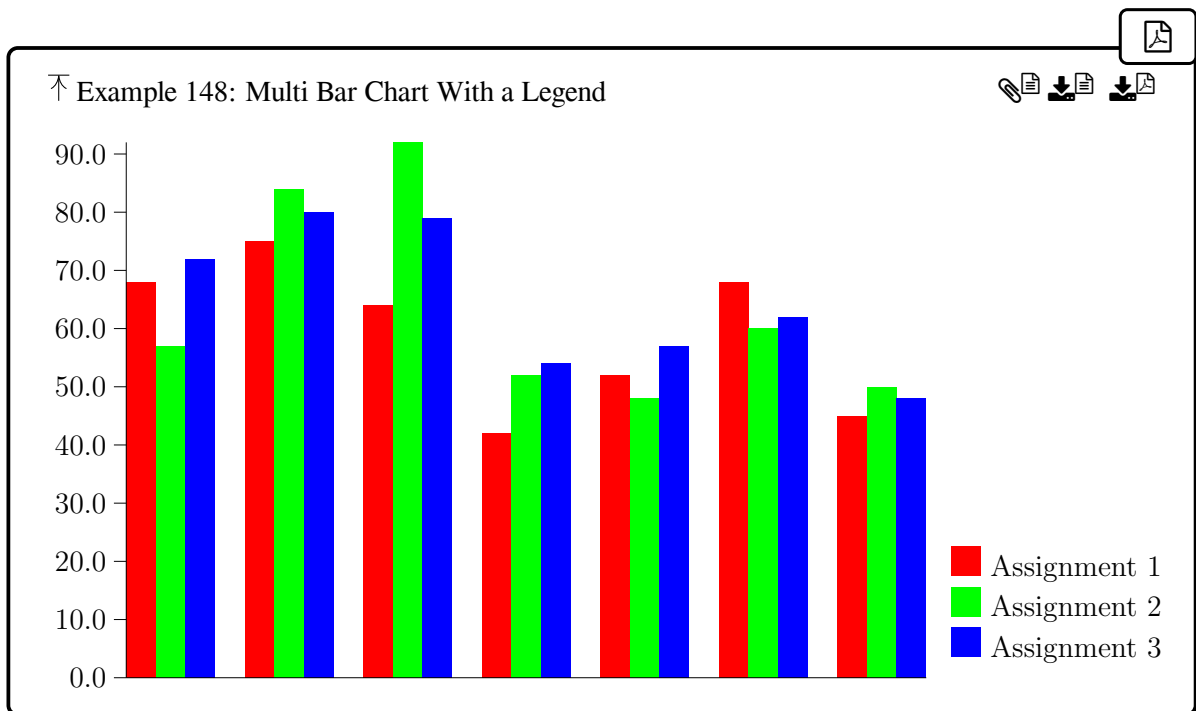
```

\newcommand{\DTLbarchart-
width,0pt),anchor=south west]
{\begin{tabular}{l}
{\DTLdobarcolor{1}\rule{\DTLbarwidth}{\DTL-
barwidth}}
Assignment 1\\
{\DTLdobarcolor{2}\rule{\DTLbarwidth}{\DTL-
barwidth}}
Assignment 2\\
{\DTLdobarcolor{3}\rule{\DTLbarwidth}{\DTL-
barwidth}}
Assignment 3
\end{tabular}};
}
},
variables={\assignI,\assignII,\assignIII},
bar-width=12pt,
group-label-align={center,top},
bar-label={\xDTLinitials{\Forename}\xDTLinitials
{\Surname}},
y-ticks,axes=both

```

5. Bar Charts (*databar* package)

```
}
{marks}% database name
{
  \assignI=Assign1,
  \assignII=Assign2,
  \assignIII=Assign3,
  \Surname=Surname,
  \Forename=Forename
}
```



5.4. Bar Chart Associated Commands

5.4.1. Axes

```
\ifDTLverticalbars <true>\else <false>\fi      initial: \iftrue
```

This conditional is set by the `verticalbars` setting, which additionally changes the label alignment. (If you change this conditional explicitly, you will also need to redefine the alignment commands.) This conditional may be referenced in hooks to determine the chart orientation.

\DTLbarchartlength*initial: 3in*

A length register used to set the total bar chart y -axis length. This may be changed with `\setlength` or via the `length` option.

\DTLbarmax*initial: empty*

This command should either be defined to expand to nothing, in which case the maximum extend will be obtained from the data, or it should be defined to the maximum y value. This command is redefined by the `max` setting.

\DTLnegextent*initial: empty*

This command should either be defined to expand to nothing, in which case the negative will be obtained from the data, or it should be defined to the negative extent (which must be a decimal value less than or equal to 0). This command is redefined by the `maxdepth` setting.

\DTLBarXAxisStyle*initial: -*

The expansion text of this command should be the `tikz` line style specifier for drawing the x -axis. The `x-axis-style` option redefines this command.

\DTLBarYAxisStyle*initial: -*

The expansion text of this command should be the `tikz` line style specifier for drawing the y -axis. The `y-axis-style` option redefines this command.

\DTLBarStyle*initial: empty*

This command is fully expanded and then appended to the list of options for `\path` when each bar is drawn.

5.4.2. Textual

DTLbarroundvar*No*

Counter used to govern the number of digits to round to for display purposes. Note that changing the counter with `\setcounter` will globally change the value. To locally change it, use the `round` setting. The `y-tick-round` setting is initialised to use this counter value, so if you

5. Bar Charts (*databar package*)

change this counter or set `round` but don't set `y-tick-round` then the default *y* tick labels will use the same rounding that's applied to the definition of `\DTLbarvalue`.

`\DTLbarvariable`

Placeholder command that may be used in lower or upper labels to show the bar value (as the value appears in the database).

`\DTLbarvalue`

Placeholder command that may be used in lower or upper labels to show the bar value as a formatted number that has been rounded according to the `round` setting. This command will actually be a decimal datum control sequence. The numeric value part will include the rounding.

For example, if the original value in the database column labelled `Profit` for the current bar is `-\$12,345.62` and the settings include `variable=\theProfit` with the assignment list `\theProfit-Profit`, then `\DTLbarvariable` will expand to the supplied variable `\theProfit`, which in turn will expand to `-\$12,345.01`, but `\DTLbarvalue` will expand to the rounded localised decimal value. For example, with `round=0` this will be `-12,345` (without the currency symbol).

`\DTLbarindex`

Placeholder command that expands to the current bar index (starting from 1). Note that with `\DTLmultibarchart`, this index is reset at the start of each group.

`\DTLbargroupindex`

Placeholder command that expands to the current bar group index (starting from 1). Note that with `\DTLbarchart`, this will expand to 0.

`\DTLbarwidth`

initial: 1cm

A length register used to store the bar width. This may be changed with `\setlength` or via the `barwidth` option.

`\DTLbarlabeloffset`

initial: 10pt

A length register used to store the distance from the *x*-axis to the lower bar label. This may be changed with `\setlength` or via the `label-offset` option.

`\DTLbarXlabelalign`

5. Bar Charts (*databar* package)

This command should expand to the `\pgftext` alignment specifications for the x -axis lower bar labels for positive values. As from v3.0, the `\ifDTLverticalbars` conditional has been moved into the definition of `\DTLbarXlabelalign` as it is now fully expanded before being passed to the pgf parser. The default definition is now:

```
\newcommand\DTLbarXlabelalign{%  
  \ifDTLverticalbars left,rotate=-90\else right\fi  
}
```

This command is redefined by the `lower-label-style` setting.

`\DTLbarXneglabelalign`

This command should expand to the `\pgftext` alignment specifications for the x -axis lower bar labels for negative values. The default definition is:

```
\newcommand\DTLbarXneglabelalign{%  
  \ifDTLverticalbars right,rotate=-90\else left\fi  
}
```

This command is redefined by the `lower-label-style` setting.

`\DTLbarXupperlabelalign`

This command should expand to the `\pgftext` alignment specifications for the upper bar labels. The default definition is:

```
\newcommand\DTLbarXupperlabelalign{%  
  \ifDTLverticalbars bottom,center\else left\fi  
}
```

The `upper-label-align=[ $\langle$ -ve align $\rangle$ ]{ $\langle$ +ve align $\rangle$ }` option redefines this command to $\langle$ +ve align $\rangle$.

`\DTLbarXnegupperlabelalign`

This command should expand to the `\pgftext` alignment specifications for the upper bar labels. The default definition is:

```
\newcommand\DTLbarXnegupperlabelalign{%  
  \ifDTLverticalbars top,center\else right\fi  
}
```


5. Bar Charts (*databar* package)

The `upper-label-align=[ $\langle -ve align \rangle$ ]{ $\langle +ve align \rangle$ }` option redefines this command to $\langle -ve align \rangle$, if provided.

```
\DTLbarsetupperlabelalign[ $\langle -ve align \rangle$ ]{ $\langle +ve align \rangle$ }
```

This command is used by the `upper-label-align=` to redefine `\DTLbarXupperlabelalign` and optionally redefine `\DTLbarXnegupperlabelalign`.

```
\DTLbargrouplabelalign initial: \DTLbarXlabelalign
```

This command should expand to the `\pgftext` alignment specification for the group label alignment (`\DTLmultibarchart` only). The `group-label-align` option redefines this command.

```
\DTLbarYticklabelalign initial: varies
```

This command should expand to the `\pgftext` alignment specifications for the y -axis tick labels. As from v3.0, the `\ifDTLverticalbars` conditional has been moved into the definition of `\DTLbarYticklabelalign` as it is now fully expanded before being passed to the `pgf` parser. The default definition is now:

```
\newcommand\DTLbarYticklabelalign{%  
  \ifDTLverticalbars right\else top,center\fi  
}
```

This command is redefined by the `y-tick-label-align` setting.

```
\DTLbardisplayYticklabel{ $\langle text \rangle$ }
```

This command encapsulates the y -tick labels. By default, this simply expands to its argument. Note that if the tick labels are automatically generated from the data (rather than by specifying them with `y-tick-labels`) then the argument will be a datum item where the string part is the formatted number. If you prefer a plain number you can redefine `\DTLbardisplayYticklabel` to use `\datatool_datum_value:Nnnnn` (which requires $\LaTeX 3$ syntax on).

```
\DTLdisplaylowerbarlabel{ $\langle text \rangle$ }
```

This command encapsulates the lower bar labels with `\DTLbarchart`. By default, this simply expands to its argument.



```
\DTLdisplaybargrouplabel{<text>}
```

This command encapsulates the group bar labels with `\DTLmultibarchart`. By default, this simply expands to its argument.



```
\DTLdisplaylowermultibarlabel{<text>}
```

This command encapsulates the lower bar labels for `\DTLmultibarchart`. By default, this simply expands to its argument.



```
\DTLdisplayupperbarlabel{<text>}
```

This command encapsulates the upper bar labels with `\DTLbarchart`. By default, this simply expands to its argument.



```
\DTLdisplayuppermultibarlabel{<text>}
```

This command encapsulates the upper bar labels for `\DTLmultibarchart`. By default, this simply expands to its argument.

5.4.3. Bar Colours



```
\DTLbaroutlinecolor initial: black
```

This command should expand to the bar colour (suitable for use in the mandatory argument of `\color`). This command is redefined by the `outline-color` option. If this command is redefined to empty then the outline won't be drawn.



```
\DTLbaroutlinewidth initial: 0pt
```

This length register should be set to the required line thickness for the bar outline. Zero or negative values indicate that the outline should not be drawn. This register is set by the `outline-width` option.



```
\DTLsetbarcolor{<index>}{<colour>}
```

Sets the colour for the given index in the general bar colour list. This command is used by `bar-colors`, `bar-default-colors`, and `bar-default-gray`.



The argument of `\DTLsetbarcolor` may be empty. This is different to the colour not being set for the given index. An empty value indicates that the bar should not be filled. Whereas an unset value may default to white, depending on the style. Most of the time there will be no visual difference, but if there is content behind the bar, a white filled bar will obscure it but an unfilled bar won't.



`\DTLclearbarcolors`

Clears the general bar colour list. All bars will be filled white unless a new colour list is set (and therefore won't be visible without an outline).



`\DTLsetnegbarcolor{⟨index⟩}{⟨colour⟩}`

Sets the colour for the given index in the negative bar colour list. Any bars with negative values will first reference this list before using the general bar colour list.



`\DTLclearnegbarcolors`

Clears the negative bar colour list. Any bars with negative values will use the general bar colour list instead.



`\DTLgetbarcolor{⟨index⟩}`

Expands to the general bar colour specification for the given `⟨index⟩` or to `white` if not set.



`\DTLgetnegbarcolor{⟨index⟩}`

Expands to the negative bar colour specification for the given `⟨index⟩`, if set, otherwise behaves like `\DTLgetbarcolor`.



`\DTLdobarcolor[⟨value⟩]{⟨index⟩}`

Does `\color{⟨colour⟩}` where `⟨colour⟩` will be obtained from `\DTLgetnegbarcolor{⟨index⟩}` if the supplied `⟨value⟩` is negative, otherwise from `\DTLgetbarcolor{⟨index⟩}`. If the `⟨value⟩` is omitted, a non-negative number is assumed. Does nothing if an empty colour specification has been applied to the given bar index



`\DTLdocurrentbarcolor`

May be used in `\DTLeverybarhook` to do `\DTLdobarcolor{⟨index⟩}` where `⟨index⟩` is the current bar index. This command can also be used in `\DTLmapdata` or `\DTLforeach` (in which case the row index will be used as the bar index), but without access to the current bar value it will assume a non-negative value.

5.4.4. Hooks

The placeholder commands assigned in the `⟨assign-list⟩` final argument of `\DTLbarchart` and `\DTLmultibarchart` may be used in hooks to add information from the current row. Bear in mind that with `\DTLmultibarchart`, those placeholder commands will expand to the same content for every bar within the same bar group. They will also be unavailable in the start and end hooks (`\DTLbaratbegintikz` and `\DTLbaratendtikz`). Placeholder commands such as `\DTLbarvariable` and `\DTLbarvalue` are updated for every bar.



If you want to add any drawing code in the hooks, remember that the x co-ordinates are in units of a bar width and the y co-ordinates are in data units.

Since both `\DTLbarchart` and `\DTLmultibarchart` internally use `\DTLmapdata`, you can reference the current row index within hooks with `\dtlrownum`. Note that this is the database row index which may not correspond to the bar or group number if filtering has been applied. Instead, use `\DTLbarindex` for the bar index or `\DTLbargroupindex` for the bar group index. See §5.3.8.



`\DTLeverybarhook`

Hook implemented after each bar is drawn.



`\DTLeveryprebarhook`

Hook implemented before each bar is drawn.

The following three commands expand to the start point, mid point and end point of the main mid-axis through the current bar. Each point is expressed as a pgf point `\pgfpointxy`. Below, s is the starting point of the bar, $b/2$ is half a bar width and v is variable value for the current bar (see Example 145).



`\DTLstartpt`

Expands to `\pgfpointxy{s + b/2}{0}` for vertical bars and to `\pgfpointxy{0}{s + b/2}` for horizontal bars.

`\DTLmidpt`

Expands to `\pgfpointxy{s+b/2}{v/2}` for vertical bars and to `\pgfpointxy{v/2}{s+b/2}` for horizontal bars.

`\DTLendpt`

Expands to `\pgfpointxy{s+b/2}{v}` for vertical bars and to `\pgfpointxy{v}{s+b/2}` for horizontal bars.

`\DTLeverybargrouphook`

Hook implemented after every bar group is drawn with `\DTLmultibarchart`. Note that the above point placeholder commands will be set for the last bar in the group.

`\DTLbaratbegintikz`

Hook implemented at the start of the `tikzpicture` environment.

`\DTLbaratendtikz`

Hook implemented at the end of the `tikzpicture` environment.

In any of these hooks you can access the total bar chart x -axis length with:

`\DTLbarchartwidth`

This expands to the bar chart width in x -units (without any unit). To obtain the actual dimension, multiply this value by `\DTLbarwidth`. The width takes the group gap into account for multi-bar charts, but it does not take into account the space taken up by the y -ticks or y axis label (if present). Expands to nothing outside of the bar chart commands.

With `\DTLmultibarchart`, you can also access the width of each bar group:

`\DTLbargroupwidth`

This expands to the bar group width in x -units (without any unit).

`\DTLtotalbars`

This expands to the total number of bars in the current chart (and to nothing outside of the bar chart commands). For `\DTLbarchart`, this is the number of rows that contribute to the

5. Bar Charts (*databar package*)

chart (which will be less than or equal to the total number of rows in the database). For `\DTLmultibarchart`, this will be the number of contributing rows multiplied by the number of bars in each group.

`\DTLbartotalvariables`

For `\DTLmultibarchart` only, this expands to the number of variables (that is, the number of bars in each group). Expands to nothing outside of `\DTLmultibarchart`.

`\DTLtotalbargroups`

For `\DTLmultibarchart` only, this expands to the total number of bar groups in the current chart. Expands to nothing outside of `\DTLmultibarchart`. The number of bar groups is equal to the number of rows that contribute to the chart (which will be less than or equal to the total number of rows in the database).

6. Scatter and Line Plots (dataplot package)

```
\usepackage[<options>]{dataplot}
```

The dataplot package can be used to draw scatter or line diagrams obtained from columns in one or more databases. This package automatically loads the datatool package.

Any package options provided when loading dataplot will be passed to datatool. If datatool has already been loaded, any options will be passed to `\DTLsetup` instead (which means that you would only be able to use options that can be set after datatool has been loaded).

The dataplot package additionally loads the tikz package and also the tikz plot libraries plotmarks, plohandlers and calc.

The dataplot package was rewritten in version 3.0 to use L^AT_EX3 commands. The xkeyval package has been dropped and the use of `\DTLforeach` has been replaced with `\DTLmapdata`. A number of bugs have also been fixed, which may cause some differences in the output.

Rollback to version 2.32 is available:

```
\usepackage{dataplot}[=2.32]
```

Note that if datatool hasn't already been loaded, this will also apply rollback to datatool. Problems may occur if a newer release of datatool has already been loaded.

The principle command provided by dataplot is:

```
\DTLplot[<condition>]{<db-list>}{<key=value list>}
```

This draws data from the listed databases as a scatter plot or line diagram. The *<db-list>* argument should be a comma-separated list of database labels. The *<key=value list>* argument should be a *<key>=<value>* list of settings. Available settings are listed in §6.1. The x and y settings are required. The other settings are optional.

6. Scatter and Line Plots (*dataplot* package)

The *<condition>* optional argument allows filtering to be applied. This should be in the syntax allowed for the first argument of `\ifthenelse` (see §2.4.2). Note that if this option is provided, it will override the `include-if` or `include-if-fn` setting.

The *x* and *y* settings should be a comma-separated list of column keys identifying the columns to use for the *x* and *y* co-ordinates. They must both have at least one element, and the number of elements in *x* must be less than or equal to the number of elements in *y*. Spaces and empty items are skipped. For example, `y={Exp1,, Exp2 ,}` is equivalent to `y={Exp1,Exp2}`. However, `y={Exp1,{ },Exp2}` specifies an empty key in the second item which is invalid.

There is also a corresponding action:

```
\DTLaction[<settings>] {plot}
```

The *name* should be set to the database name or list of names *<db-list>*, and *options* should be the *<key=value list>* plot settings. If the *name* is omitted, the default database is assumed.

```
\DTLaction[name={<db-list>}, options={<key=value list>}] {plot}
```

The above is equivalent to:

```
\DTLplot{<db-list>}{<key=value list>}
```

Note that you can't use any of the usual action column identifiers, such as *keys*, as they need to be separated into the *x* and *y* lists.

The primary return value is the total number of plot streams. This will be 0 if an error occurred. The secondary return values will only be set if no errors occurred while parsing the settings:

- *min-x*: the value of `\DTLminX` within the plot;
- *min-y*: the value of `\DTLminY` within the plot;
- *max-x*: the value of `\DTLmaxX` within the plot;
- *max-y*: the value of `\DTLmaxY` within the plot;
- *stream-count*: the total number of plot streams (same as the primary return value);
- *x-count*: the total number of *x* column keys;
- *y-count*: the total number of *y* column keys;
- *name-count*: the total number of databases provided in the *name* option.

For both `\DTLplot` and the `plot` action, each plot stream is constructed in the following order:

1. For each database listed in *<db-list>*:

2. If the `group-styles` setting indicates that the marker or line styles or colours should apply to each database, the next style in the sequence will be selected, as applicable. Otherwise, if the corresponding `style-resets` option is on, the setting will be reset back to the applicable `marks`, `lines`, `mark-colors` or `line-colors` value.
 - a) For each $\langle y\text{-key} \rangle$ in y :
 - i. If the $\langle \text{pending-}x \rangle$ list is empty, it's repopulated with the x list of column keys. (This means you can have a single x column plotted against multiple y columns. However, be careful if you have more than one element in the x list but less than the total number of elements in the y list, see Example 155.)
 - ii. The first $\langle x\text{-key} \rangle$ is popped off the $\langle \text{pending-}x \rangle$ list.
 - iii. A plot stream is constructed from the (x, y) co-ordinates supplied by the $\langle x\text{-key} \rangle$ and $\langle y\text{-key} \rangle$ columns, omitting any that don't match the provided conditional or filter function and any that lie outside the plot bounds.
 - iv. If the `group-styles` setting indicates that the marker or line styles or colours should be applied to each stream, the next setting in the sequence is selected. (See the examples in §6.2.3.)
 - v. The stream will be drawn with lines, if applicable.
 - vi. The stream will be drawn with markers, if applicable.
 - vii. If applicable, the stream will be added to the legend with the label obtained from the corresponding item in the `legend-labels` option or the default label, if omitted. Note that an empty label will cause the corresponding plot stream to be omitted from the legend, but bear in mind that the $\langle \text{key} \rangle = \langle \text{value} \rangle$ list parser will strip empty items, so you will need to explicitly group the empty value, if applicable. See §6.2.2 for examples.

The bounding box, grid, axes, tick marks, tick labels and axis labels are drawn first, if applicable. The legend is drawn last, if applicable. The hook `\DTLplotatbegintikz` occurs at the start, and `\DTLplotatendtikz` at the end.

6.1. Plot Settings

The plot settings may be provided in the final $\langle \text{settings} \rangle$ argument of `\DTLplot` or the `options` value for the `plot` action or set with the `plot` option in `\DTLsetup`. For example, to set defaults:

```
\DTLsetup{plot={grid,tick-dir=in}}
```

To override the defaults for a specific plot:

```
\DTLplot{mydata}{x=Time,y=Temperature,legend}
```

Alternatively, using the `plot` action:

```
\DTLaction[
  namemydata,
  options={x=Time,y=Temperature,legend}
]{plot}
```

6.1.1. Database Content

`x` = { *<key-list>* }

The list of column keys to use for *x* co-ordinates. This setting is required. There must be at least one item and no more than the total number of items in *y*.

`y` = { *<key-list>* }

The list of column keys to use for *y* co-ordinates. This setting is required. There must be at least one item.

`extra-assign` = { *<assign-list>* }

If set, the *<assign-list>* will be passed to `\DTLmapgetvalues` when `\DTLplot` loops through each row of the database using `\DTLmapdata`. The placeholder commands identified in the assignment list can be used in the filter condition. Alternatively, you can simply use `\DTLmapget` in the filter function `include-if` or `include-if-fn` (but not in the optional *<condition>* argument of `\DTLplot` which needs to expand).

`include-if` = *<definition>*

The value should be the definition (expansion text) for a command that takes a single argument. The command should do #1 for any row that should have its *x* and *y* values included in the plot, and do nothing otherwise.

include-if-fn= $\langle cs \rangle$

An alternative to `include-if`, this sets the filter function to $\langle cs \rangle$ rather than providing the function definition.

The `include-if` and `include-if-fn` override each other, but both will be ignored if the $\langle condition \rangle$ optional argument is provided with `\DTLplot`. Either use one form or the other. Don't use both.

For example, suppose the database “results” has columns with labels “Time”, “Temperature” and “Notes” where the notes column is set to “Incorrect” for rows where the measurement was incorrect. These rows can be excluded:

```
\DTLplot{results}
{
  x=Time,
  y=Temperature,
  extra-assign={\Notes=Notes},
  include-if={
    \DTLifstringeq{\Notes}{Incorrect}{}{#1}
  }
}
```

This uses `\DTLifstringeq`, which expands its arguments, to test value of the placeholder command. Note that while you can use `etoolbox`'s `\ifdefstring` with the default `store-datum=false`, it won't work with `store-datum=true` as then `\Notes` will be a datum control sequence.

6.1.2. Plot Size

width= $\langle dimension \rangle$

The total width of the plot.

height= $\langle dimension \rangle$

The total height of the plot.

bounds= $\langle value \rangle$ *initial: empty*

The plot bounds. If not set, the bounds will be determined by the data. Instead of setting all the bounds in one go, you can also set them individually using the following options. Any missing options will be calculated from the data.

min-x= $\langle value \rangle$ The lower x bound.**minx**= $\langle value \rangle$ *alias: min-x; initial: empty*

A synonym of min-x.

min-y= $\langle value \rangle$ *initial: empty*The lower y bound.**miny**= $\langle value \rangle$ *alias: min-y*

A synonym of min-y.

max-x= $\langle value \rangle$ *initial: empty*The upper x bound.**maxx**= $\langle value \rangle$ *alias: max-x*

A synonym of max-x.

max-y= $\langle value \rangle$ *initial: empty*The upper x bound.**maxy**= $\langle value \rangle$ *alias: max-y*

A synonym of max-y.

6.1.3. Marker and Line Styles

Note that the $\langle key \rangle = \langle value \rangle$ parser strips leading and trailing spaces and empty items so for example, `blue, red, , green,` is converted to `blue, red, green`. This means that if you explicitly need an empty item, you must group it. For example: `blue, red, {}, green`.

The default colour indicates the default for `tikzpicture`. This can be changed in the `\DTLplotatbegintikz` hook.

legend= $\langle value \rangle$

default: northeast; initial: none

Indicates where to draw the legend. The value may be one of: `none` (no legend), `north`, `northeast`, `east`, `southeast`, `south`, `southwest`, `west`, `northwest` or `custom`. If you use `legend=custom`, you will need to redefine `\DTLcustomlegend` to position the legend in the required place.

legend-labels= $\{ \langle list \rangle \}$

initial: empty

Comma-separated list of text to supply for each row of the plot legend. If omitted, the legend text will be formed from the database name or column key, depending on how many databases or columns have been selected for the plot. Note that an empty label will cause the corresponding plot stream to be omitted from the legend, but bear in mind that the $\langle key \rangle = \langle value \rangle$ list parser will strip empty items, so you will need to explicitly group the empty value, if applicable. For example, in the following:

```
legendlabels={Experiment1, , Experiment2}
```

the empty item will be stripped by the parser and it will be equivalent to:

```
legendlabels={Experiment1, Experiment2}
```

However, with the following:

```
legendlabels={Experiment1, {}, Experiment2}
```

The first and third plot streams will be added to the legend but the second won't.

legendlabels= $\{ \langle list \rangle \}$

alias: legend-labels; initial: empty

A synonym of `legend-labels`.

legend-offset= $\{ \langle x-dim \rangle, \langle y-dim \rangle \}$

The legend offset may be a single value, in which case it applies to both x and y offsets, or two

values: the x offset and y offset.

style= $\langle value \rangle$

initial: **markers**

Indicates whether to show markers, draw lines or both. Available values: **markers** (only markers, no lines), **lines** (only lines, no markers), or **both** (lines and markers).

If you want some plot streams within the same `\DTLplot` to have markers only and some to have lines only, then use `style=both` and use `\relax` or `{ }` for the corresponding item in `marks` or `lines`. For example:

```
style=both,
lines={
  \pgfsetdash{{0pt}}{0pt},% solid line
  \relax,% no line
  \pgfsetdash{{10pt}}{5pt}}{0pt},% dashed
  \pgfsetdash{{5pt}}{5pt}}{0pt},% dashed
  \relax,% no line
  \pgfsetdash{{5pt}}{5pt}}{1pt}}{5pt}}{0pt},% dash dot
},
marks={
  \relax,% no marker
  \pgfuseplotmark{o},% circle marker
  \pgfuseplotmark{x},% cross marker
  \relax,% no marker
  \pgfuseplotmark{+},% plus marker
  \pgfuseplotmark{asterisk},% asterisk marker
}
```

style-resets= $\{ \langle settings \rangle \}$

default: **all**; *initial:* **none**

If `group-styles` setting isn't on for a particular style (line style, line colour, marker style or marker colour), then the style will either cycle through the corresponding list, wrapping round whenever the end of the list is reached, or will reset at the start of each database, according to the `style-resets` setting.

This is a multiple choice setting so you can combine allowed values in $\langle settings \rangle$.

style-resets=none

None of the styles will be reset at the start of each database.

style-resets=all

All of the styles will be reset at the start of each database unless the `group-styles` setting applies.

This setting is equivalent to `style-resets={mark-style, mark-color, line-style, line-color}`.

style-resets=mark-style

The plot marks listed in `marks` will be reset at the start of each database, unless `mark-style` has been set.

style-resets=mark-color

The plot mark colours listed in `mark-colors` will be reset at the start of each database, unless `mark-color` has been set.

style-resets=line-style

The line styles listed in `lines` will be reset at the start of each database, unless `line-style` has been set.

style-resets=line-color

The line colours listed in `line-colors` will be reset at the start of each database, unless `line-color` has been set.

group-styles={ *<settings>* }

default: all; initial: none

Indicates whether each item the marker and line styles should be the same for each database.

This is a multiple choice setting so you can combine allowed values in *<settings>*.

group-styles=none

None of the styles are applied per-database. Each style setting (`marks`, `mark-colors`, `lines` and `line-colors`) will be reset or cycled round for each plot stream according to the `style-resets` setting.

group-styles=all

All of the styles are applied per-database. Multiple plot streams from a single database will have

the same styles.

This setting is equivalent to `group-styles={mark-style, mark-color, line-style, line-color}`.

group-styles=line-style

The line style will be applied per-database. Multiple plot streams from a single database will use the same line style.

group-styles=line-color

The line colour will be applied per-database. Multiple plot streams from a single database will use the same line colour.

group-styles=mark-style

The marker style will be applied per-database. Multiple plot streams from a single database will use the same marker.

group-styles=mark-color

The marker colour will be applied per-database. Multiple plot streams from a single database will use the same marker colour.

colors={ <colour-list> }

Equivalent to `line-colors=<colour-list>, mark-colors=<colour-list>`.

line-colors={ <colour-list> }

The value should be a comma-separated list of colour names (suitable for use in the argument of `\color`) to apply to the corresponding plot line style given in `lines`. An empty item `{ }` or `\relax` indicates that no colour change should apply to that line (so the line will be black unless the default colour is changed). This setting simply redefines `\DTLplotlinecolors` to the supplied value.

This setting has no effect if no plot lines should be shown (`style=markers`).

linecolors={ <colour-list> }

alias: **line-colors**

A synonym of `line-colors`.



```
lines={\line style list}
```

The value should be a comma-separated list of line styles (specified with `\pgfsetdash`). An empty item `{}` or `\relax` indicates that lines should be omitted for the corresponding plot stream. This setting simply redefines `\DTLplotlines` to the supplied value.

This setting has no effect if no plot lines should be shown (`style=markers`).



```
mark-colors={\colour-list}
```

The value should be a comma-separated list of colour names (suitable for use in the argument of `\color`) to apply to the corresponding plot mark given in `marks`. An empty item `{}` or `\relax` indicates that no colour change should apply to that marker (so the marker will be black unless the default colour is changed). This setting simply redefines `\DTLplotmarkcolors` to the supplied value.

This setting has no effect if no plot markers should be shown (`style=lines`).



```
markcolors={\colour-list}
```

alias: **mark-colors**

A synonym of `mark-colors`.



```
marks={\mark-list}
```

The value should be a comma-separated list of plot markers (specified with `\pgfuseplotmark`). An empty item `{}` or `\relax` indicates that markers should be omitted for the corresponding plot stream. This setting simply redefines `\DTLplotmarks` to the supplied value.

This setting has no effect if no plot markers should be shown (`style=lines`).

6.1.4. Axes

By default, the tick marks will be automatically generated from the data bounds (which will either be determined from the maximum and minimum values in the data or by options such as `bounds`). If no gap is explicitly set (with `x-tick-gap` or `y-tick-gap`) then the gap between major tick marks is calculated according to an algorithm that's based on the maximum and minimum values and the minimum gap length (as an absolute dimension) given by `\DTLmintickgap`. There is a similar length `\DTLminminortickgap` for the suggested minimum distance between minor tick marks.

If an exact set of the major tick marks is required, the desired co-ordinates can be set with `x-tick-points` (for the *x*-axis) and `y-tick-points` (for the *y*-axis). These should be a comma-separated list of plain numbers in data units. This will override the above tick mark algorithm and the gap setting will be ignored. The minor minimum distance will still be referenced if minor tick marks are required.

6. Scatter and Line Plots (*dataplot* package)

The tick mark labels will then be obtained from the tick mark values and converted to decimal datum items, where the string part is the formatted numbers corresponding to the tick mark value (with rounding applied according to the applicable rounding setting).

The x -tick mark labels will be encapsulated with `\DTLplotdisplayXticklabel`, and the y -tick mark labels will be encapsulated with `\DTLplotdisplayYticklabel`. Both these commands default to using `\DTLplotdisplayticklabel` so only that command needs redefining if the same formatting should be used for both x and y tick labels.



If you override the tick mark labels with `x-tick-labels` or `y-tick-labels` then the tick labels will no longer be datum items, but will be the corresponding label in the supplied setting. The tick label will still be encapsulated with the corresponding formatting command.



axes=*<value>*

initial: **both**

Determines whether or not to display the axes.



axes=both

Switches on x and y axes and x and y ticks.



axes=none

Switches off x and y axes and x and y ticks.



axes=x

Switches on x axis and x ticks, and switches off y axis and y ticks.



axes=y

Switches on y axis and y ticks, and switches off x axis and x ticks.



omit-zero-label=*<value>*

default: **both**; *initial:* **auto**

If either axis includes a tick mark at zero, this option determines whether or not that tick mark should have a label.



If you explicitly set the tick labels and this option is set to omit zero, then the tick label corresponding to zero will be ignored, but must still be set to prevent the labels from shifting out of sync.



omit-zero-label=auto

Determines whether or not to show the zero tick labels based on whether the axes cross at zero. Note that this doesn't take the axis extensions into account.



omit-zero-label=both

Omit the tick labels at $x = 0$ and $y = 0$.



omit-zero-label=x

Omit the tick label at $x = 0$. If there happens to be a tick mark at $y = 0$, the $y = 0$ label will be shown.



omit-zero-label=y

Omit the tick label at $y = 0$. If there happens to be a tick mark at $x = 0$, the $x = 0$ label will be shown.



omit-zero-label=false

Don't omit the tick label at $x = 0$ and $y = 0$, if there are tick marks at those locations.



side-axes=*<boolean>*

initial: **false**

If true, the axes will be drawn at the minimum plot bounds (not taking the axis extension into account). If false, the axes will intersect at zero if that is within the plot bounds, otherwise the axes will be drawn at the minimum plot bounds.

If zero doesn't lie within the plot bounds, then setting will only make a difference if you have the box and tick marks. With `side-axes` true, tick marks will be drawn along the top and right sides of the box. This will only be noticeable if you have extended the axes. With `side-axes` false, tick marks will be drawn along all sides of the box.



x-axis-style=*<value>*

initial: **-**

Sets the line style for the x -axis. The given *<value>* should be valid tikz options. For example, `x-axis-style={->, thick}` for a thick line with an arrow head.

y-axis-style= $\langle value \rangle$

initial: -

Sets the line style for the y -axis. The given $\langle value \rangle$ should be valid tikz options.

axis-style= $\langle value \rangle$

initial: -

Shortcut that sets the line style for both the x -axis and y -axis. This is equivalent to `x-axis-style={ $\langle value \rangle$ }, y-axis-style={ $\langle value \rangle$ }`.

extend-x-axis= $\{ \langle lower extent \rangle, \langle upper extent \rangle \}$

initial: 0

Extends the x axis by the given values beyond the plot bounds. If only one value is provided, the axis is extended by the same amount at both ends. Otherwise, the axis is extended to the left by $\langle lower extent \rangle$ and to the right by $\langle upper extent \rangle$.

This option affects the length of the x -axis, the location of the lower and upper x labels, and the size of the encapsulating box if `box=true`. The values should be in data co-ordinates. A positive value extends the axis. A negative value shortens it.

If you shorten the axis with the tick mark setting on, this may result in detached tick marks beyond the shortened axis. The x -axis length will end up less than `\DTLplotwidth` (set with `width`).

Extending the axis may result in it overlapping a tick label. The x -axis length will end up greater than `\DTLplotwidth`.

extend-y-axis= $\{ \langle lower extent \rangle, \langle upper extent \rangle \}$

initial: 0

Extends the y axis by the given values beyond the plot bounds. If only one value is provided, the axis is extended by the same amount at both ends. Otherwise, the axis is extended to the left by $\langle lower extent \rangle$ and to the right by $\langle upper extent \rangle$.

This option affects the length of the y -axis, the location of the lower and upper y labels, and the size of the encapsulating box if `box=true`. The values should be in data co-ordinates. A positive value extends the axis. A negative value shortens it.

If you shorten the axis with the tick mark setting on, this may result in detached tick marks beyond the shortened axis. The y -axis length will end up less than `\DTLplotheight` (set with `height`).

Extending the axis may result in it overlapping a tick label. The y -axis length will end

up greater than `\DTLplotheight`.

extend-axes= $\{ \langle \text{lower extent} \rangle, \langle \text{upper extent} \rangle \}$

initial: 0

Equivalent to:

`extend-x-axis={\langle lower extent \rangle, \langle upper extent \rangle},`
`extend-y-axis={\langle lower extent \rangle, \langle upper extent \rangle}`

Again a single value may be supplied if the lower and upper extent are the same.

box= $\langle \text{boolean} \rangle$

initial: **false**

If true, the plot will be enclosed in a box. This takes the axis extensions into account.

box-ticks= $\langle \text{value} \rangle$

default: **match-axes**; initial: **match-axes**

If `box=true`, this setting determines which direction to draw the tick marks.

If tick marks are drawn, they don't take the axis extensions into account. This can lead to detached tick marks outside of the box if the axes have been shortened.

box-ticks=**none**

Don't draw tick marks on the box.

box-ticks=**match-axes**

Match the tick direction used on the axes. That is, if the axis tick marks point inwards then so will the tick marks on the box.

box-ticks=**in**

The tick marks on the box will point inwards.

box-ticks=**out**

The tick marks on the box will point outwards.

grid= $\langle\text{boolean}\rangle$ *initial:* **false**

If true, the plot will have a grid in the background. The minor grid lines will only be drawn if the corresponding minor tick mark setting is also on and `\DTLminorgridstyle` (minor-grid-style) has a non-empty definition.

major-grid-style= $\{ \langle\text{value}\rangle \}$ *initial:* **color=gray, -**

Sets the style for the major grid lines. The given $\langle\text{value}\rangle$ should be valid tikz options. This option simply redefines `\DTLmajorgridstyle` to the given $\langle\text{value}\rangle$.

minor-grid-style= $\{ \langle\text{value}\rangle \}$
thin*initial:* **color=lightgray, very**

Sets the style for the minor grid lines. The given $\langle\text{value}\rangle$ should be valid tikz options. This option simply redefines `\DTLminorgridstyle` to the given $\langle\text{value}\rangle$.

xlabel= $\langle\text{value}\rangle$ *initial:* **empty**

Sets the mid x -axis label. This label is positioned mid-way (using the plot bounds not including the axis extension) along the x -axis, below the tick labels.

x-label= $\langle\text{value}\rangle$ *alias:* **xlabel**

Synonym of `xlabel`.

ylabel= $\langle\text{value}\rangle$ *initial:* **empty**

Sets the mid y -axis label. This label is positioned mid-way (using the plot bounds not including the axis extension) along the y -axis, to the left of the tick labels.

y-label= $\langle\text{value}\rangle$ *alias:* **ylabel**

Synonym of `ylabel`.

min-x-label= $\langle\text{value}\rangle$ *initial:* **empty**

Sets the text for the lower x -axis label. This is positioned at the minimum end of the x -axis, taking the axis extension into account.

min-x-label-style= $\langle value \rangle$

initial: empty

Sets the tikz node style for the lower x -axis label.

max-x-label= $\langle value \rangle$

initial: empty

Sets the text for the upper x -axis label. This is positioned at the maximum end of the x -axis, taking the axis extension into account.

max-x-label-style= $\langle value \rangle$

initial: empty

Sets the tikz node style for the upper x -axis label.

min-y-label= $\langle value \rangle$

initial: empty

Sets the text for the lower y -axis label. This is positioned at the minimum end of the y -axis, taking the axis extension into account.

min-y-label-style= $\langle value \rangle$

initial: empty

Sets the tikz node style for the lower y -axis label.

max-y-label= $\langle value \rangle$

initial: empty

Sets the text for the upper y -axis label. This is positioned at the maximum end of the y -axis, taking the axis extension into account.

max-y-label-style= $\langle value \rangle$

initial: empty

Sets the tikz node style for the upper y -axis label.

round= $\langle n \rangle$

initial: 2

Equivalent to `round-x= $\langle n \rangle$ , round-y= $\langle n \rangle$ .`

round-x= $\langle n \rangle$

initial: 2

Sets the number of digits to round the x tick labels (when no label provide with `x-tick-labels`).

round-y= $\langle n \rangle$ *initial: 2*

Sets the number of digits to round the y tick labels (when no label provide with `y-tick-labels`).

ticks= $\langle \text{boolean} \rangle$

A shortcut for `x-ticks= $\langle \text{boolean} \rangle$ , y-ticks= $\langle \text{boolean} \rangle$` .

tics= $\langle \text{boolean} \rangle$ *alias: ticks*

Synonym for `ticks`.

minor-ticks= $\langle \text{boolean} \rangle$

A shortcut for `x-minor-ticks= $\langle \text{boolean} \rangle$ , y-minor-ticks= $\langle \text{boolean} \rangle$` .

minorticks= $\langle \text{boolean} \rangle$ *alias: minor-ticks*

Synonym for `minor-ticks`.

tick-label-style= $\{ \langle \text{value} \rangle \}$ *initial: empty*

A shortcut for:

```
x-tick-label-style={ $\langle \text{value} \rangle$ },
y-tick-label-style={anchor= east,  $\langle \text{value} \rangle$ }
```

tic-label-style= $\{ \langle \text{value} \rangle \}$ *alias: tick-label-style; initial: empty*

Synonym of `tick-label-style`.

tick-label-offset= $\langle \text{dimension} \rangle$ *initial: empty*

Sets the offset for the tick labels. This option simply changes the value of the length register `\DTLticklabeloffset`.

tick-dir= $\langle \text{value} \rangle$

A shortcut for `x-tick-dir= $\langle \text{value} \rangle$ , y-tick-dir= $\langle \text{value} \rangle$` .

ticdir= $\langle value \rangle$

alias: **tick-dir**; initial: **in**

Synonym for **tick-dir**.

tick-gap= $\langle value \rangle$

initial: **empty**

A shortcut for **x-tick-gap**= $\langle value \rangle$, **y-tick-gap**= $\langle value \rangle$.

ticgap= $\langle value \rangle$

alias: **tick-gap**

Synonym for **tick-gap**.

x-ticks= $\langle boolean \rangle$

initial: **true**

Indicates whether or not to draw x ticks.

xtics= $\langle boolean \rangle$

alias: **x-ticks**

Synonym for **x-ticks**.

x-tick-label-style= $\langle value \rangle$

initial: **empty**

Sets the tikz node style for the x -axis tick labels.

x-tic-label-style= $\langle value \rangle$ alias: **x-tick-label-style**; initial: **empty**

Synonym for **x-tick-label-style**.

x-minor-ticks= $\langle boolean \rangle$

initial: **false**

Indicates whether or not to draw minor x ticks. Note that **x-minor-ticks**=**true** also implements **x-ticks**=**true**, but **x-minor-ticks**=**false** doesn't alter the **x-ticks** setting.

xminortics= $\langle boolean \rangle$

alias: **x-minor-ticks**

Synonym for **x-minor-ticks**.

x-tick-dir=*<value>**initial: in*

Indicates whether the x ticks should be drawn inwards (`x-tick-dir=in`) or outwards (`x-tick-dir=out`).

xticdir=*<value>**alias: x-tick-dir; initial: in*

Synonym for `x-tick-dir`.

x-tick-gap=*<value>**initial: empty*

Sets the gap (in data co-ordinates) for the x -tick marks. If the value is not empty, this option will automatically enable x -ticks and the x -axis. Note that `x-tick-points` overrides this option.

xticgap=*<value>**alias: x-tick-gap*

Synonym for `x-tick-gap`.

x-tick-labels=*{ <list> }**initial: empty*

A list of labels for the x -ticks. If the value is not empty, this option will automatically enable x ticks and the x -axis. Note that the parser will automatically trim spaces and remove empty items, so if you want some tick marks to be unlabelled then you will need an empty group. For example, `x-tick-labels={0, {}, 2, {}, 4}`.

xticlabels=*{ <list> }**alias: x-tick-labels*

Synonym for `x-tick-labels`.

x-tick-points=*{ <list> }**initial: empty*

A comma-separated list of plain numbers in data co-ordinates for the x tick marks. If the value is non-empty, this option will automatically enable x -ticks and the x -axis and will override the `x-tick-gap` setting.

Any tick marks in `x-tick-points` that are outside of the plot bounds (`minx` and `maxx`) will be omitted.

xticpoints= $\langle list \rangle$

alias: **x-tick-points**

Synonym for x-tick-points.

y-ticks= $\langle boolean \rangle$

initial: **true**

Indicates whether or not to draw y ticks.

yticks= $\langle boolean \rangle$

alias: **y-ticks**

Synonym for y-ticks.

y-tick-label-style= $\langle value \rangle$

initial: **anchor=east**

Sets the tikz node style for the y -axis tick labels.

y-tic-label-style= $\langle value \rangle$ alias: **y-tick-label-style**; initial: **empty**

Synonym for y-tick-label-style.

y-minor-ticks= $\langle boolean \rangle$

initial: **false**

Indicates whether or not to draw minor y ticks. Note that y-minor-ticks=true also implements y-ticks=true, but y-minor-ticks=false doesn't alter the y-ticks setting.

yminorticks= $\langle boolean \rangle$

alias: **y-minor-ticks**

Synonym for y-minor-ticks.

y-tick-dir= $\langle value \rangle$

initial: **in**

Indicates whether the y ticks should be drawn inwards (y-tick-dir=in) or outwards (y-tick-dir=out).

yticdir= $\langle value \rangle$

alias: **y-tick-dir**; initial: **in**

Synonym for y-tick-dir.

y-tick-gap=*<value>**initial: empty*

Sets the gap (in data co-ordinates) for the *y*-tick marks. If the value is not empty, this option will automatically enable *y* tick marks and the *y*-axis. Note that `y-tick-points` overrides this option.

yticgap=*<value>**alias: y-tick-gap*

Synonym for `y-tick-gap`.

y-tick-labels=*{ <list> }**initial: empty*

A list of labels for the *y*-ticks. If the value is not empty, this option will automatically enable *y* tick marks and the *y*-axis. Note that the parser will automatically trim spaces and remove empty items, so if you want some tick marks to be unlabelled then you will need an empty group. For example, `y-tick-labels={0, {}, 2, {}, 4}`.

yticlabels=*{ <list> }**alias: y-tick-labels*

Synonym for `y-tick-labels`.

y-tick-points=*{ <list> }**initial: empty*

A comma-separated list of plain numbers in data co-ordinates for the *y* tick marks. If the value is non-empty, this option will automatically set enable *y*-tick marks and the *y*-axis and will override the `y-tick-gap` setting.

Any tick marks in `y-tick-points` that are outside of the plot bounds (`miny` and `maxy`) will be omitted.

yticpoints=*{ <list> }**alias: y-tick-points*

Synonym for `y-tick-points`.

6.2. Plot Examples

The examples in this section use the “time to growth” data, described in §§3.2.9 & 3.2.10, and the “xydata” database, described in §3.2.11.

The time to growth data represents hypothetical microbiological experiments observing microbial populations after certain time intervals at a particular temperature. The population figures are actually recorded as the log count, rather than the actual count. The *dataplot* package does support logarithmic axes. The sample data is simply a list of numbers to demonstrate how to use `\DTLplot`.

6.2.1. Basic Examples

The examples in this section demonstrate basic use. They set the x -axis label with `x-label` and the y -axis label with `y-label`. The width and height are also set to produce a smaller image than the default setting.

For examples demonstrating how to adjust the legend text or move it to a different position, see §6.2.2. For examples demonstrating how to have line plots or change the colours, see §6.2.3.

6.2.1.1. One Database

Example 149 plots the data from the “growth1” database (see §3.2.9). The horizontal x -axis corresponds to the `Time` column. The vertical y -axis corresponds to the `Experiment 1` and `Experiment 2` columns which has the observations at the given point in time. The legend setting helps to distinguish between the two data streams.

149

```
\DTLplot{growth1}{
  x=Time, y={Experiment 1,Experiment 2},
  x-label={Time ($t$)}, y-label={Log Count},
  legend, width=2.5in, height=2.5in
}
```

6.2.1.2. Two Databases, One X Column

Example 150 modifies Example 149 to include the “growth2” data as well.

150

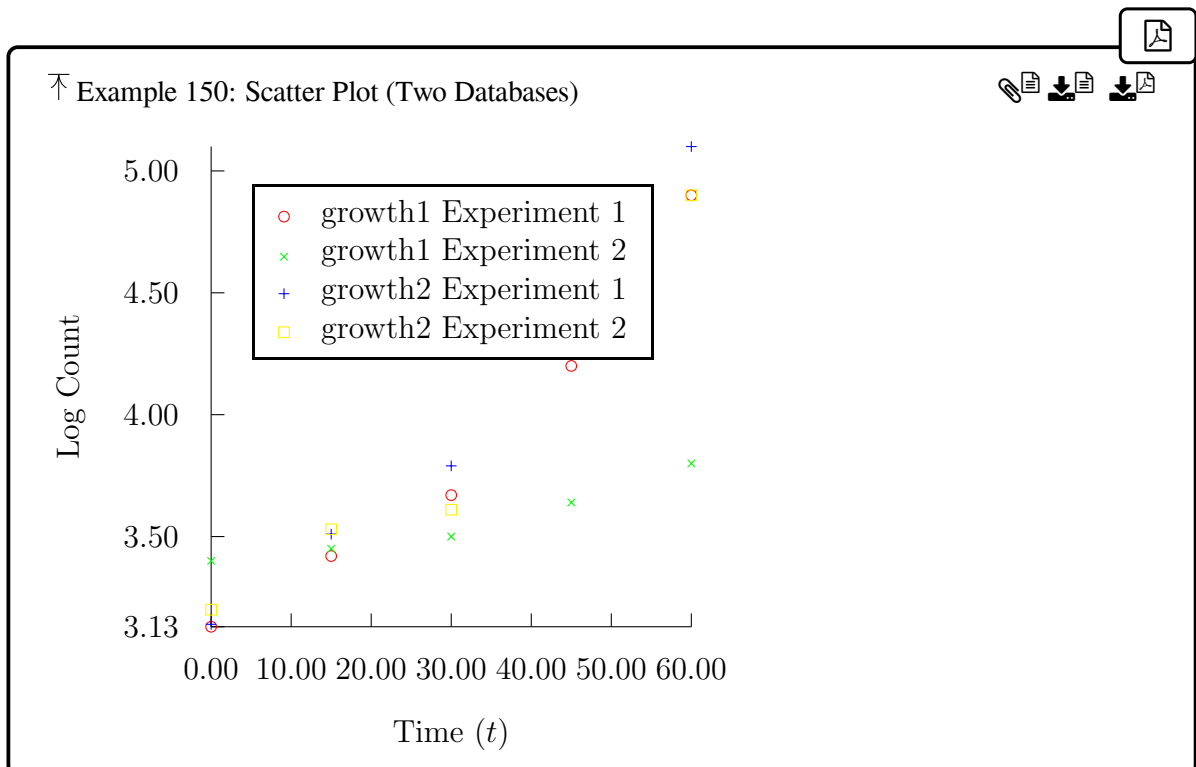
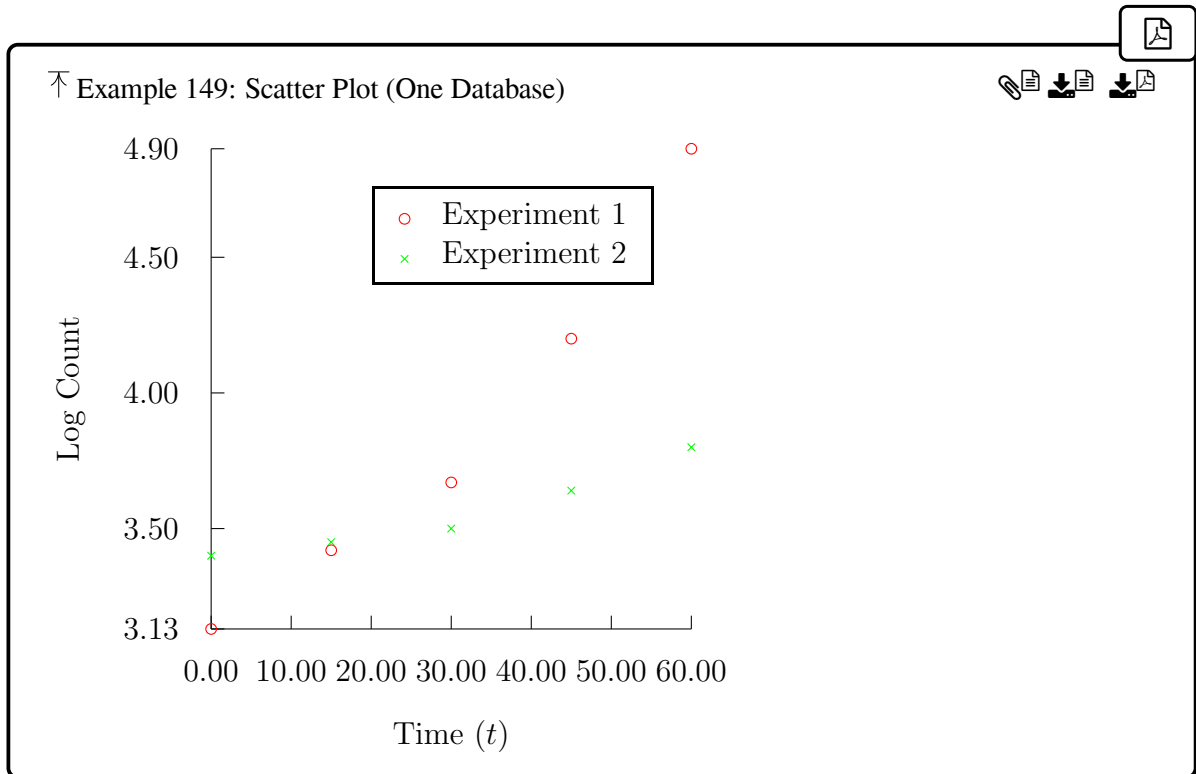
```
\DTLplot{growth1,growth2}{
  x=Time, y={Experiment 1,Experiment 2},
  x-label={Time ($t$)}, y-label={Log Count},
  legend, width=2.5in, height=2.5in
}
```

6.2.1.3. Plot Action

Example 151 is an alternative to Example 150 that uses the `plot` action instead of explicitly using `\DTLplot`.

151

6. Scatter and Line Plots (*dataplot* package)



```
\DTLaction[name={growth1,growth2},
options={
  x=Time, y={Experiment 1,Experiment 2},
  x-label={Time ($t$)}, y-label={Log Count},
  legend, width=2.5in, height=2.5in
}
]{plot}
```

The return values can then be accessed:

```
Number of streams: \DTLuse{stream-count}.
Minimum X: \DTLuse{min-x}.
Minimum Y: \DTLuse{min-y}.
Maximum X: \DTLuse{max-x}.
Maximum Y: \DTLuse{max-y}.
```

Alternatively, you can use the `return` setting:

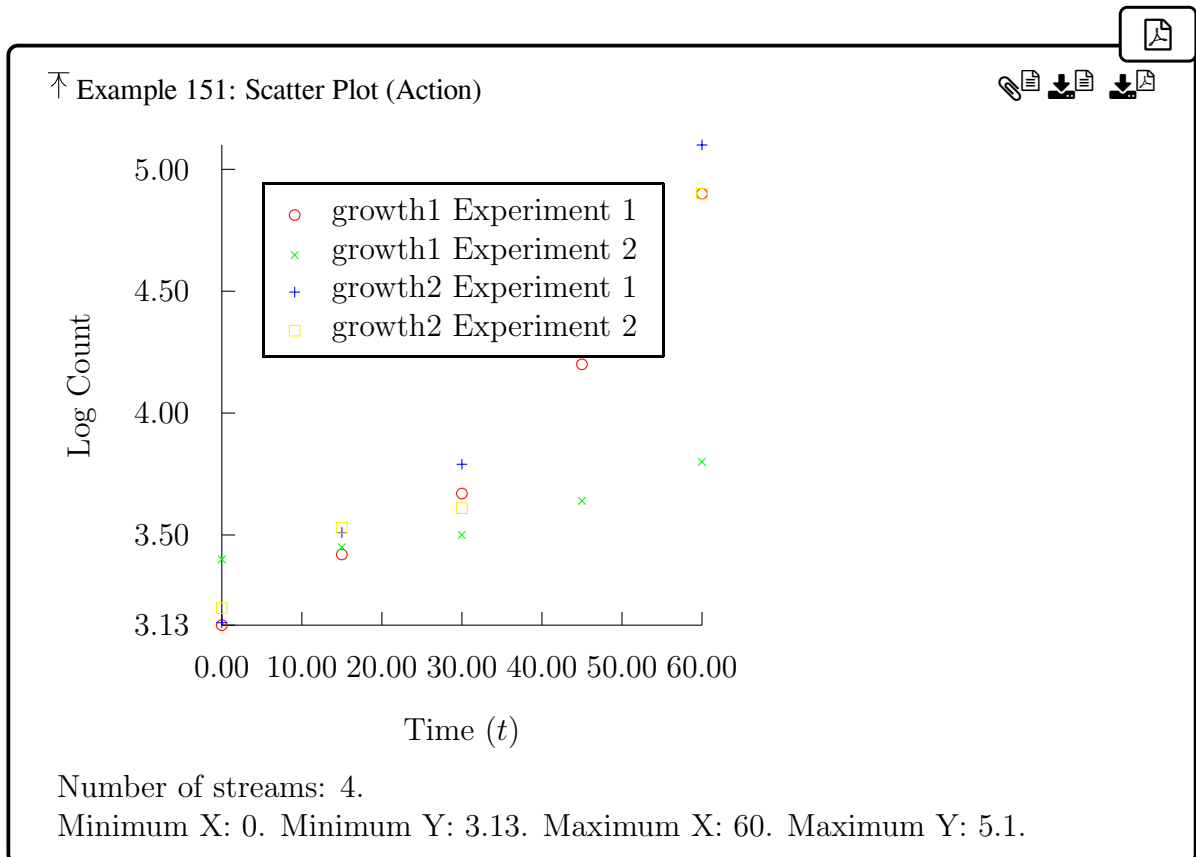
```
\DTLaction[name={growth1,growth2},
options={
  x=Time,
  y={Experiment 1,Experiment 2},
  x-label={Time ($t$)}, y-label={Log Count},
  legend, width=2.5in, height=2.5in
},
return={
  \theMinX=min-x, \theMinY=min-y,
  \theMaxX=max-x, \theMaxY=max-y,
  \NumStreams=stream-count
}
]{plot}
```

```
Number of streams: \NumStreams.
Minimum X: \theMinX. Minimum Y: \theMinY.
Maximum X: \theMaxX. Maximum Y: \theMaxY.
```

6.2.1.4. One Database, Two X and Y Columns

Example 152 plots the data from the “growthdata” database (see §3.2.10) which is read from a TSV file that has four columns. The first two (Exp1Time and Exp1Count) are the time and

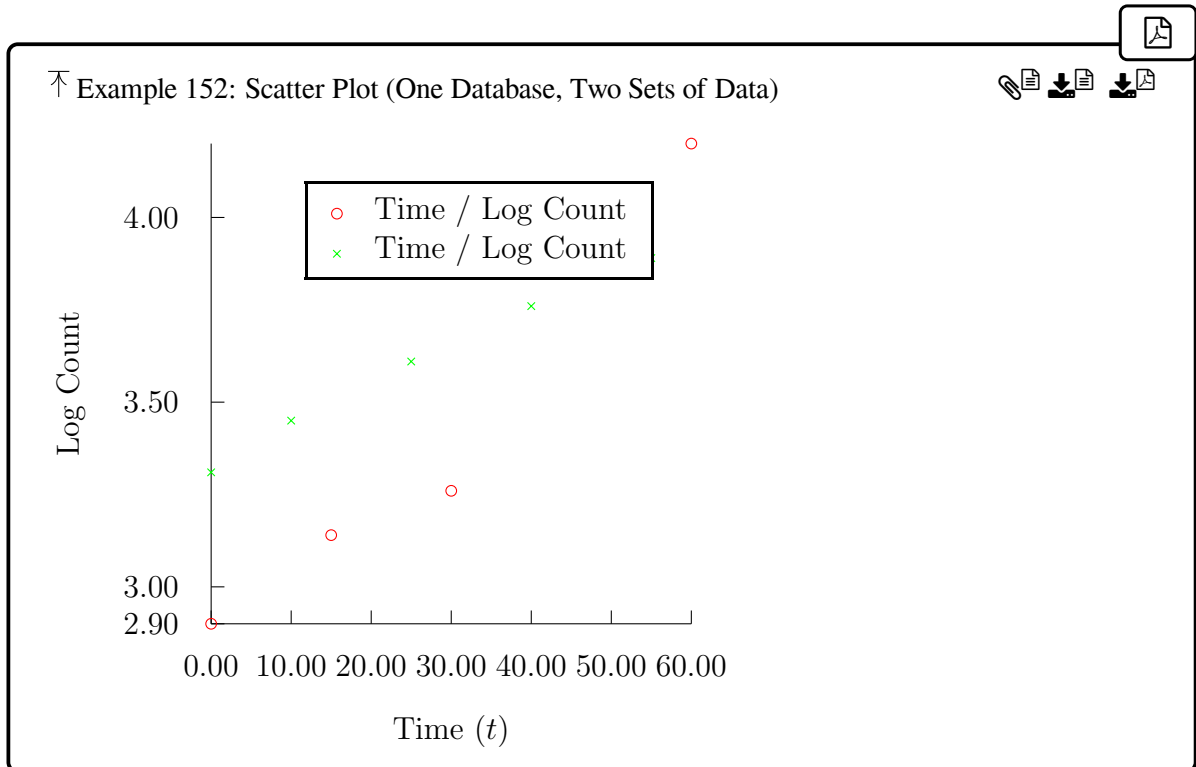
152



6. Scatter and Line Plots (dataplot package)

log count data from the first experiment and the last two (Exp2Time and Exp2Count) are the time and log count data from the second experiment. The two sets of data can be plotted with:

```
\DTLplot{growthdata}{
  x={Exp1Time,Exp2Time}, y={Exp1Count,Exp2Count},
  x-label={Time ($t$)}, y-label={Log Count},
  legend, width=2.5in, height=2.5in
}
```



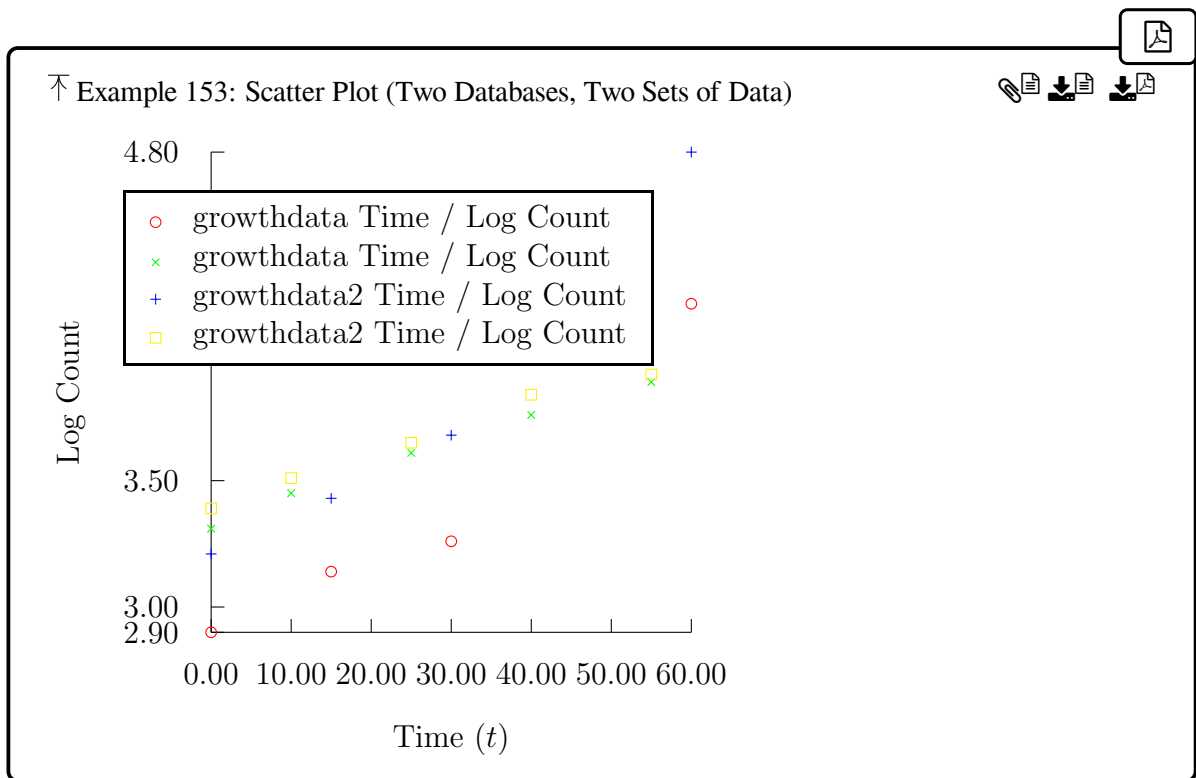
6.2.1.5. Two Databases, Two X and Y Columns

Example 153 makes a minor modification to Example 152 so that both the “growthdata” and “growthdata2” databases are loaded (see §3.2.10) and included in the plot.

153

```
\DTLplot{growthdata,growthdata2}{
  x={Exp1Time,Exp2Time},
  y={Exp1Count,Exp2Count},
  legend, width=2.5in,height=2.5in
}
```

This results in a very cluttered diagram where the legend obscures a large part of the plot.



6.2.1.6. Two Databases, Three X and Y Columns

Note that the “growthdata2” database actually has six columns, rather than four. Example 154 modifies Example 153 to include those two extra columns in the list:

154

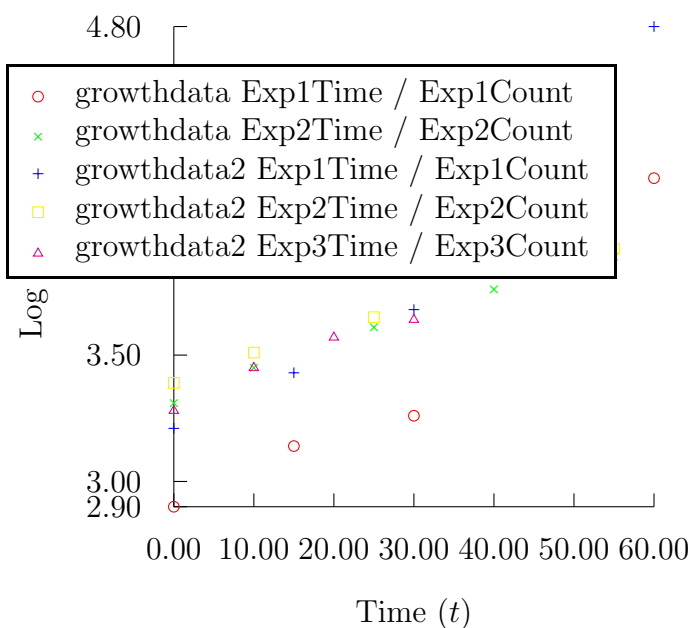
```
\DTLplot{growthdata,growthdata2}{
  x={Exp1Time,Exp2Time,Exp3Time},
  y={Exp1Count,Exp2Count,Exp3Count},
  x-label={Time ($t$)},y-label={Log Count},
  legend, width=2.5in,height=2.5in
}
```

The unknown columns will be skipped for the first database. If you switch on `verbose` mode, messages should be shown in the transcript. Those messages will turn into a warning if there are unknown columns when only one database is specified.

For debugging purposes, it’s useful to show the column keys in the legend instead of the column headers. This can be done as follows:

```
\RenewDocumentCommand \DTLplotlegendx { O{0} m O{0}
m }{#4}
\RenewDocumentCommand \DTLplotlegendy { O{0} m O{0}
m }{#4}
```

Example 154: Scatter Plot (Two Databases, Multiple Sets of Data)



The legend is now so large it obscures half of the plot. Examples that adjust the legend settings are in §6.2.2.

6.2.1.7. Two Databases, Two X and Three Y Columns

In general, it's best to either have a single key in the x setting or the same number of keys as for the y setting. To demonstrate what happens when the x list has more than one key but less than the total number of y keys, Example 155 modifies Example 154 to have two x keys and three y keys.

155

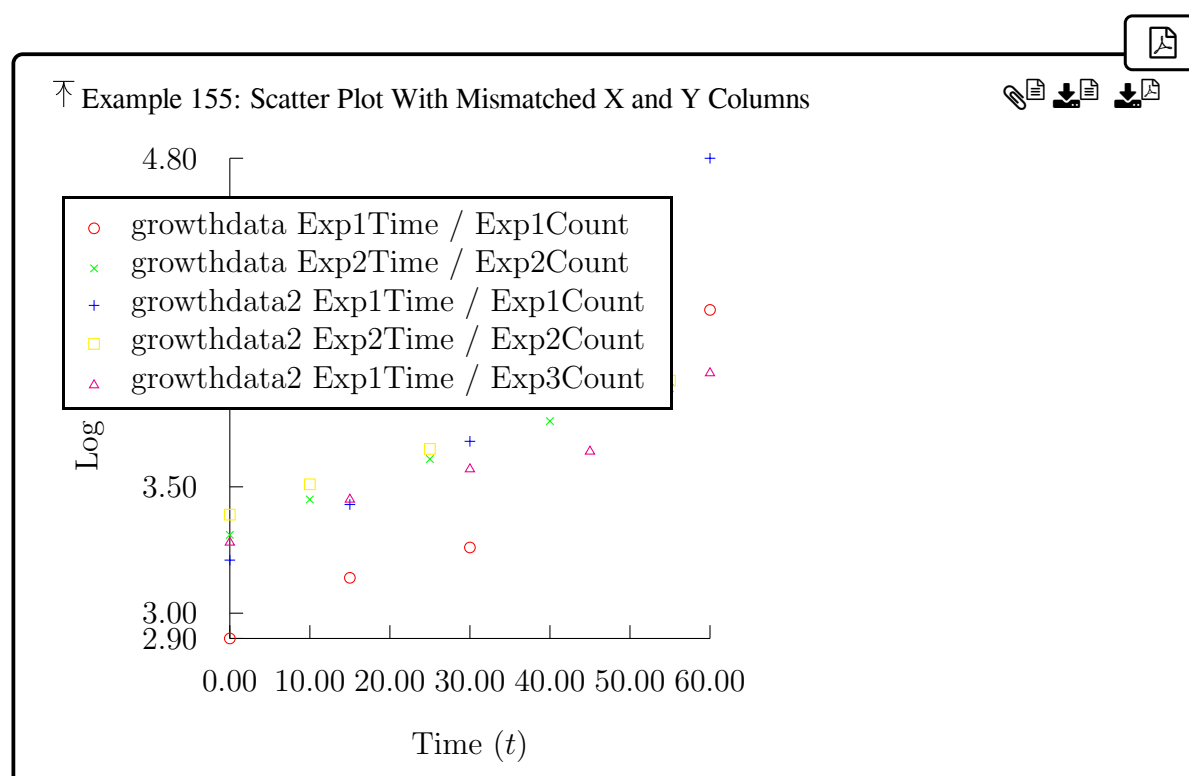
```
\DTLplot{growthdata,growthdata2}{
x={Exp1Time,Exp2Time},
y={Exp1Count,Exp2Count,Exp3Count},
x-label={Time ($t$)}, y-label={Log Count},
```

```
legend, width=2.5in, height=2.5in
}
```

Again, `\DTLplotlegendx` and `\DTLplotlegandy` are redefined to show the column keys, and, again, the legend obscures the plot, but it's the legend that is of interest as it shows the combinations of the x and y columns used to generate the plot.

The plot streams for the second database are obtained from the X/Y pairs: `Exp1Time/Exp1Count`, `Exp2Time/Exp2Count` and `Exp1Time/Exp3Count`. Once the end of the x list is reached, it cycles back until all the y columns have been iterated over. This method allows a single x column to be plotted against multiple y columns, but also allows a list of x columns to be plotted against a matching element from a list of y columns.

In general, it's better to repeat a column key in x than to have this situation.



6.2.2. Legend Examples

If the `legend` option is set then each plot stream for a given x and y pair for a given database will add a row to the legend (as demonstrated in the previous examples). There are user hooks available (see §6.3) but essentially each row in the legend starts with the plot mark or line (or both) used by the plot stream followed by text in the form:

```
<name> <x-header> / <y-header>
```

If there is only one database the $\langle name \rangle$ part will be omitted (unless there is also only one x and y), and if there is only one x column the $\langle x-header \rangle$ and following slash separator will be omitted.

The $\langle x-header \rangle$ part defaults to the column header for the corresponding x column, which is obtained with `\DTLplotlegendx`, and the $\langle y-header \rangle$ part defaults to the column header for the corresponding y column, which is obtained with `\DTLplotlegendy`. This means that each set of data is labelled “Time / Log Count” in examples 152 & 153, since the column headers are repeated for each x/y pair.

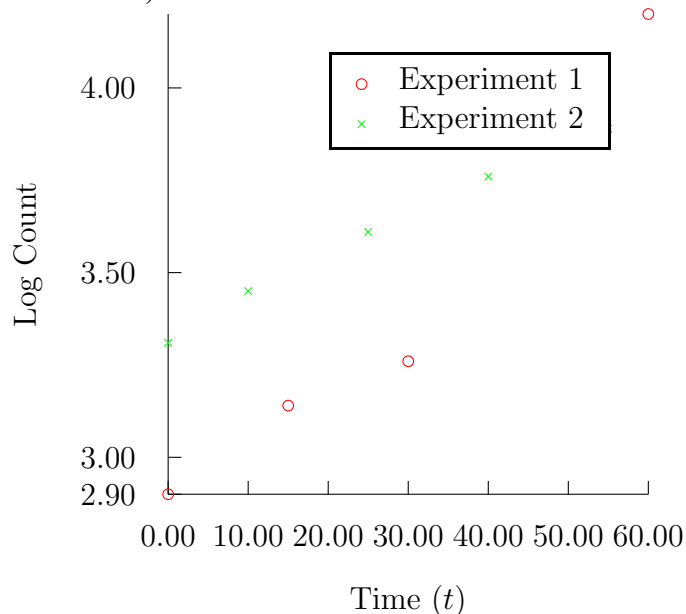
6.2.2.1. Custom Legend Text With `legend-labels`

Example 156 has a minor change to Example 152 that explicitly sets the legend labels with the `legend-labels` option:

156

```
\DTLplot{growthdata}{
  x={Exp1Time,Exp2Time},
  y={Exp1Count,Exp2Count},
  x-label={Time ($t$)},y-label={Log Count},
  legend, legend-labels={Experiment 1,Experiment 2},
  width=2.5in, height=2.5in
}
```

Example 156: Scatter Plot with Custom Legend Labels (One Database, Two Sets of Data)



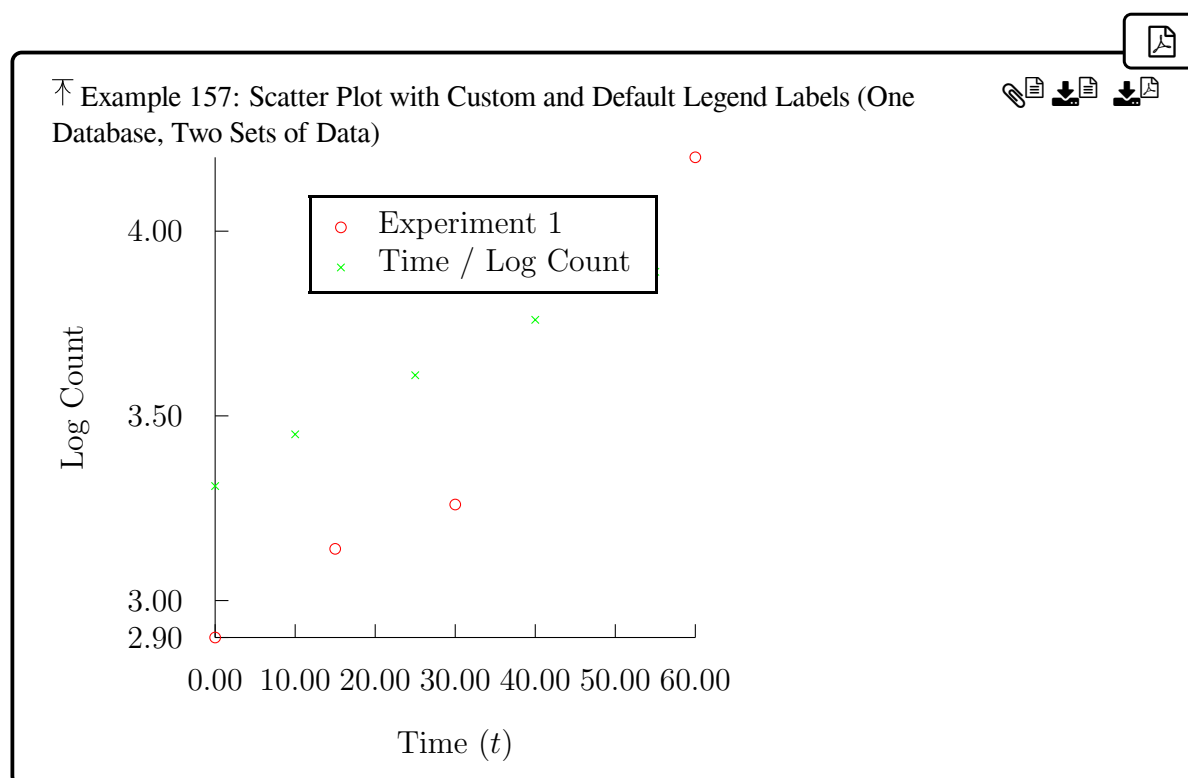
6.2.2.2. Subset Custom Legend Text With `legend-labels`

If you don't provide enough items in `legend-labels`, the defaults will be used for the missing ones. Example 157 modifies Example 156 so that only one legend label is supplied:

157

```
\DTLplot{growthdata}{
  x={Exp1Time,Exp2Time},
  y={Exp1Count,Exp2Count},
  x-label={Time ($t$)},y-label={Log Count},
  legend, legend-labels={Experiment 1},
  width=2.5in, height=2.5in
}
```

This only occurs when the `legend-labels` list is shorter than the total number of plot streams. It's not possible to have the default for the first and specify a label for the second with `legend-labels`.



The `<key>=<value>` parser discards empty elements, so `legend-labels={Experiment 1, }` (with a trailing comma) is equivalent to `legend-labels={Experiment 1}`. If an empty value is required, the empty value needs to be grouped.

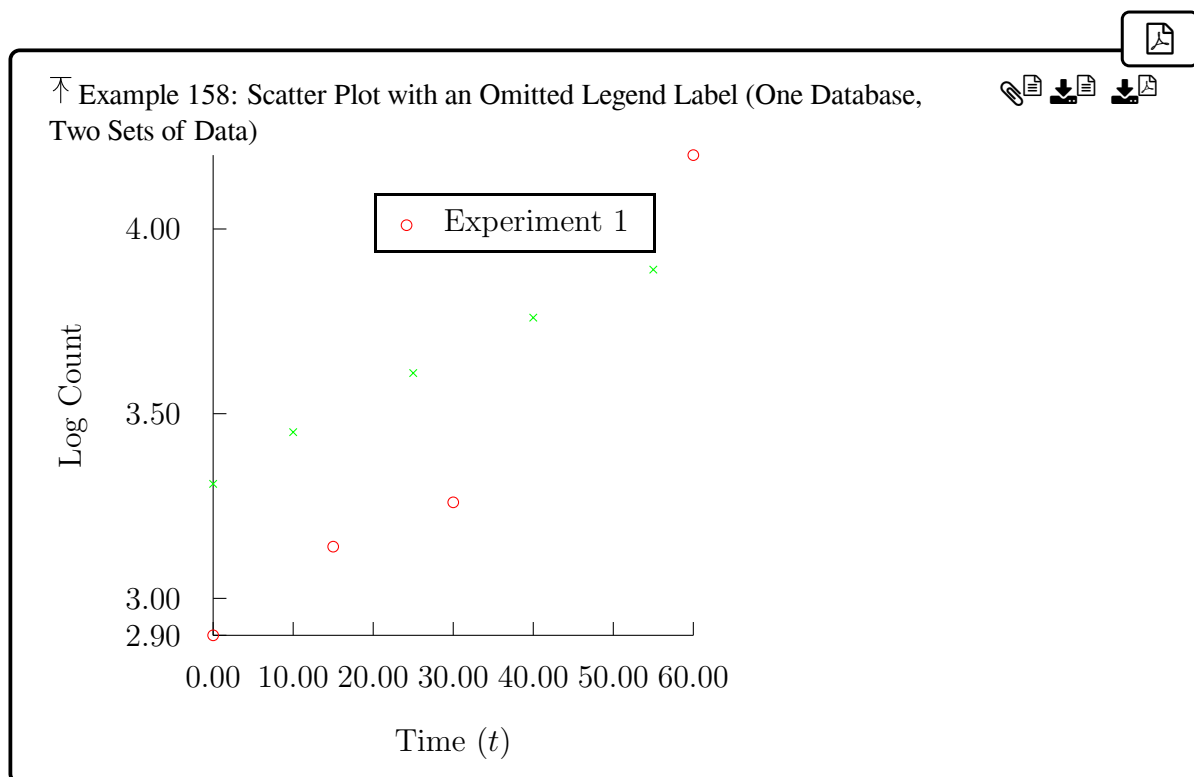
6.2.2.3. Omitting Stream from Legend With `legend-labels`

Example 158 modifies Example 157 so that the second label is explicitly set to empty:

158

```
\DTLplot{growthdata}{
  x={Exp1Time,Exp2Time},
  y={Exp1Count,Exp2Count},
  x-label={Time ($t$)}, y-label={Log Count},
  legend, legend-labels={Experiment 1,{ }},
  width=2.5in, height=2.5in
}
```

This causes the second stream to be omitted from the legend.

**6.2.2.4. Legend Database Mapping**

The use of `legend-labels` may be sufficient for a small number of plot streams, as in Example 156, but it can become more complicated with additional files. The earlier Example 150 which had two files, showed the database names in the legend (“growth1” and “growth2”). Let’s suppose that the experiments in the first database were conducted at a temperature of 6 degrees and the experiments in the second database were conducted at a temperature of 8 degrees.

The `siunitx` package conveniently provides a way of typesetting temperature, so the legend labels could be set using:

```
legend-labels={
  \qty{6}{\degreeCelsius} Experiment 1,
  \qty{6}{\degreeCelsius} Experiment 2,
  \qty{8}{\degreeCelsius} Experiment 1,
  \qty{8}{\degreeCelsius} Experiment 2
}
```

However, this can get quite cumbersome (particularly for Example 154, which has two pairs of x/y data in one database and three in the other).

Example 159 modifies Example 150 to set up mappings to supply in place of the database name in the legend. 159

```
\DTLplotlegendsetname{growth1}{\qty{6}
{\degreeCelsius}}
\DTLplotlegendsetname{growth2}{\qty{8}
{\degreeCelsius}}
```

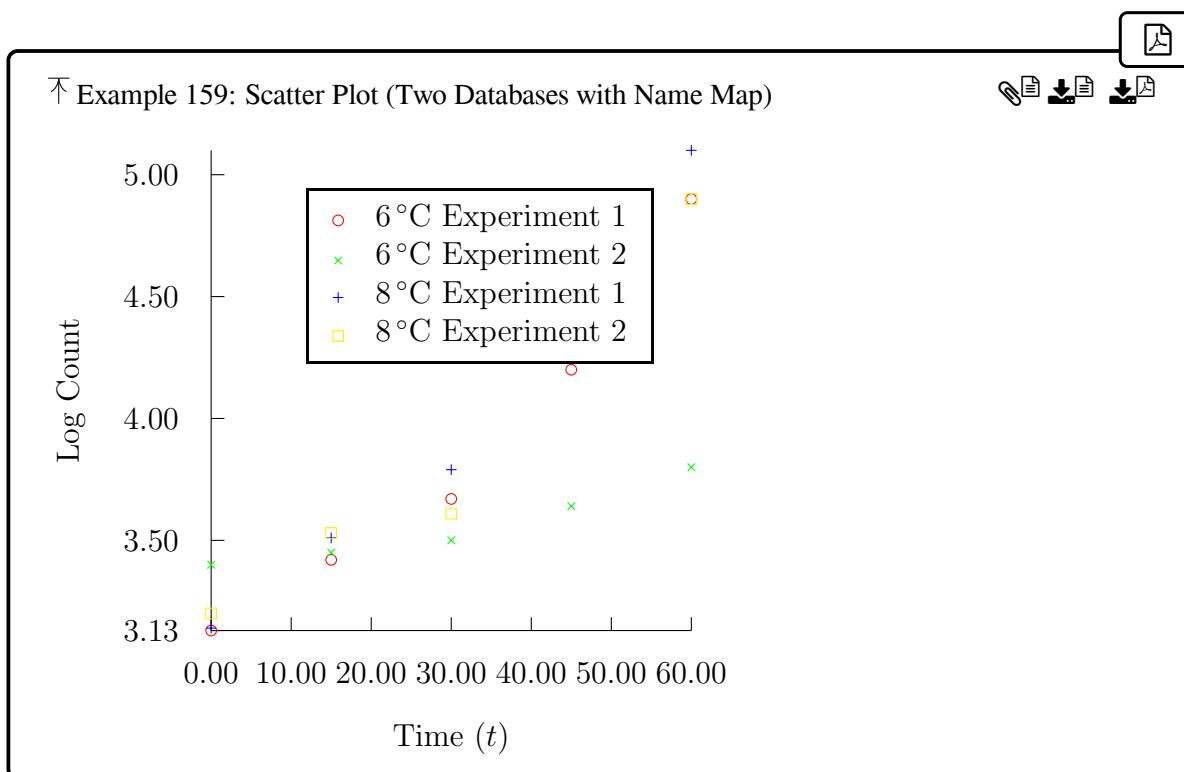
This needs to be done before `\DTLplot`. The rest of Example 159 is as Example 150 (but remember to include `siunitx`).

The mappings are used by `\DTLplotlegendname`, which is only present if there are multiple databases.

6.2.2.5. Legend Column Key Mapping

Example 160 modifies Example 154 in a similar manner, but also establishes a mapping for the column keys. Remember to remove the redefinitions of `\DTLplotlegendx` and `\DTLplotlegende` from that example otherwise the mappings will be ignored. 160

```
\DTLplotlegendsetname{growthdata}{\qty{6}
{\degreeCelsius}}
\DTLplotlegendsetname{growthdata2}{\qty{8}
{\degreeCelsius}}
\DTLplotlegendsetxlabel{Exp1Time}{\textit{t}_1}
\DTLplotlegendsetxlabel{Exp2Time}{\textit{t}_2}
```

```
\DTLplotlegendsetxlabel{Exp3Time}{\t_3$}
\DTLplotlegendsetylabel{Exp1Count}{\N_1$}
\DTLplotlegendsetylabel{Exp2Count}{\N_2$}
\DTLplotlegendsetylabel{Exp3Count}{\N_3$}
```

6.2.2.6. Customising the X/Y Legend

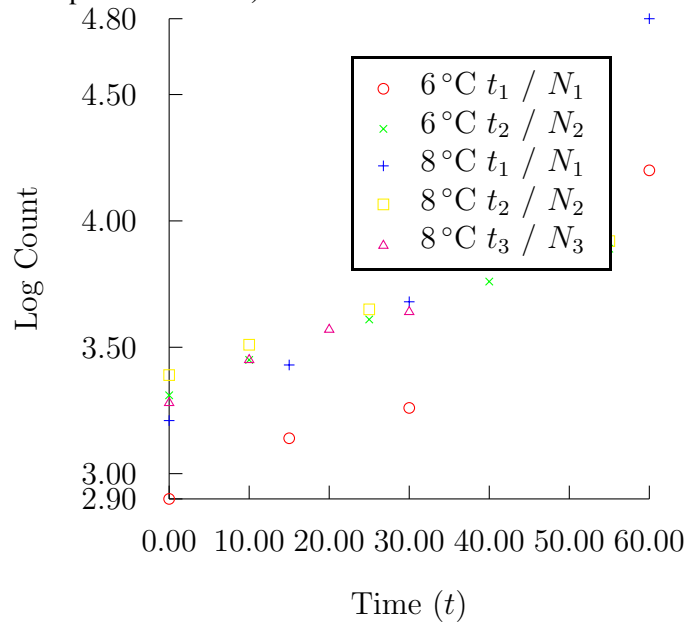
The slash separator can be changed by redefining `\DTLplotlegendxysep` but Example 161 instead modifies Example 160 to include a redefinition of `\DTLplotlegendxy` that doesn't use `\DTLplotlegendxysep` but instead uses a comma and parentheses:

161

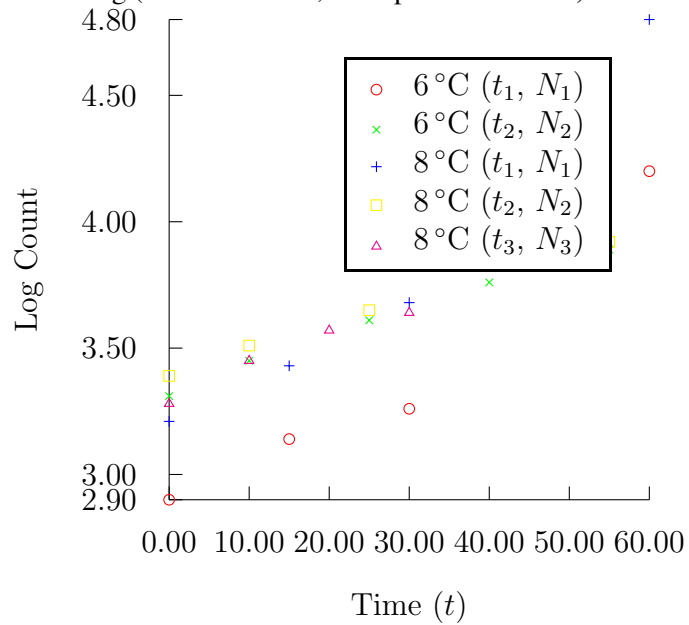
```
\RenewDocumentCommand \DTLplotlegendxy { O{0} m O{0}
m O{0} m }
{%
(\DTLplotlegendx[#1]#2[#3]{#4},
\DTLplotlegendy[#1]#2[#5]{#6})%
}
```

6. Scatter and Line Plots (dataplot package)

Example 160: Scatter Plot with Legend Label Mappings (Two Databases, Multiple Sets of Data)



Example 161: Scatter Plot with Legend Label Mappings and Custom formatting (Two Databases, Multiple Sets of Data)



6.2.2.7. Customising the Legend, No X Label

Example 162 uses an alternative approach to Example 161 that redefines `\DTLplotlegendxy` to omit the x text, which means that only the database name and y column key mappings are needed:

162

```
\RenewDocumentCommand \DTLplotlegendxy { O{0} m O{0}
m O{0} m }
{%
\DTLplotlegendy[#1]{#2}[#5]{#6}%
}
\DTLplotlegendsetname{growthdata}{$T=6$}
\DTLplotlegendsetname{growthdata2}{$T=8$}
\DTLplotlegendsetylabel{Exp1Count}{Experiment 1}
\DTLplotlegendsetylabel{Exp2Count}{Experiment 2}
\DTLplotlegendsetylabel{Exp3Count}{Experiment 3}
```

In this case, since each y label needs to be in the form “Experiment $\langle y\text{-index} \rangle$ ”, this can be simplified further:

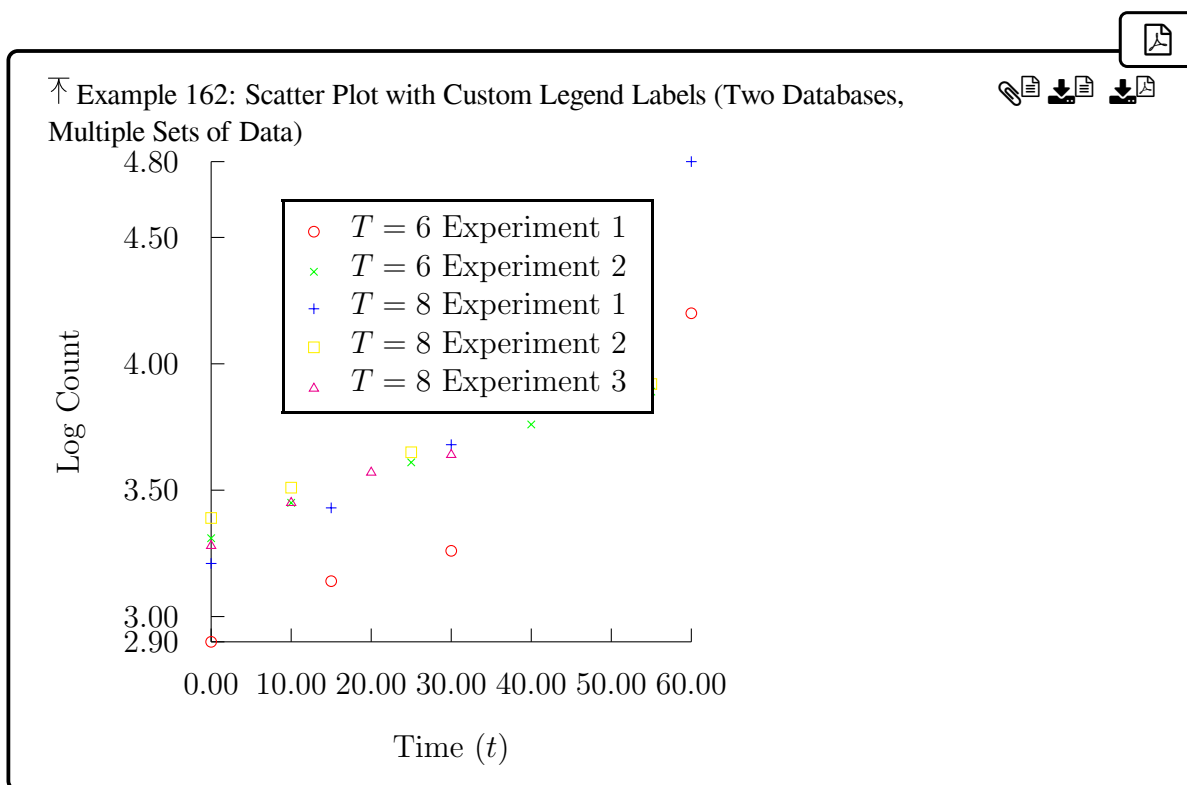
```
\RenewDocumentCommand \DTLplotlegendxy { O{0} m O{0}
m O{0} m }
{%
\DTLplotlegendy[#1]{#2}[#5]{#6}%
}
\DTLplotlegendsetname{growthdata}{$T=6$}
\DTLplotlegendsetname{growthdata2}{$T=8$}
\RenewDocumentCommand \DTLplotlegendy { O{0} m O{0}
m }
{Experiment #3}
```

6.2.2.8. Shifting the Legend

The examples so far have simply used `legend` without a value. This is equivalent to `legend=northwest`. This positions the legend in the north west (top right) of the plot at an offset given by the lengths `\DTLlegendxoffset` and `\DTLlegendyoffset`. These can be more conveniently set with the `legend-offset` option. The default value for both offsets is 10pt. Note that this is an absolute length not in data co-ordinates. A negative offset will position the legend outside of the plot.

Example 163 makes a minor modification to Example 162 to shift the legend to the right of the plot:

163



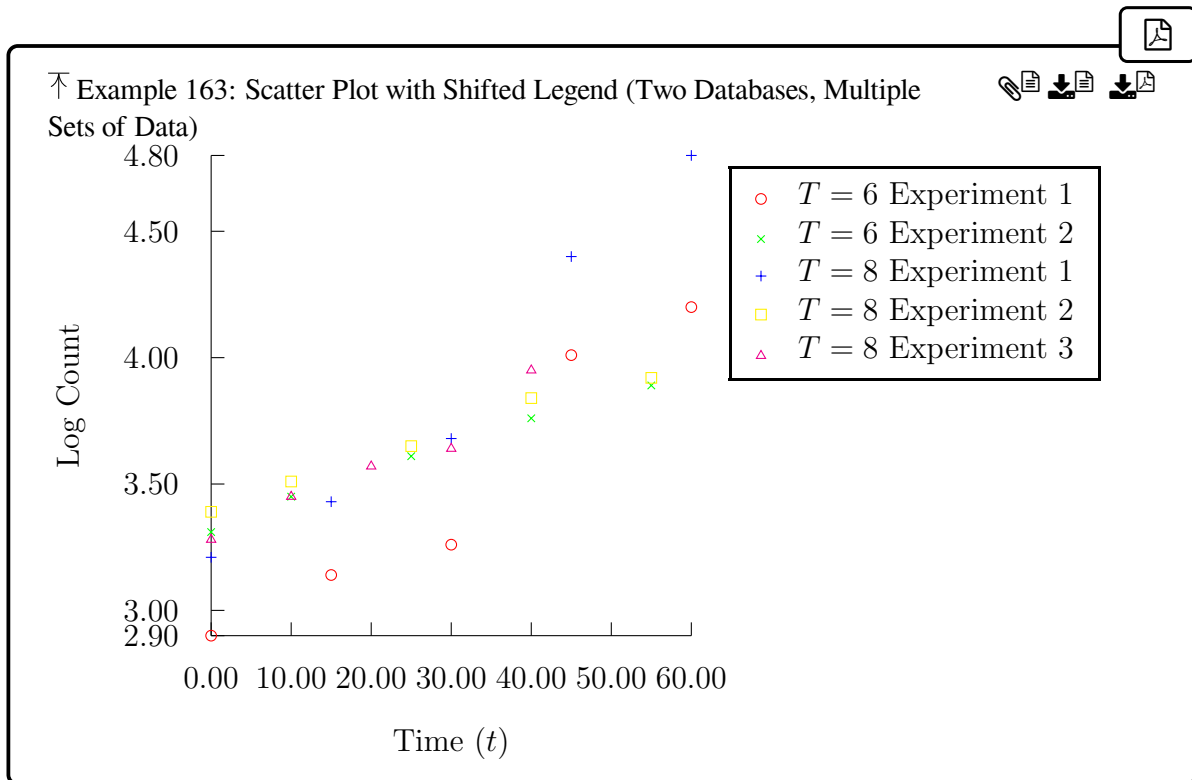
```
\DTLplot{growthdata,growthdata2}{
  x={Exp1Time,Exp2Time,Exp3Time},
  y={Exp1Count,Exp2Count,Exp3Count},
  x-label={Time ($t$)},y-label={Log Count},
  legend, legend-offset={-10pt,0pt},
  width=2.5in,height=2.5in
}
```

6.2.2.9. Custom Legend Position

Instead of using one of the pre-defined legend locations, you can use the `legend=custom` setting and redefine `\DTLcustomlegend` to position the legend exactly where you want it. Example 164 modifies Example 163 to use the `tikz shapes.callouts` library:

```
\usetikzlibrary{shapes.callouts}
```

and redefines `\DTLcustomlegend` to position the legend at $(\langle W \rangle, \langle H \rangle / 4)$ where $\langle W \rangle$ is the plot width and $\langle H \rangle$ is the plot height:



```
\renewcommand{\DTLcustomlegend}[1]{%
\node

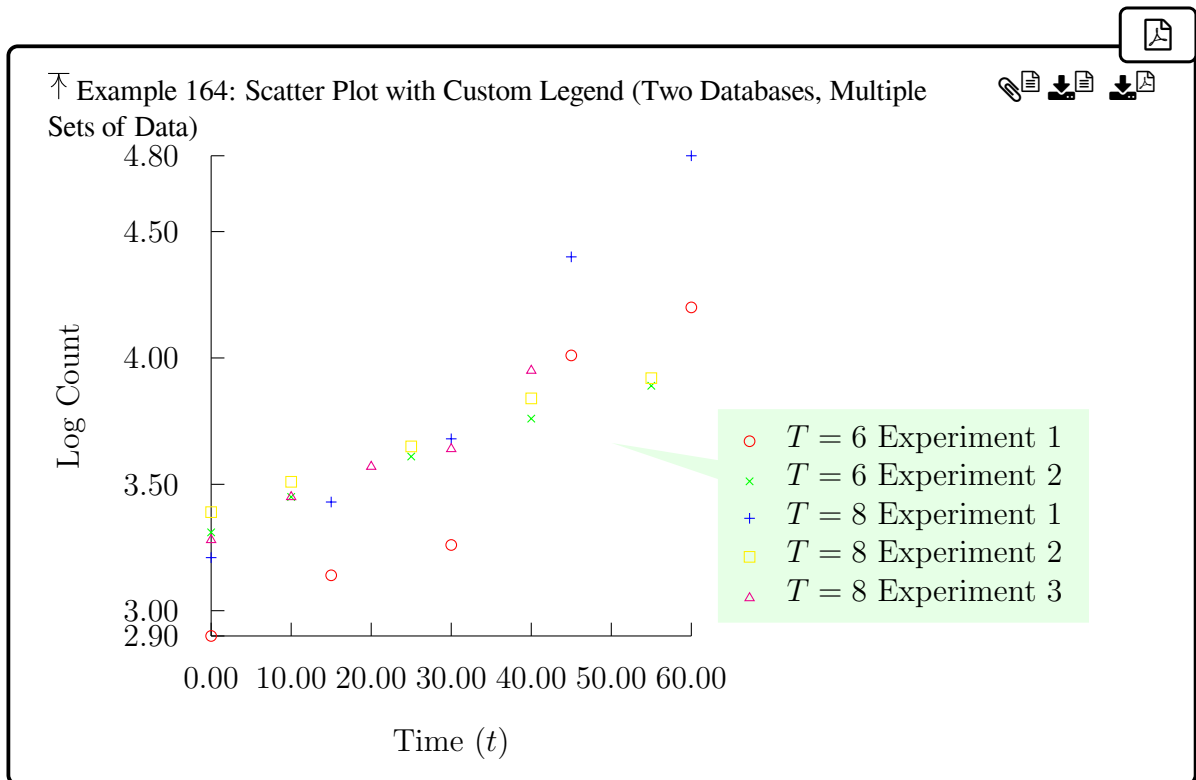
[rectangle callout,fill=green!10,anchor=west,outer sep=10pt,
callout relative pointer={(-40pt,10pt)}
] at (\DTLplotwidth,0.25\DTLplotheight)
{#1} ;
}
```

Note that this definition doesn't include `\DTLformatlegend` but instead uses the `\node` options to set the shape and background colour.

The plot now needs the `legend` option adjusting:

```
\DTLplot{growthdata,growthdata2}{
x={Exp1Time,Exp2Time,Exp3Time},
y={Exp1Count,Exp2Count,Exp3Count},
x-label={Time ($t$)},y-label={Log Count},
legend=custom,
width=2.5in,height=2.5in}
```

}



6.2.3. Plot Style Examples

The examples so far have only shown markers with the default mark style and colours. The plot can have the style changed to show both lines and markers with `style=both` or only lines with `style=lines`.

6.2.3.1. Line and Scatter Plot

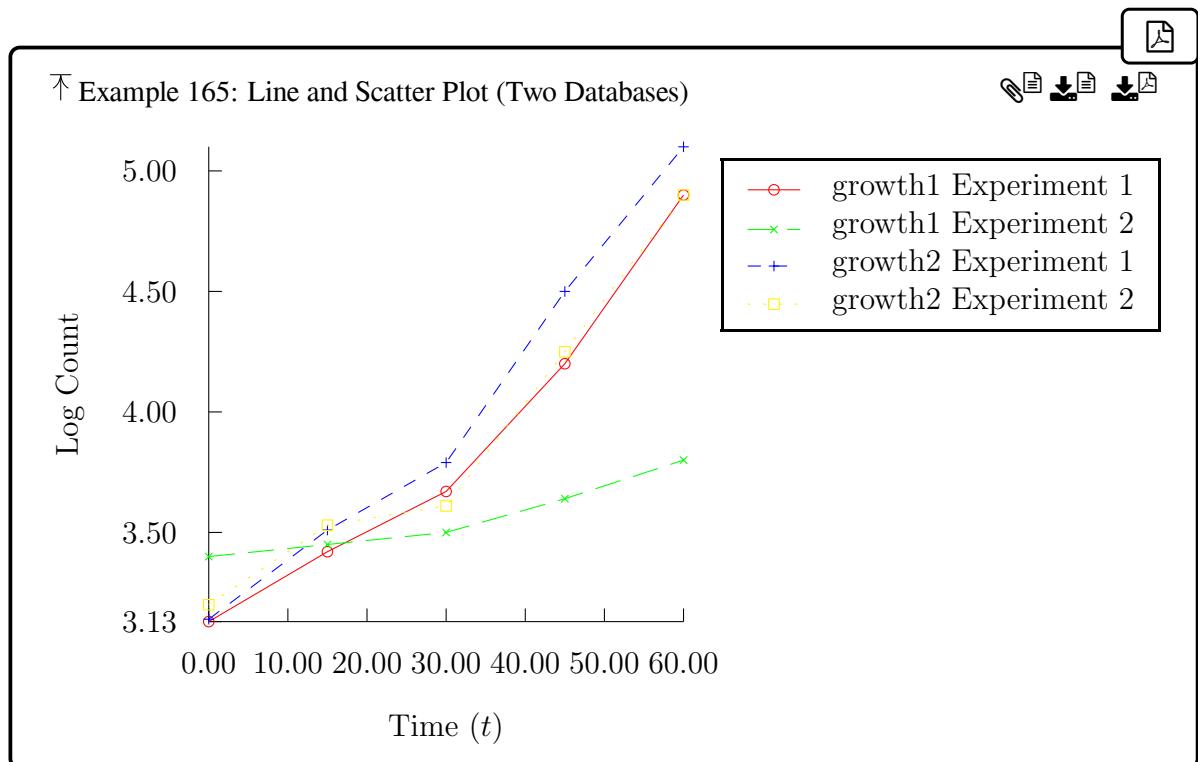
Example 165 is a simple modification of Example 150 which includes lines as well as plot marks. This is done by setting `style=both`:

165

```
\DTLplot{growth1,growth2}{
  style=both,
  x=Time, y={Experiment 1,Experiment 2},
  x-label={Time ($t$)},y-label={Log Count},
  legend, legend-offset={-10pt,0pt},
  width=2.5in,height=2.5in}
```

}

Note that the legend has also been shifted, as per Example 163.



6.2.3.2. Changing the Colours and Styles

Example 166 modifies Example 161 to show lines as well as markers but also sets the line and marker colours and the line style and plot marks (again the legend is shifted):

166

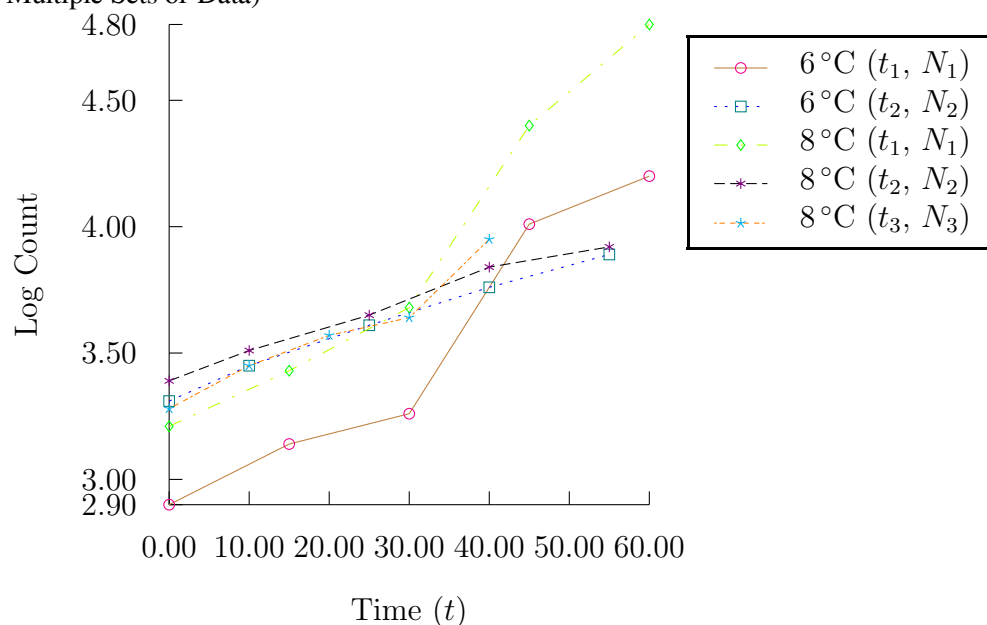
```
\DTLplot{growthdata,growthdata2}{
  style=both,% lines and markers
  line-colors={brown,blue,lime,black,orange},
  mark-colors={magenta,teal,green,violet,cyan},
  lines={
    \pgfsetdash{{0pt}},% solid line
    \pgfsetdash{{1pt}}{3pt}}{0pt},
    \pgfsetdash{{5pt}}{5pt}}{1pt}}{5pt}}{0pt},
    \pgfsetdash{{4pt}}{2pt}}{0pt},
    \pgfsetdash{{1pt}}{1pt}}{2pt}}{2pt}}{2pt}}{1pt}}{0pt}
  },
  marks={
```

```

\pgfuseplotmark{o},
\pgfuseplotmark{square},
\pgfuseplotmark{diamond},
\pgfuseplotmark{asterisk},
\pgfuseplotmark{star}
},
x={Exp1Time,Exp2Time,Exp3Time},
y={Exp1Count,Exp2Count,Exp3Count},
x-label={Time ($t$)},y-label={Log Count},
legend, legend-offset={-10pt,0pt},
width=2.5in,height=2.5in
}

```

Example 166: Scatter Plot with Custom Colours and Styles (Two Databases, Multiple Sets of Data)



6.2.3.3. One Line Colour Per Database

The `group-styles` option may be used to only change a particular style per database, rather than using each style per plot stream. Example 167 modifies Example 166 to use the same line colour for each stream in a given database:

167


```

\DTLplot{growthdata,growthdata2}{
  style=both, group-styles={line-color},
  line-colors={brown,blue,lime,black,orange},
  mark-colors={magenta,teal,green,violet,cyan},
  lines={
    \pgfsetdash{}{0pt},% solid line
    \pgfsetdash{{1pt}{3pt}}{0pt},
    \pgfsetdash{{5pt}{5pt}{1pt}{5pt}}{0pt},
    \pgfsetdash{{4pt}{2pt}}{0pt},
    \pgfsetdash{{1pt}{1pt}{2pt}{2pt}{2pt}{1pt}}{0pt}
  },
  marks={
    \pgfuseplotmark{o},
    \pgfuseplotmark{square},
    \pgfuseplotmark{diamond},
    \pgfuseplotmark{asterisk},
    \pgfuseplotmark{star}
  },
  x={Exp1Time,Exp2Time,Exp3Time},
  y={Exp1Count,Exp2Count,Exp3Count},
  x-label={Time ($t$)},y-label={Log Count},
  legend, legend-offset={-10pt,0pt},
  width=2.5in,height=2.5in
}

```

Note that the other styles (line style, marker style and marker colour) still cycle round the given lists.

6.2.3.4. Resetting the Style for Each Database

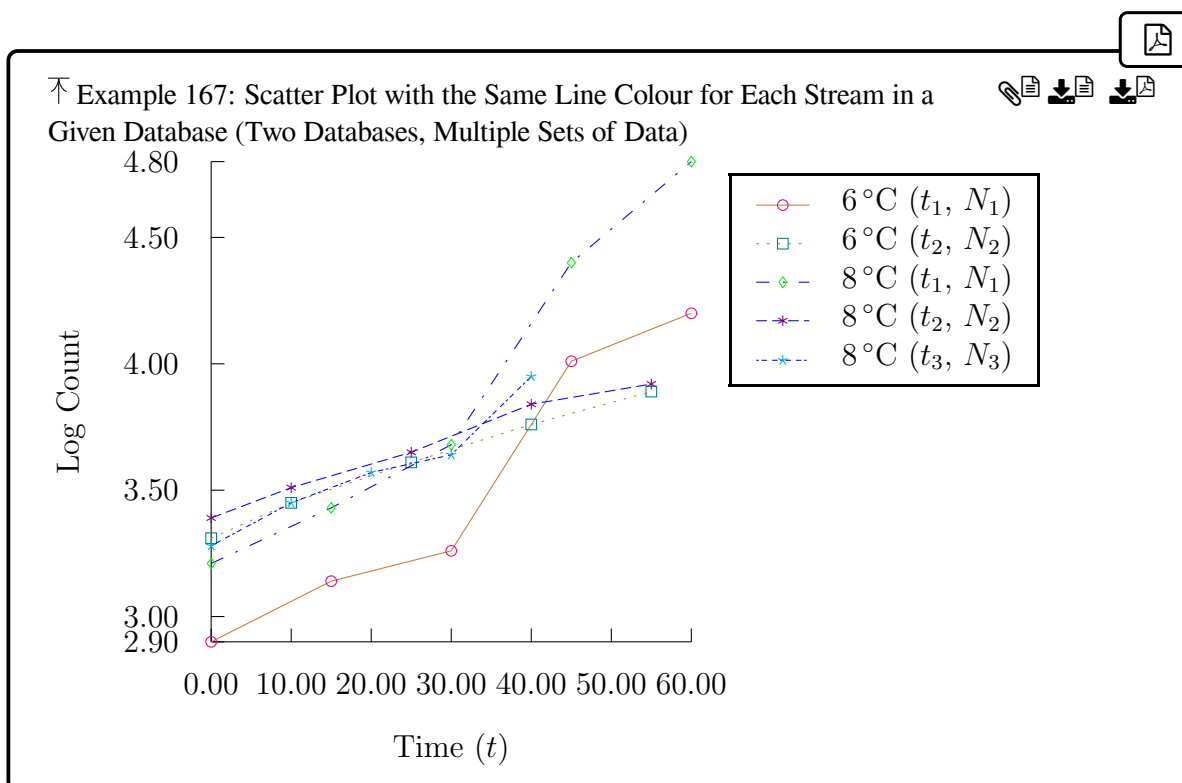
Example 168 modifies Example 167 so that the plot marks are reset at the start of each database, and the only available line style is solid.

168

```

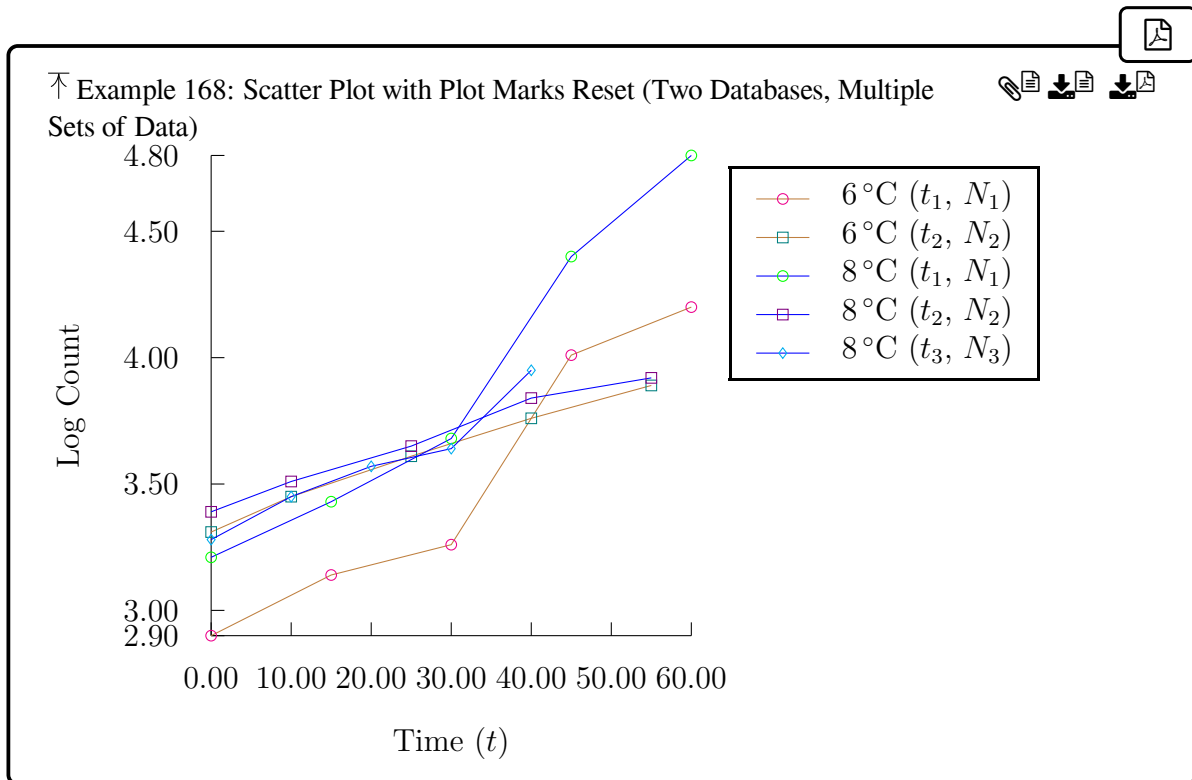
\DTLplot{growthdata,growthdata2}{
  style=both, group-styles={line-color},
  style-resets={mark-style},
  line-colors={brown,blue,lime,black,orange},
  mark-colors={magenta,teal,green,violet,cyan},
  lines={
    \pgfsetdash{}{0pt}% solid line
  },

```



```
marks={
  \pgfuseplotmark{o},
  \pgfuseplotmark{square},
  \pgfuseplotmark{diamond},
  \pgfuseplotmark{asterisk},
  \pgfuseplotmark{star}
},
x={Exp1Time,Exp2Time,Exp3Time},
y={Exp1Count,Exp2Count,Exp3Count},
x-label={Time ($t$)},y-label={Log Count},
legend, legend-offset={-10pt,0pt},
width=2.5in,height=2.5in
}
```

This means that a solid line will always be used (with either `style=both` or `style=lines`) regard of the `group-styles` or `style-resets` options. The first database has brown lines and the second database has blue lines. Experiment 1 for each database has circle markers, Experiment 2 has square markers and Experiment 3 (which is only in the second database) has diamond markers. The other listed markers aren't used. Note, however, that the marker colours still cycle through the entire `mark-colors` list.



6.2.4. Plot Axes Examples

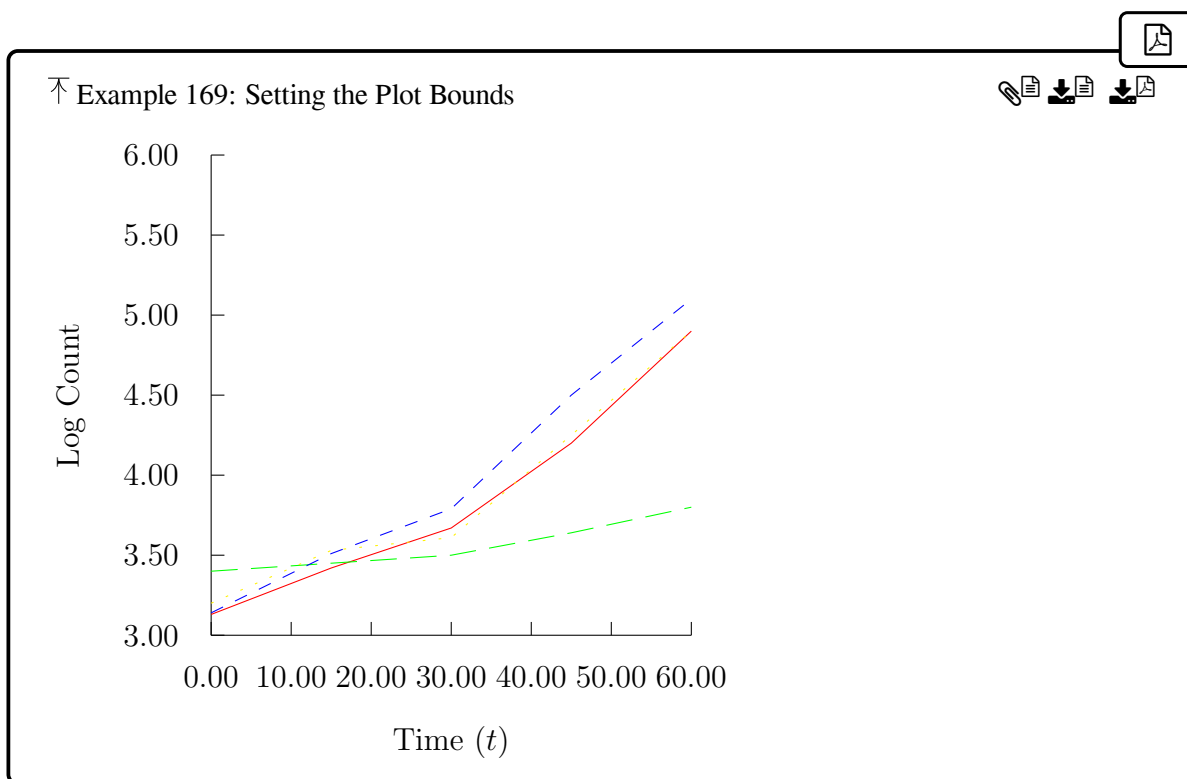
6.2.4.1. Setting the Bounds

The examples so far have the plot bounds automatically calculated. This means that the y tick marks look a little strange as they don't start from a whole or half number. The x tick marks look more even as the sample data happens to have convenient x values. The examples in this section demonstrate how to set the upper and lower bounds.

Example 169 modifies Example 150 to set upper and lower bounds for the y axis. Since only two of the bounds need changing, it's simpler to use `min-y` and `max-y` rather than use `bounds`. This doesn't show the legend and switches to lines rather than plot marks.

169

```
\DTLplot{growth1,growth2}{
  x=Time,
  y={Experiment 1,Experiment 2},
  x-label={Time ($t$)}, y-label={Log Count},
  min-y=3,max-y=6,
  style=lines, width=2.5in, height=2.5in
}
```



6.2.4.2. Tick Label Rounding

Example 170 modifies Example 169 to round the x tick labels to whole numbers and round the y tick labels to one decimal place.

170

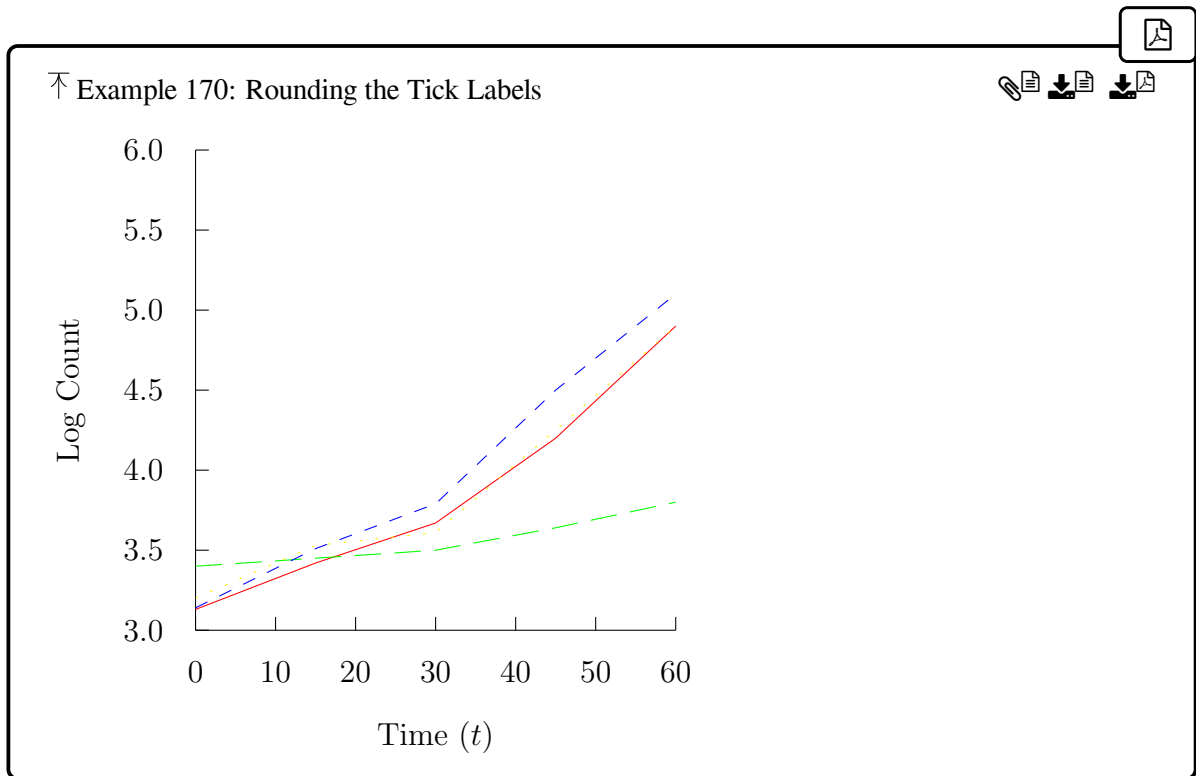
```
\DTLplot{growth1,growth2}{
  x=Time,
  y={Experiment 1,Experiment 2},
  x-label={Time ($t$)}, y-label={Log Count},
  min-y=3,max-y=6,
  round-x=0, round-y=1,
  style=lines, width=2.5in, height=2.5in
}
```

6.2.4.3. Axis Style

The tikz line drawing style of the x and y axes can be changed with the `axis-style` option.

Example 171 modifies Example 170 to include an arrow head and also makes the lines thicker using `axis-style={->,thick}`. The tick direction is also changed to outside of the plot using `tick-dirout` and the minor tick marks are included with `minor-ticks`.

171



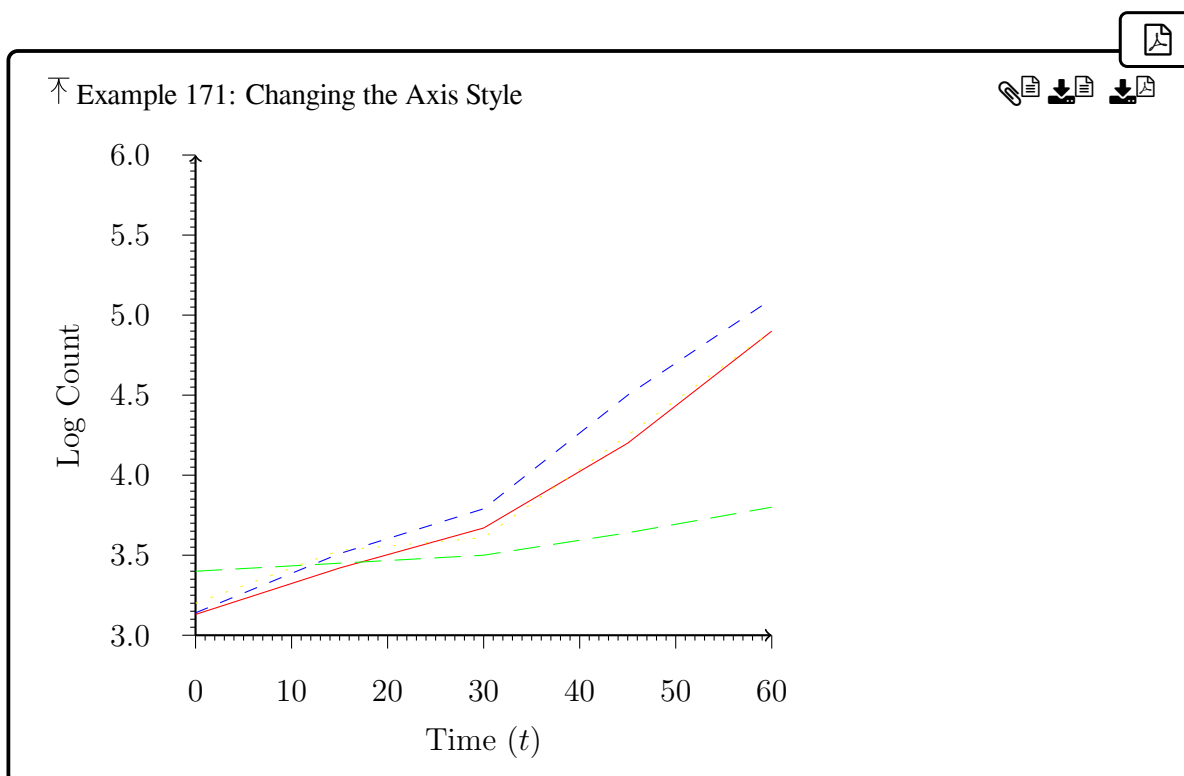
```
\DTLplot{growth1,growth2}{
  x=Time,
  y={Experiment 1,Experiment 2},
  x-label={Time ($t$)}, y-label={Log Count},
  min-y=3,max-y=6,
  round-x=0, round-y=1,
  tick-dir=out, minor-ticks, axis-style={->,thick},
  style=lines, width=3in, height=2.5in
}
```

Note that the arrow heads overlap the end tick marks. This can be dealt with by extending the axes using `extend-x-axis` and `extend-y-axis` (see §6.2.4.7). This is different to changing the plot bounds as it extends the axes without adding additional tick marks.

6.2.4.4. Grid

The `grid` option draws a grid in the background before drawing the plot streams. Example 172 modifies Example 170 to show the grid. As with Example 171, the minor tick marks are also enabled. If minor tick marks aren't enabled, only the major grid lines will be drawn.

172

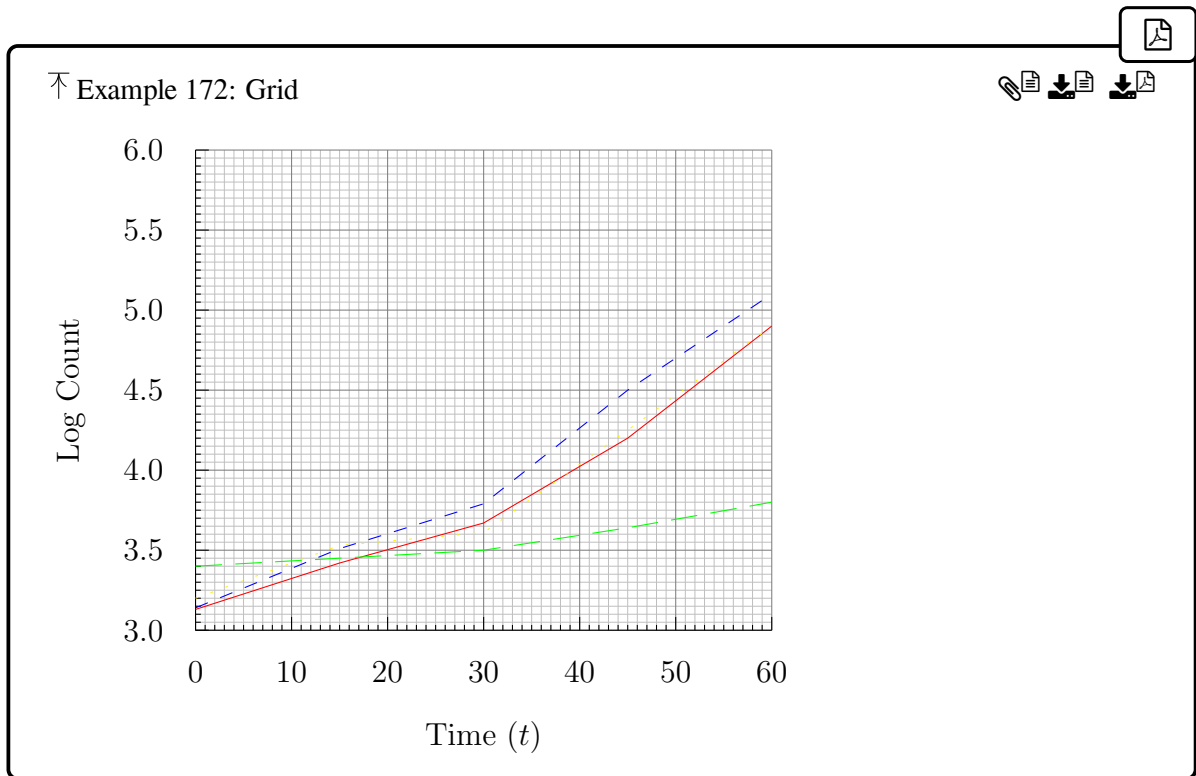


```
\DTLplot{growth1,growth2}{
  x=Time,
  y={Experiment 1,Experiment 2},
  x-label={Time ($t$)}, y-label={Log Count},
  min-y=3,max-y=6,
  round-x=0, round-y=1,
  grid, minor-ticks,
  style=lines, width=3in, height=2.5in
}
```

The style of the major grid lines is obtained by expanding `\DTLmajorgridstyle`, and the style of the minor grid lines is obtained by expanding `\DTLminorgridstyle`. These can be redefined as required but should expand to valid `tikz` options. If `\DTLminorgridstyle` is redefined to expand to nothing then the minor grid lines won't be drawn even if the tick marks are enabled. For example:

```
\renewcommand{\DTLmajorgridstyle}{gray,ultra thick}
\renewcommand{\DTLminorgridstyle}{{}}
```

Instead of explicitly redefining those commands, you can use the `major-grid-style` and



minor-grid-style options.

Example 173 modifies Example 172 to change the grid style, but rather than explicitly redefining the commands, as above, the plot options are used instead:

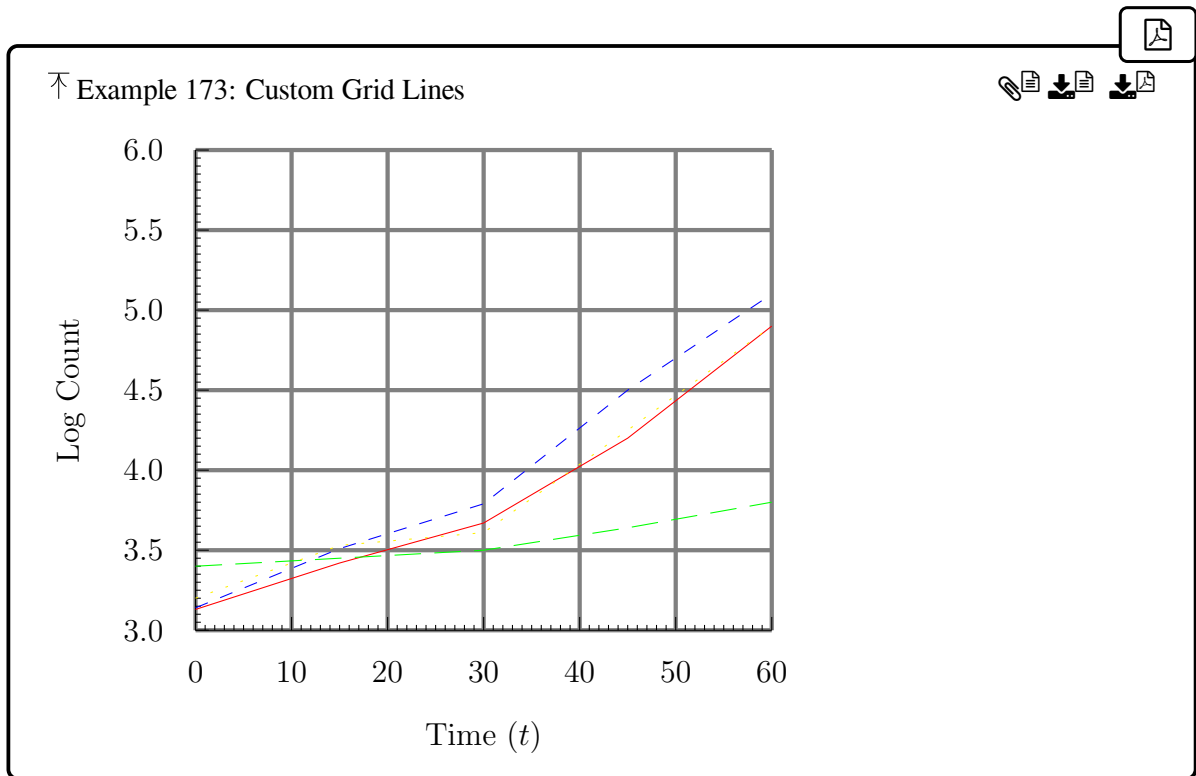
173

```
\DTLplot{growth1,growth2}{
  x=Time,
  y={Experiment 1,Experiment 2},
  x-label={Time ($t$)}, y-label={Log Count},
  min-y=3,max-y=6, minor-ticks,
  round-x=0, round-y=1,
  grid, major-grid-style={gray,ultra thick}, minor
-grid-style={},
  style=lines, width=3in, height=2.5in
}
```

This results in thick major grid lines and no minor grid lines (even though the minor ticks are on).

6.2.4.5. Box

The `box` option encapsulates the (extended) plot bounds with a box. Note that this doesn't include the tick labels and axis labels unless they also happen to lie within those bounds. Exam-



Example 174 adapts Example 149 to include tick rounding, an adjustment to the plot bounds, and the encapsulating box.

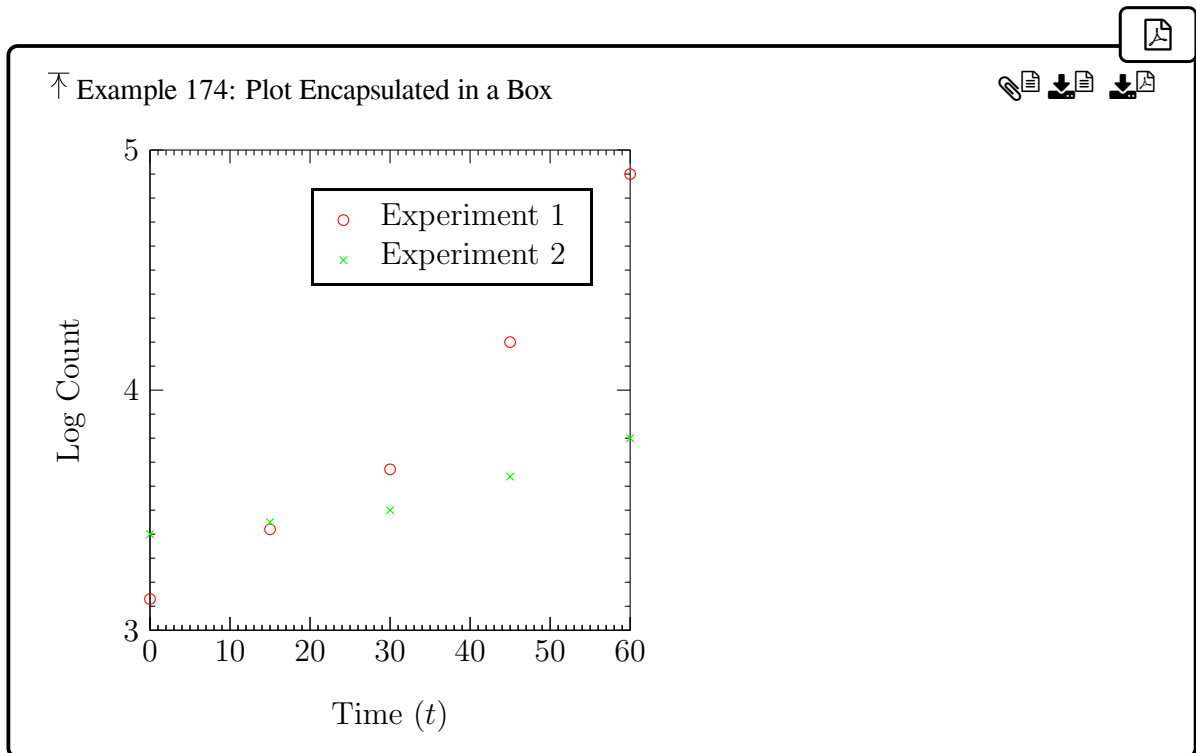
174

```
\DTLplot{growth1}{
  x=Time,
  y={Experiment 1,Experiment 2},
  x-label={Time ($t$)}, y-label={Log Count},
  legend, round=0, minor-ticks,
  min-y=3,max-y=5,y-tick-gap=1,
  tick-label-offset=0pt, box,
  width=2.5in, height=2.5in
}
```

By default, the box will have tick marks that match the axes tick marks. This can be changed with the `box-ticks` option. Example 175 modifies Example 174 so that the box doesn't have tick marks.

175

```
\DTLplot{growth1}{
  x=Time,
```

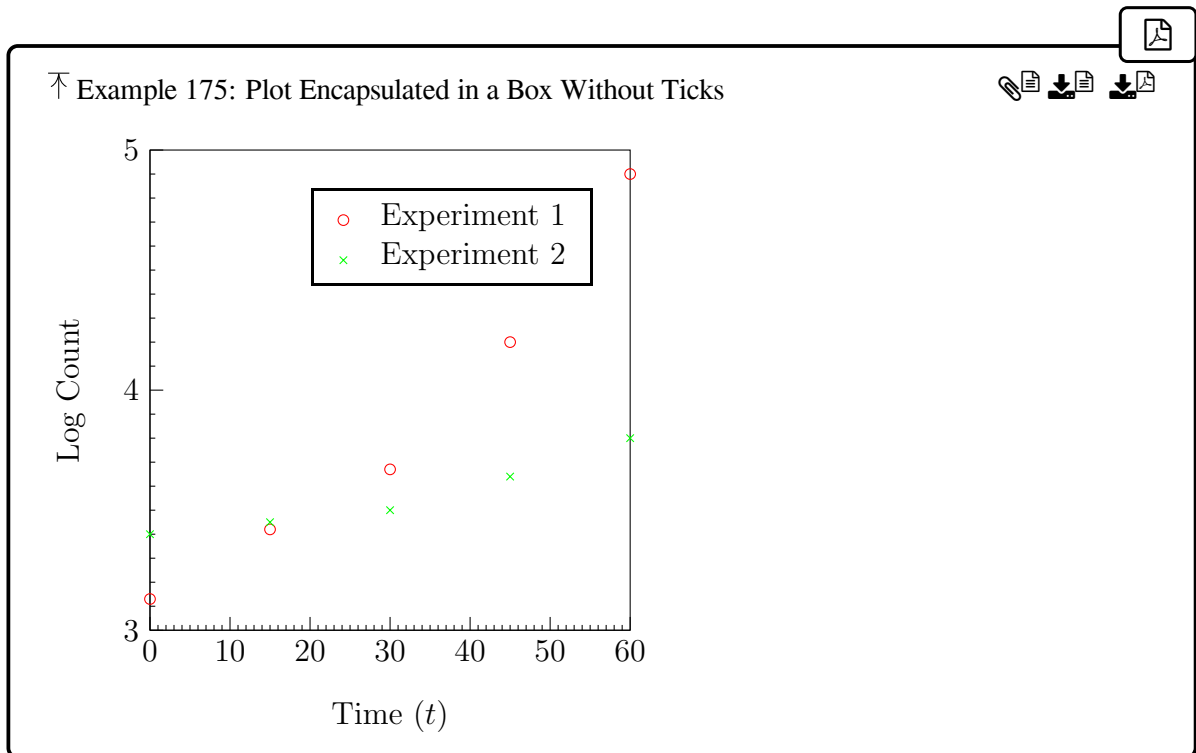
```
y={Experiment 1,Experiment 2},
x-label={Time ($t$)}, y-label={Log Count},
legend, round=0, minor-ticks,
min-y=3,max-y=5,y-tick-gap=1,
tick-label-offset=0pt, box, box-ticks=none,
width=2.5in, height=2.5in
}
```

Note that extending the axes and the `side-axes` option has an effect on the `box` option. See §6.2.4.9.

6.2.4.6. Negative Axes

The examples so far have all had non-negative x and y values. The “xydata” database (see §3.2.11) has some negative values that can be used to demonstrate a graph with axes that stretch from positive to negative. Example 176 uses this data to plot a line graph. The tick direction is changed from the default inwards to outwards, which means that the x ticks extend below the x -axis (instead of above) and the y ticks extend to the left of the y -axis (instead of to the right). The tick labels are rounded to 1 decimal place.

176



```
\DTLplot{xydata}{
  x=X, y={Y},
  tick-dir=out, round=1
  x-label={x}, y-label={y},
  style=lines, width=3in, height=3in
}
```

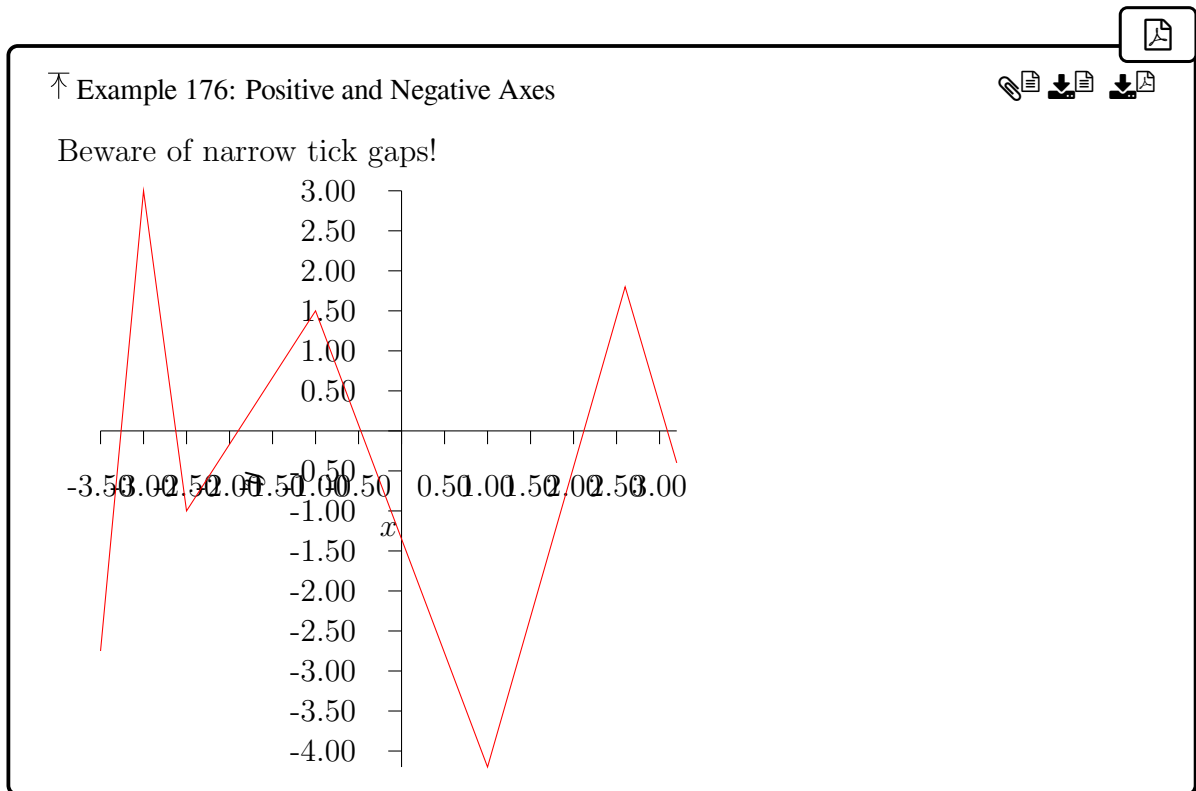
The labels in Example 176 are very untidy as they overlap. This can be addressed by rounding the tick label values and moving the axis labels to the end (see Example 177).

6.2.4.7. Extending the Axes

Example 177 modifies Example 176 so that the x and y axis are extended and the axis labels are moved to the maximum end. Choosing a wider gap for the x ticks can also help reduce the clutter.

177

```
\DTLplot{xydata}{
  x=X, y={Y},
  tick-dir=out, x-tick-gap=1, round-x=0, round-y=1,
  extend-x-axis={0,1}, extend-y-axis={0,0.5},
}
```



```

tick-label-style={font=\small}, tick-label-offset=
0pt,
max-x-label={$x$}, max-y-label={$y$},
style=lines, width=3in, height=3in
}

```

Note that even though the tick label offset has been set to zero, there is still a slight gap between the tick mark and the label. This is due to `tikz`'s default `inner sep` for nodes.

6.2.4.8. Changing the Tick Label Node Style

Examples 176 & 177 used `tick-label-style` to change the font to a smaller size:

```

tick-label-style={font=\small}

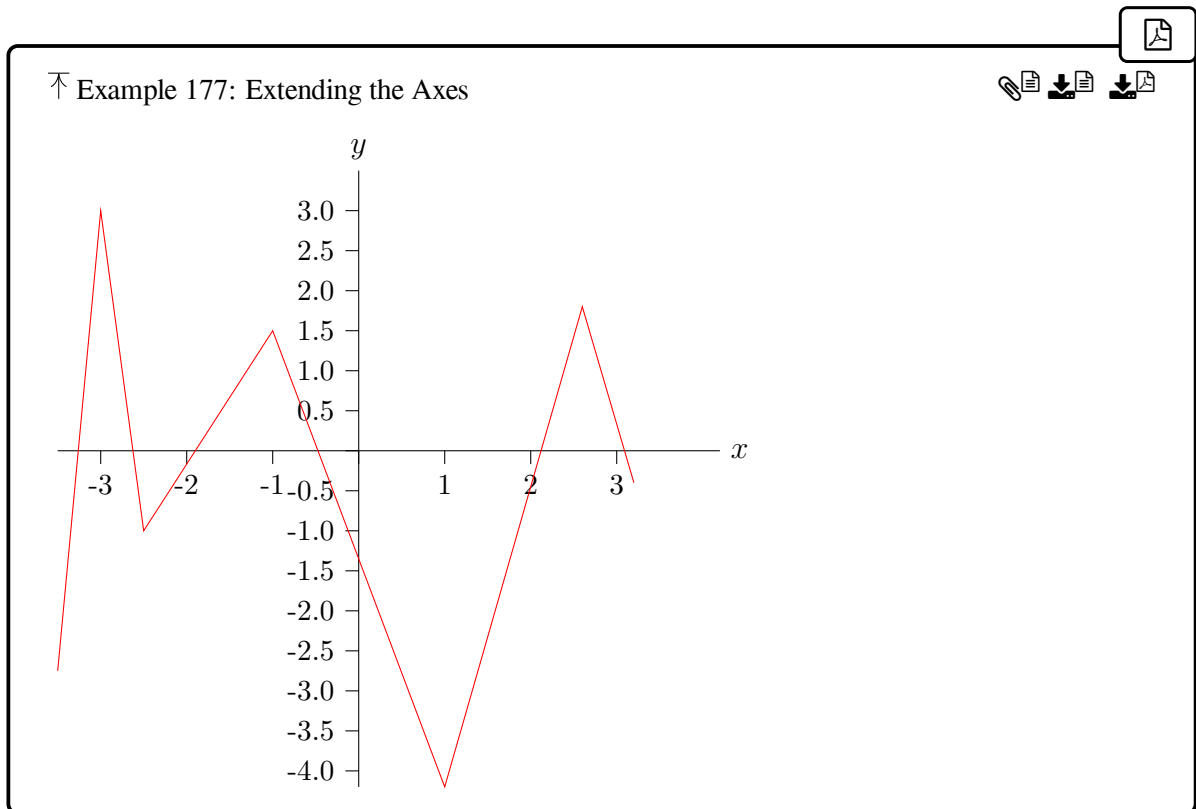
```

This is equivalent to:

```

x-tick-label-style={font=\small},
y-tick-label-style={anchor=east,font=\small}

```



Note that the `y-tick-label-style` includes the node anchor. Example 178 changes these two values separately so that they both use a small font, but the `y`-tick labels are additionally rotated, and the `inner` opt is set to 1pt to bring the `y`-tick label closer to the tick mark.

178

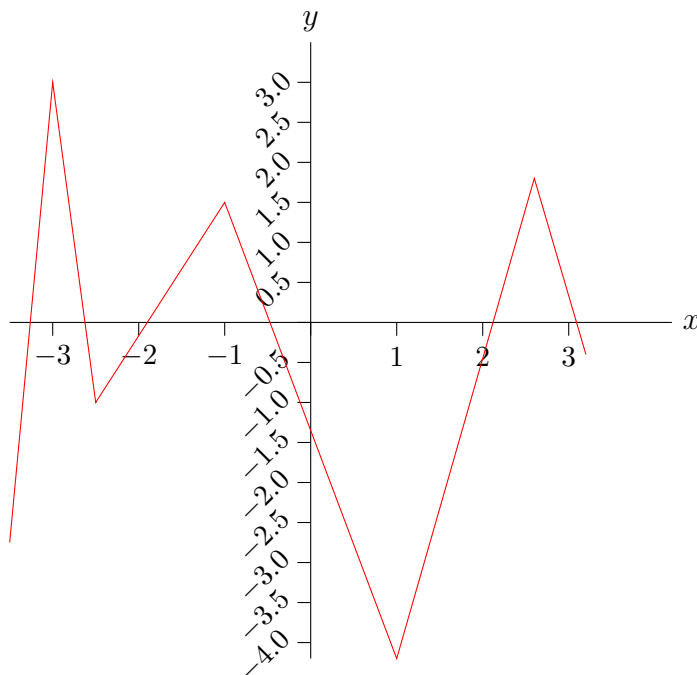
```
\DTLplot{xydata}{
  x=X, y={Y},
  tick-dir=out, x-tick-gap=1, round-x=0, round-y=1,
  extend-x-axis={0,1}, extend-y-axis={0,0.5},
  x-tick-label-style={font=\small},
  y-tick-label-style=
{anchor=south east, inner sep=1pt, rotate=45, font=\small}
,
  tick-label-offset=0pt,
  max-x-label={\mathstrut}, max-y-label={\mathstrut},
  style=lines, width=3in, height=3in
}
```

Additionally, `\DTLplotdisplayticklabel` is redefined to ensure that the tick labels are in math mode:

```
\renewcommand{\DTLplotdisplayticklabel}[1]
{\ensuremath{\#1}}
```

This shows the negative numbers with a correct minus sign rather than a hyphen.

↑ Example 178: Changing the Tick Label Node Style



6.2.4.9. Side Axes

The default `side-axes=false` setting will draw the axes crossing at zero, if zero lies within the plot bounds. If zero doesn't lie within the plot bounds, the axes will be on the side with the x and y -axis meeting at $(x_{\min}, y_{\min})$. That is, the minimum x and y of the bounds plots.

Example 179 plots the “xydata” from the previous examples with `side-axes=true`. This draws the axes at the side (lower left bounds) rather than intersecting at the origin.

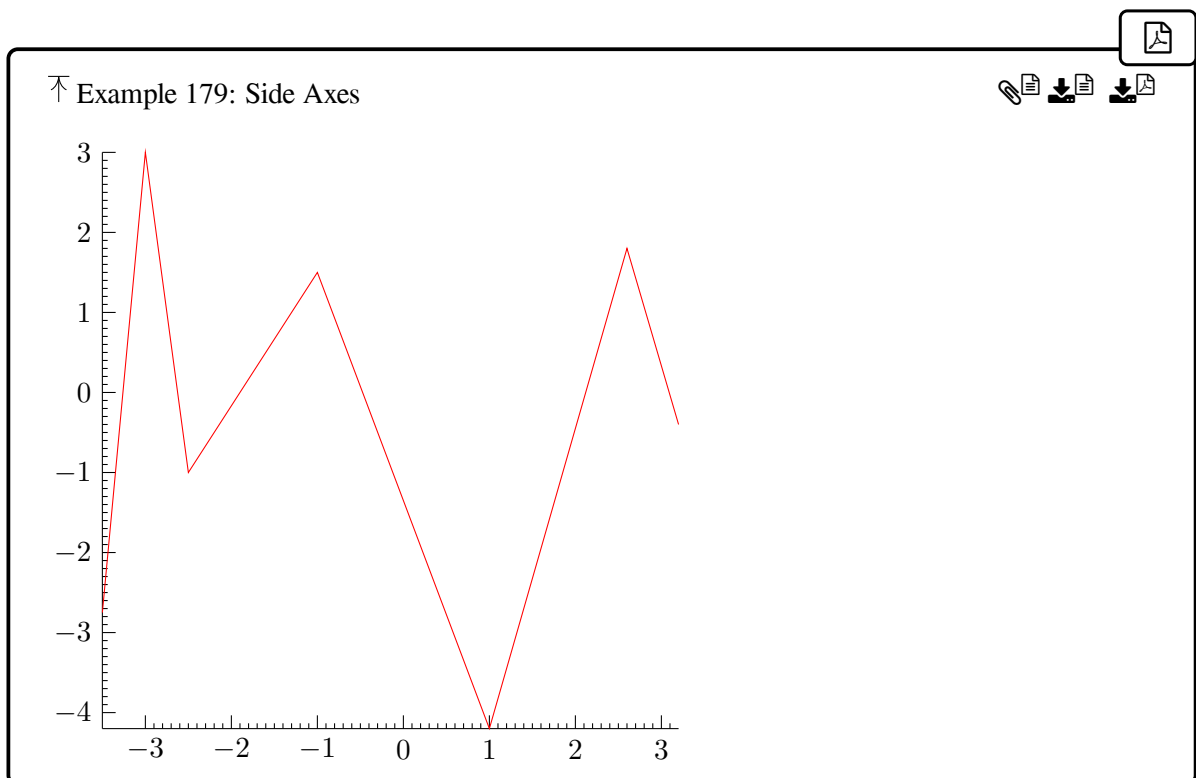
179

```
\DTLplot{xydata}{
  x=X, y={Y},
  tick-gap=1, round=0,
  tick-label-style={font=\small},
  tick-label-offset=0pt,
```

```
minor-ticks, side-axes,
style=lines, width=3in, height=3in
}
```

As with Example 178, the tick label style is redefined to use math mode:

```
\renewcommand{\DTLplotdisplayticklabel}[1]
{\ensuremath{#1}}
```



The “time to growth” data (see §3.2.9) used in earlier examples happens to have 0 as the minimum x value (which in this case is the time t value). The y values are all non-negative, which means that the axes will be on the side regardless of the `side-axes` setting.

In this case, the difference between `side-axes=true` and `side-axes=false` is only noticeable when the axes are extended beyond the minimum bounds and the `box` option is set. This is illustrated in examples 180 & 181 which adapt Example 174 to extend both axes. The difference between them is the `side-axes` setting.

Example 180 has the following options:

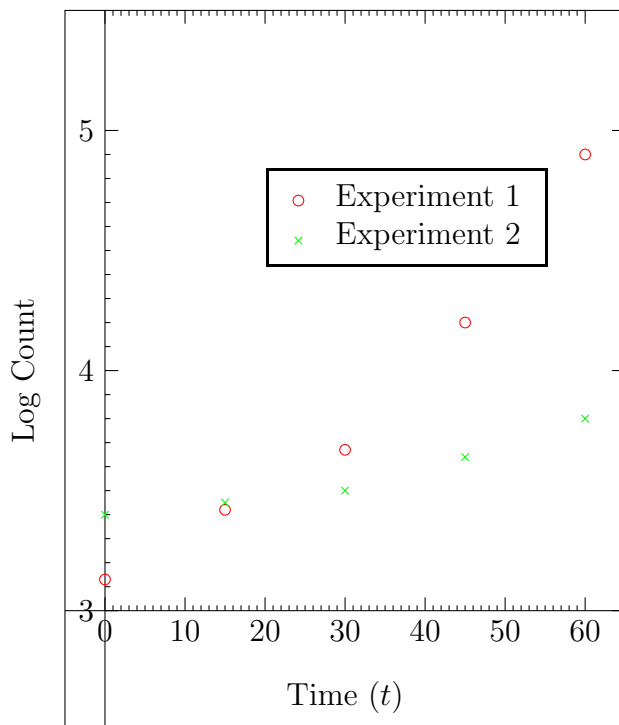
180

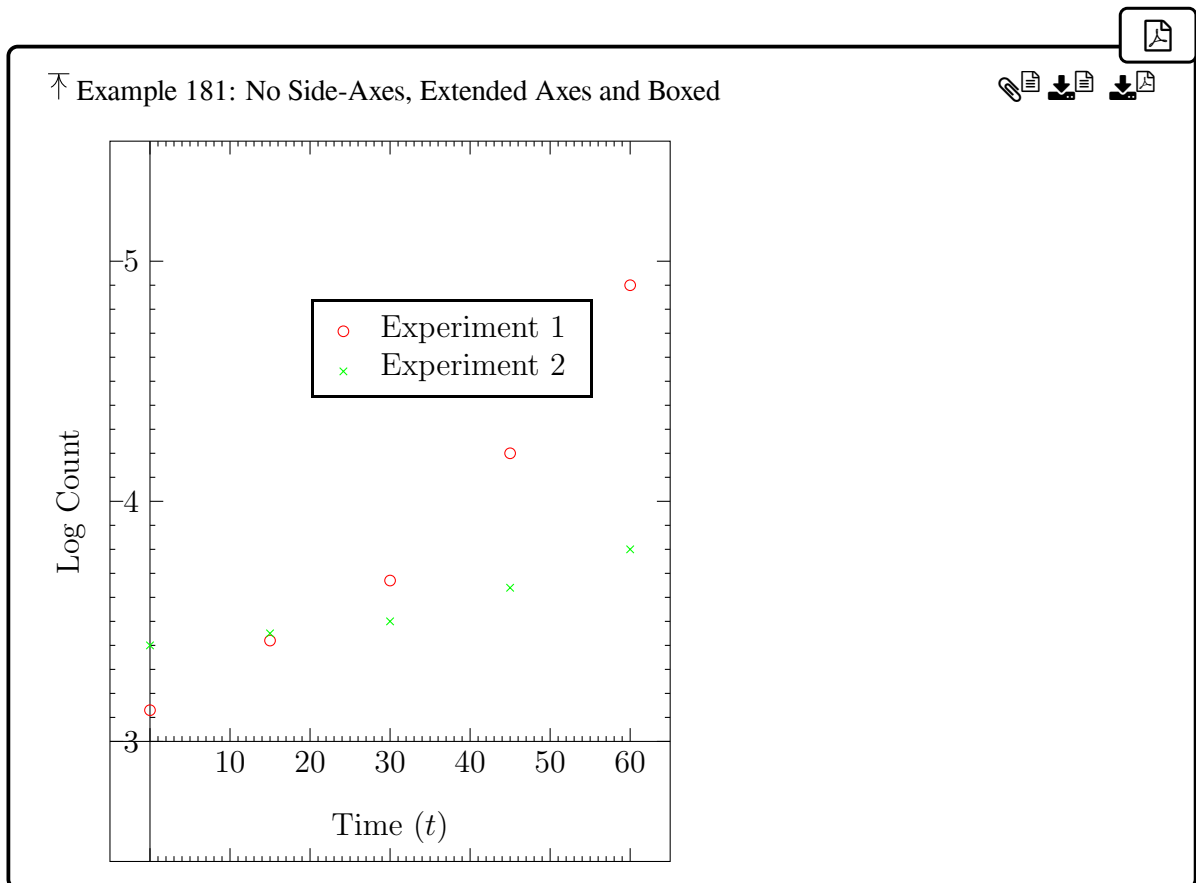
```
\DTLplot{growth1}{
  x=Time,
  y={Experiment 1,Experiment 2},
  x-label={Time ($t$)}, y-label={Log Count},
  legend, round=0, minor-ticks,
  min-y=3,max-y=5,y-tick-gap=1,
  tick-label-offset=0pt, box,
  side-axes,
  extend-x-axis={5},
  extend-y-axis={0.5},
  width=2.5in, height=2.5in
}
```

Example 181 is the same except for `side-axes=false`. With `side-axes=true`, the bottom and left edges of the box don't have tick marks. With `side-axes=false`, all sides of the box have tick marks.

181

↑ Example 180: Side-Axes, Extended Axes and Boxed







The tick marks only go up to the plot bounds. They don't go on the extended parts. This means that the ends of the axes and the corners of the box in examples 180 & 181 don't have tick marks. If you want the tick marks to go all the way to the end then you need to change the plot bounds not the axis extension options.

The x and y axis labels are independent of the axis extensions. In examples 180 & 181, the x label is inside the box because the y extension is large enough to include it, but the y label is outside of the box because the x extension isn't large enough to include it.

6.2.5. Begin and End Hooks

Additional information can be added to the plot with the start and end hooks. Examples 176, 177 & 178 don't show the tick label at $(0,0)$ because of the default `omit-zero-label=auto` setting. Changing this setting to `omit-zero-label=false` would show 0 at both $x = 0$ and $y = 0$. However, this will result in the axes passing through the 0 labels.

Example 182 adds 0 at the origin (shifted below right) by redefining the start hook:

182



```
\renewcommand{\DTLplotatbegintikz}{
  \draw (0,0) node[anchor=north east] {0};
}
```

The end hook is also redefined to show the minimum and maximum y values, for illustrative purposes.

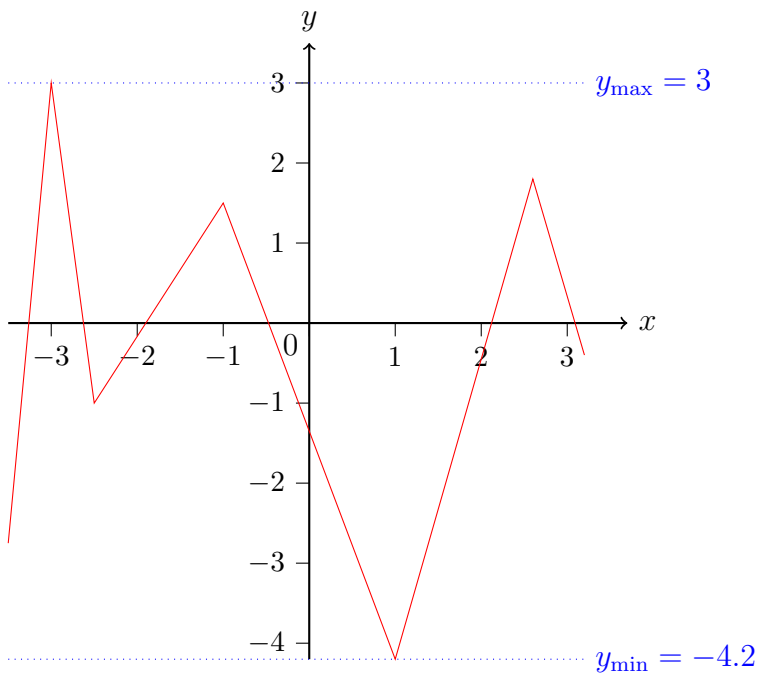


```
\renewcommand{\DTLplotatendtikz}{
  \draw[blue,dotted] (\DTLminX,\DTLminY) -- (\DTLmax-
X, \DTLminY)
  node[right] {$y_{\min} = \DTLminY$};
  \draw[blue,dotted] (\DTLminX,\DTLmaxY) -- (\DTLmax-
X, \DTLmaxY)
  node[right] {$y_{\max} = \DTLmaxY$};
}
```

The plot options are similar to previous examples in §6.2.4 but the x and y tick settings are made the same for simplicity. Both axes are extend at their maximum end and drawn with thick lines and an arrow head at the end.

```
\DTLplot{xydata}{
  x=X, y={Y},
  tick-dir=out, tick-gap=1, round=0,
  extend-x-axis={0,0.5}, extend-y-axis={0,0.5},
  tick-label-style={font=\small},
  tick-label-offset=0pt,
  axis-style={->,thick},
  max-x-label={$x$}, max-y-label={$y$},
  style=lines, width=3in, height=3in
}
```

↑ Example 182: Redefining the Start and End Hooks



6.3. Supplementary Plot Commands

`\DTLplot` internally starts and ends the `tikzpicture` environment. The following hooks are provided to add additional content.

```
\DTLplotatbegintikz
```

This hook occurs at the start of the `tikzpicture` environment.

```
\DTLplotatendtikz
```

This hook occurs at the end of the `tikzpicture` environment.

A transformation will be applied to provide a convenient way of referencing co-ordinates using the plot data units. For example, if your data has x and y values 1234 and 567 then you can reference that point with the `tikz` co-ordinate `(1234, 567)`.

The transformation used by `\DTLplot` can cause plot marks or other content to appear skewed. If necessary, you can reset the transformation matrix with `pgf's` `\pgftransformreset` but make sure that `\DTLplot's` transformation matrix is restored before the end of `\DTLplotatbegintikz`. This can either be done by scoping `\pgftransformreset` or by using:

```
\dataplot_apply_pgfttransform:
```

Note that this command requires $\text{\LaTeX}3$ syntax to be on, which can interfere with `tikz's` syntax.

6.3.1. General Functions and Variables

```
\DTLminX
```

When used within the `\DTLplot` hooks, this command will expand to the minimum x value of the plot bounds. Since the plot code is scoped, it can't be referenced outside of the plot hooks. However, if you use the `plot` action, you can access this value with the `min-x` secondary return value.

```
\DTLminY
```

When used within the `\DTLplot` hooks, this command will expand to the minimum y value of the plot bounds. Since the plot code is scoped, it can't be referenced outside of the plot hooks. However, if you use the `plot` action, you can access this value with the `min-y` secondary return value.

```
\DTLmaxX
```

When used within the `\DTLplot` hooks, this command will expand to the maximum x value of the plot bounds. Since the plot code is scoped, it can't be referenced outside of the plot hooks.

However, if you use the `plot` action, you can access this value with the `max-x` secondary return value.

`\DTLmaxY`

When used within the `\DTLplot` hooks, this command will expand to the maximum y value of the plot bounds. Since the plot code is scoped, it can't be referenced outside of the plot hooks. However, if you use the `plot` action, you can access this value with the `max-y` secondary return value.

The total number of keys provided in the x and y lists and the total number of databases can be obtained with the following macros, which simply use `\seq_count:N` or `\clist_count:N` on the internal variables used to store those lists.

`\dataplot_x_key_count:`

May be used with `\DTLplot` hooks to return the total number of items given in the x list. If you use the `plot` action, this value can also be accessed after the plot using the `x-count` secondary return value.

`\dataplot_y_key_count:`

May be used with `\DTLplot` hooks to return the total number of items given in the y list. If you use the `plot` action, this value can also be accessed after the plot using the `y-count` secondary return value.

`\dataplot_db_count:`

May be used with `\DTLplot` hooks to return the total number of database names. If you use the `plot` action, this value can also be accessed after the plot using the `name-count` secondary return value.

6.3.2. Stream Functions and Variables

`\DTLaddtopplotlegend{<marker>}{<line style>}{<text>}`

This command is used by `\DTLplot` to add the current plot stream to the legend. The `<marker>` is the marker code and may be empty or `\relax` to indicate no marker. The `<line style>` is the line style code and may be empty or `\relax` to indicate no line. The `<text>` is the legend text associated with the current plot stream. Note that `<marker>` and `<line style>` should include the associated colour change, if applicable. The marker or line code is encapsulated in a `pgfpicture` environment with the line style scoped so that any colour change doesn't affect the marker.

If you want to redefine `\DTLaddtoplotlegend` to change the style of the legend, you will need to switch to $\text{\LaTeX}$ 3 syntax. The legend is constructed by appending content to:

`\l_dataplot_legend_tl`

This is a token list variable which by the end of `\DTLplot` (but before `\DTLplotatendtikz`) should contain the code to typeset the plot legend.

At the end of each stream, the legend text is obtained and, if not empty, `\DTLaddtoplotlegend` is used to add the current stream information to the legend. If `legend-labels` has been used, the legend text is obtained from the corresponding item in that list. Otherwise it's obtained using the following command.

`\dataplot_get_default_legend:Nnnnnn <tl-var> {<db-index>}
{<db-name>} {<x-key>} {<y-key>}`

Gets the default legend label (the text supplied to `\DTLaddtoplotlegend`) when there is no corresponding label provided by `legendlabels`, and stores it in the token list variable `<tl-var>`. The `<x-key>` and `<y-key>` arguments are the column keys for the current x and y plot stream. The default behaviour for each row of the legend depends on the number of databases and x and y variables.

The token list variable is cleared before passing it to this function. If the function is redefined so that it doesn't modify the token list variable then the current plot stream won't be added to the legend.

The default behaviour of `\dataplot_get_default_legend:Nnnnnn` and its associated commands is described in more detail in §6.3.3.

If you need to access other information about the current plot stream, you can do so with the following integer variables, but bear in mind that while `ldataplotstreamindexint`, `\l_dataplot_x_key_int` and `\l_dataplot_y_key_int` can be used in `\dataplot_get_default_legend:Nnnnnn` they shouldn't be added to the token list variable provided in the first argument as their values will have changed by the time the token list variable content is added to the input stream. This means they also shouldn't be referenced within commands like `\DTLplotlegendx` which are added unexpanded to the token list variable. This is why their expanded values are passed in the optional arguments of the legend hooks.

`\l_dataplot_stream_index_int`

Integer variable that keeps track of the current stream index (starting from 1). In `\dataplot_get_default_legend:Nnnnnn`, this can be used to reference current stream index. In the end hook, it may be used to reference the final stream index, which will be the total number of streams. (In the start hook, it will be zero.) The `plot` action assigns the primary return value (and the secondary `stream-count`) value to the value of this integer.

`\l_dataplot_x_key_int`

Used by `\DTLplot` to keep track of the current x column key. That is, the index (starting from 1) of the current item in the x comma-separated list.

`\l_dataplot_y_key_int`

Used by `\DTLplot` to keep track of the current y column key. That is, the index (starting from 1) of the current item in the y comma-separated list.

6.3.3. Post-Stream Hooks

`\DTLformatlegend{legend code}`

The legend code is encapsulated with `\DTLformatlegend`. The default is to draw the legend in a colour box with a white background and a black border. Note that this will cause the legend to blank out the region of the plot drawn behind it. The default definition simply does:

```
\setlength{\fboxrule}{1.1pt}% sets the border width
\fcolorbox{black}{white}{legend}
```

`\DTLcustomlegend{legend code}`

This command is used to position the legend with the `legend=custom` setting. By default, this simply does:

```
\node {\DTLformatlegend{legend code}};
```

This should be redefined as applicable to position the legend in the required place. The command should expand to valid tikz (or pgf) code. See Example 164.

`\dataplot_legend_add_begin:`

Adds the beginning legend code to `\l_dataplot_legend_tl`. By default, this inserts the beginning of a tabular environment:

```
\cs_new:Nn \dataplot_legend_add_begin:
{
  \tl_put_left:Nn \l_dataplot_legend_tl
    { \begin { tabular } { c l } }
}
```

```
\dataplot_legend_add_end:
```

Adds the end legend code to `\l_dataplot_legend_tl`. By default, this inserts the end of a tabular environment:

```
\cs_new:Nn \dataplot_legend_add_end:
{
  \tl_put_right:Nn \l_dataplot_legend_tl
    { \end { tabular } }
}
```

As described in §6.3.2, the default legend text is obtained with `\dataplot_get_default_legend:Nnnnn`. This constructs the legend text according to the number of items in the database list $\langle db\text{-}list \rangle$, the x list, and the y list. The applicable commands (described below) are added unexpanded to the legend token list variable but their arguments are expanded to ensure that the correct values are present when the legend is finally drawn.

In the following legend commands, $\langle db\text{-}index \rangle$ is the index of the database name in the $\langle db\text{-}list \rangle$ supplied to `\DTLplot`, $\langle x\text{-}index \rangle$ is the index of the $\langle x\text{-}key \rangle$ within the x list, and $\langle y\text{-}index \rangle$ the index of the $\langle y\text{-}key \rangle$ within the y list. This is not the same as the column index (which can be obtained with `\dtlcolumnindex{\langle db\text{-}name \rangle}{\langle key \rangle}`). For example, with the following:

```
\DTLplot{growthdata,growthdata2}{
  x={Exp1Time,Exp2Time,Exp3Time},
  y={Exp1Count,Exp2Count,Exp3Count},
  legend
}
```

The $\langle db\text{-}index \rangle$ of “growthdata2” is 2 as it’s the second in the list `growthdata, growthdata2`, the $\langle x\text{-}index \rangle$ of “Exp1Time” is 1 as it’s the first in the x list (coincidentally that also happens to be its column index), and the $\langle y\text{-}index \rangle$ of “Exp3Count” is 3 as it’s the third in the y list (but its column index is 6). Where the indexes are optional, the default value is 0.

If data is obtained from more than one database and more than one column for the x variable, then the legend row will be in the form:

```
⟨name⟩ ⟨x-header⟩ / ⟨y-header⟩
```

If there is only one database, the name will be omitted:

```
⟨x-header⟩ / ⟨y-header⟩
```

If there is only one x variable then the $\langle x\text{-header} \rangle$ and separator will be omitted, so with more than one database and more than one y variable:

```
⟨name⟩ ⟨y-header⟩
```

If there is only one database and only one x variable, then only the $\langle y\text{-header} \rangle$ will be used if the y option contained more than one key otherwise only the $\langle name \rangle$ will be shown.

The above $\langle name \rangle$, $\langle x\text{-header} \rangle$, $\langle y\text{-header} \rangle$ and slash separator are obtained with the commands described below. However, bear in mind that although `\dataplot_get_default_legend:Nnnnn` is used at the end of each plot stream (and so, can reliably access variables such as `\l_dataplot_stream_index_int`), the legend commands below are only expanded when the legend is drawn after all the streams have been plotted, at which point it's too late to access stream variables.

```
\DTLplotlegendname[⟨db-index⟩]{⟨db-name⟩}
```

This produces the $\langle name \rangle$ part of the legend (the optional argument is ignored), and which defaults to the database name $\langle db\text{-name} \rangle$ unless you have supplied a mapping for $\langle db\text{-name} \rangle$ with:

```
\DTLplotlegendsetname{⟨db-name⟩}{⟨text⟩}
```

This will make `\DTLplotlegendname` use $\langle text \rangle$ instead of $\langle db\text{-name} \rangle$. This command is simply a user-level interface that puts the mapping in the following property list:

```
\l_dataplot_legend_names_prop
```

If you have $\text{\LaTeX}3$ syntax on, you can access this property list directly. A token list variable is defined specifically for accessing values in this property list:

```
\l_dataplot_legend_name_tl
```

If the legend includes $\langle name \rangle$ and at least $\langle y\text{-header} \rangle$, `\DTLplotlegendname` will be followed by:


```
\DTLplotlegendnamesep
```

initial: `\`

This defaults to a space.

```
\DTLplotlegendxy[ $\langle db-index \rangle$ ]{ $\langle db-name \rangle$ }[ $\langle x-index \rangle$ ]{ $\langle x-key \rangle$ }[ $\langle y-index \rangle$ ]{ $\langle y-key \rangle$ }
```

This command is used to produce the $\langle x-header \rangle$ / $\langle y-header \rangle$ part where there are multiple x column keys. The default definition is:

```
\DTLplotlegendx[ $\langle db-index \rangle$ ]{ $\langle db-name \rangle$ }[ $\langle x-index \rangle$ ]{ $\langle x-key \rangle$ }%
\DTLplotlegendxysep
\DTLplotlegendy[ $\langle db-index \rangle$ ]{ $\langle db-name \rangle$ }[ $\langle y-index \rangle$ ]{ $\langle y-key \rangle$ }
```

```
\DTLplotlegendx[ $\langle db-index \rangle$ ]{ $\langle db-name \rangle$ }[ $\langle x-index \rangle$ ]{ $\langle x-key \rangle$ }
```

The $\langle x-header \rangle$ text, which defaults to the header for the column identified by the given key (the optional arguments are ignored) unless you have supplied a mapping with:

```
\DTLplotlegendsetxlabel{ $\langle x-key \rangle$ }{ $\langle text \rangle$ }
```

This will make `\DTLplotlegendx` use $\langle text \rangle$ instead of the column header for the given column key. This command is simply a user-level interface that puts the mapping in the following property list:

```
\l_dataplot_legend_xlabel_prop
```

If you have L^AT_EX3 syntax on, you can access this property list directly. A token list variable is defined specifically for accessing values in this property list:

```
\l_dataplot_legend_xlabel_tl
```

```
\DTLplotlegendxysep
```

The separator between the $\langle x-header \rangle$ and $\langle y-header \rangle$. This defaults to a slash / with a space on either side.

```
\DTLplotlegendy[⟨db-index⟩]{⟨db-name⟩}[⟨y-index⟩]{⟨y-key⟩}
```

The $\langle y\text{-header} \rangle$ text, which defaults to the header for the column identified by the given key (the optional arguments are ignored) unless you have supplied a mapping with:

```
\DTLplotlegendsetylabel{⟨y-key⟩}{⟨text⟩}
```

This will make `\DTLplotlegendy` use $\langle text \rangle$ instead of the column header for the given column key. This command is simply a user-level interface that puts the mapping in the following property list:

```
\l_dataplot_legend_ylabels_prop
```

If you have $\text{\LaTeX}3$ syntax on, you can access this property list directly. A token list variable is defined specifically for accessing values in this property list:

```
\l_dataplot_legend_ylabel_tl
```

6.4. Conditionals

The following conditionals are defined by *dataplot*. These are $\text{\TeX}$ style `\if⟨name⟩` conditionals that can be switched on or off with the corresponding `\⟨name⟩true` and `\⟨name⟩false` commands. There are corresponding settings that do this, such as `box` and `grid`.

```
\ifDTLbox ⟨true⟩\else ⟨false⟩\fi initial: \iffalse
```

Determines whether or not to enclose the plot in a box. This conditional may be changed with the `box` setting.

```
\ifDTLgrid ⟨true⟩\else ⟨false⟩\fi initial: \iffalse
```

Determines whether or not to draw a grid. This conditional may be changed with the `grid` setting.

```
\ifDTLshowlines ⟨true⟩\else ⟨false⟩\fi initial: \iffalse
```

Determines whether or not to use plot lines. This conditional is changed by the `style` setting.

```
\ifDTLshowmarkers <true>\else <false>\fi          initial: \iftrue
```

Determines whether or not to use plot markers. This conditional is changed by the `style` setting.

The `style` setting doesn't provide an option which switches both `\ifDTLshowlines` and `\ifDTLshowmarkers` to false, since that combination makes no sense. If you explicitly change these conditionals, make sure you don't set them both to false.

```
\ifDTLxaxis <true>\else <false>\fi          initial: \iftrue
```

Determines whether or not to display the x axis. This conditional is changed by the `axes` setting.

```
\ifDTLxminortics <true>\else <false>\fi      initial: \iffalse
```

Determines whether or not to display the x axis minor tick marks. This conditional may be changed with the `x-minor-ticks` setting.

```
\ifDTLxticsin <true>\else <false>\fi         initial: \iftrue
```

Determines whether or not to display the x axis tick marks inwards or outwards (if they should be displayed). This conditional may be changed with the `x-tick-dir` setting, where `x-tick-dir=in` is equivalent to setting the conditional to true and `x-tick-dir=out` is equivalent to setting the conditional to false.

```
\ifDTLxtics <true>\else <false>\fi          initial: \iftrue
```

Determines whether or not to display the x axis tick marks. This conditional may be changed with the `x-ticks` setting.

```
\ifDTLyaxis <true>\else <false>\fi          initial: \iftrue
```

Determines whether or not to display the y axis. This conditional is changed by the `axes` setting.

```
\ifDTLyminortics <true>\else <false>\fi      initial: \iffalse
```

Determines whether or not to display the y axis minor tick marks. This conditional may be changed with the `y-minor-ticks` setting.

`\ifDTLyticsin <true>\else <false>\fi` *initial: \iftrue*

Determines whether or not to display the y axis tick marks inwards or outwards (if they should be displayed). This conditional may be changed with the `y-tick-dir` setting, where `y-tick-dir=in` is equivalent to setting the conditional to true and `y-tick-dir=out` is equivalent to setting the conditional to false.

`\ifDTLytics <true>\else <false>\fi` *initial: \iftrue*

Determines whether or not to display the y axis tick marks. This conditional may be changed with the `y-ticks` setting.

6.4.1. Lengths

The following length registers are defined by *dataplot*. They may be changed with `\setlength` or, if available, via a setting.

`\DTLlegendxoffset` *initial: 10pt*

The x offset from the border of the plot and the legend. This register is changed by the `legend-offset` setting.

`\DTLlegendyoffset` *initial: 10pt*

The y offset from the border of the plot and the legend. This register is changed by the `legend-offset` setting.

`\DTLminminortickgap` *initial: 5pt*

The suggested minimum distance between minor tick marks.

`\DTLminorticklength` *initial: 2pt*

The minor tick mark length.

`\DTLmintickgap` *initial: 20pt*

The suggested minimum distance between tick marks when the gap is not specified. Ignored for the applicable axis if `x-tick-gap`, `y-tick-gap`, `x-tick-points` or `y-tick-points` are specified.

`\DTLplotheight`*initial: 4in*

The plot height. This register is set by the `height` option.

`\DTLplotwidth`*initial: 4in*

The plot width. This register is set by the `width` option.

`\DTLticklabeloffset`*initial: 8pt*

The offset from the axis to the tick label.

`\DTLticklength`*initial: 5pt*

The tick mark length.

6.4.2. Counters

The following counters are defined by *dataplot*. They can be globally changed with `\set-counter` or locally changed with the corresponding setting.

`DTLplotroundXvar`*No*

The rounding value for default x tick mark labels. This value is ignored if the labels are explicitly provided with `x-tick-labels` or if `x-ticks=false`. The corresponding setting is `round-x`.

`DTLplotroundYvar`*No*

The rounding value for default y tick mark labels. This value is ignored if the labels are explicitly provided with `y-tick-labels` or if `y-ticks=false`. The corresponding setting is `round-y`.

6.4.3. Macros

`\DTLplotlinecolors`

The expansion text of this command should be a comma-separated list of colours (suitable for use in the argument of `\color`). The `line-colors` option redefines this command. The default definition is:

```
\newcommand{\DTLplotlinecolors}
{red, green, blue, yellow,
magenta, cyan, orange, black, gray}
```

Note that spaces and empty items will be removed. Use `{}` or `\relax` to indicate the default colour.

`\DTLplotlines`

The expansion text of this command should be a comma-separated list of line styles. The `lines` option redefines this command. The default definition is:

```
\newcommand{\DTLplotlines}{
  \pgfsetdash{{0pt},{0pt}},% solid line
  \pgfsetdash{{10pt}{5pt}}{0pt},
  \pgfsetdash{{5pt}{5pt}}{0pt},
  \pgfsetdash{{1pt}{5pt}}{0pt},
  \pgfsetdash{{5pt}{5pt}1pt5pt}{0pt},
  \pgfsetdash{{1pt}{3pt}}{0pt}
}
```

Note that spaces and empty items will be removed. Use `{}` or `\relax` to indicate the line should be omitted for the corresponding plot stream.

`\DTLplotmarkcolors`

The expansion text of this command should be a comma-separated list of colours (suitable for use in the argument of `\color`). The `mark-colors` option redefines this command. The default definition is:

```
\newcommand{\DTLplotmarkcolors}{red, green, blue,
yellow, magenta, cyan, orange, black, gray}
```

Note that spaces and empty items will be removed. Use `{}` or `\relax` to indicate the default colour.

`\DTLplotmarks`

The expansion text of this command should be a comma-separated list of plot marks. The `marks` option redefines this command. The default definition is:

```
\newcommand*{\DTLplotmarks}{
  \pgfuseplotmark{o},
  \pgfuseplotmark{x},
  \pgfuseplotmark{+},
  \pgfuseplotmark{square},
  \pgfuseplotmark{triangle},
  \pgfuseplotmark{diamond},
  \pgfuseplotmark{pentagon},
  \pgfuseplotmark{asterisk},
  \pgfuseplotmark{star}
}
```

Note that spaces and empty items will be removed. Use {} or \relax to indicate the marker should be omitted for the corresponding plot stream.

```
\DTLplotdisplayticklabel{<text>}
```

Both \DTLplotdisplayXticklabel and \DTLplotdisplayYticklabel are initially defined to simply use \DTLplotdisplayticklabel to encapsulate the tick labels. This means that if you want the same formatting for both x and y tick labels, you can redefine this command instead of redefining both \DTLplotdisplayXticklabel and \DTLplotdisplayYticklabel. For example:

```
\renewcommand{\DTLplotdisplayticklabel}[1]
{\ensuremath{#1}}
```

Alternatively, you can use the `tick-label-style` option to change the node style (see Example 178).

Note that if the tick labels are automatically generated from the data (rather than by specifying them with `x-tick-labels` or `y-tick-labels`) then the argument will be a datum item where the string part is the formatted number. If you prefer a plain number you can redefine \DTLplotdisplayticklabel to use \datatool_datum_value:Nnnnn (which requires L^AT_EX3 syntax on).

```
\DTLplotdisplayXticklabel{<text>}
```

This command encapsulates the x -tick labels.

```
\DTLplotdisplayYticklabel{<text>}
```

This command encapsulates the y -tick labels.



```
\DTLXAxisStyle
```

initial: –

This command should expand to the tikz options that set the line style for the x -axis. The `x-axis-style` and `axis-style` options redefine this command.



```
\DTLYAxisStyle
```

initial: –

This command should expand to the tikz options that set the line style for the y -axis. The `y-axis-style` and `axis-style` options redefine this command.



```
\DTLmajorgridstyle
```

initial: color=gray, –

This command should expand to the tikz options that set the line style for the major grid. The `major-grid-style` option redefines this command.



```
\DTLminorgridstyle
```

initial: color=lightgray, very thin

This command should expand to the tikz options that set the line style for the minor grid. If this command is redefined to nothing, the minor grid won't be drawn. The `minor-grid-style` option redefines this command.

6.5. Adding to a Plot Stream at the Start or End

These commands are provided for use in the start and end hooks to add extra plot streams. However, they are largely redundant now that `\DTLplot` can take a comma-separated list of y values.



```
\dtlplotohandlermark{<mark code>}
```

Only for use within the hooks, this command essentially does:

```
\pgftransformreset
\pgfplotohandlermark{<mark code>}
```

and ensures that `\dataplot_apply_pgfttransform:` occurs after the hook. You can use `\pgfplotohandlermark` directly but remember that the plot marks will be distorted by the `\DTLplot` transformation matrix if it is still in effect.



```
\DTLplotstream[<condition>]{<db-name>}{<x-key>}{<y-key>}
```


6. Scatter and Line Plots (*dataplot* package)

Adds data from the columns identified by the keys $\langle x\text{-key} \rangle$ and $\langle y\text{-key} \rangle$ in the database $\langle db\text{-name} \rangle$ to the current pgf plot stream. The $\langle condition \rangle$ argument is as for `\DTLplot`.

7. Converting a $\text{BIB}\text{T}\text{E}\text{X}$ database into a datatool database (databib package)



```
\usepackage[\langle options \rangle]{databib}
```

The databib package was added to the datatool bundle in version 1.01 (2007-08-17) and provides a way of converting $\text{BIB}\text{T}\text{E}\text{X}$ data into a datatool database. Note that this still requires the document to be compiled with `bibtex`.



If you want a flexible system for managing bibliographies, consider using `biblatex` and `biber`, which is far more efficient than using `databib`. I don't intend to develop `databib` any further as it's far quicker to use `biblatex` and `biber`.

The `databib` package was rewritten in version 3.0 to use $\text{L}\text{T}\text{E}\text{X}3$ commands. The `xkeyval` package has been dropped. Rollback to version 2.32 is available:



```
\usepackage{databib}[=v2.32]
```

Note that if `datatool` hasn't already been loaded, this will also apply rollback to `datatool`. Problems may occur if a newer release of `datatool` has already been loaded.

The `databib` package works by using $\text{BIB}\text{T}\text{E}\text{X}$ with the custom `databib.bst` style file to create a `bbl` file that contains commands to construct the datatool database (as opposed to the more typical behaviour of writing the bibliography typesetting commands to the `bbl` file). The commands `\DTLloadbbl` or `\DTLloadmbbl` are provided to set the bibliography style to `databib`, identify the required `bib` file (or files), create the database and input the `bbl` file. See §7.4 for further details.

Once the database has been created, it can then be sorted using `\DTLsortdata` or `\DTLsort`. Bear in mind that a column may not be created if there was no selected reference with the corresponding field set in the `bib` file.

Whilst the database can be iterated over with `\DTLmapdata` or `\DTLforeach`, a wrapper command `\DTLforeachbibentry` is provided (see §7.8) for custom loops, and `\DTLbibliography` (see §7.6) may be used to display the contents of the database in a format similar the basic `plain`, `abbrv` and `alpha bibtex` styles.



An “entry” in the context of databib, means the row of data in the database corresponding to the information obtained from a bib entry. An entry field is one of the special databib database column keys, and the field value is the value for that column in the current row. The column keys that should always be set are `CiteKey` (the label used in `\cite` that uniquely identifies the entry) and `EntryType` (the BibTeX entry type in lowercase, without the leading @). The other fields are listed in §7.2.2.

7.1. Package Options

The databib package will automatically load datatool, if it has not already been loaded. The databib package has the following options:



style=*⟨name⟩*

This option sets the bibliography style to *⟨name⟩*, which may be one of: `plain`, `abbrv` or `alpha` (see §7.7). The style can also be set with `\DTLbibliographystyle` after databib has loaded.



auto=*⟨value⟩*

default: true; initial: false

If true, `bibtex` will be run from the shell escape. When this is set, `\DTLloadbbl` may only be used in the preamble.



These package options can't be used in `\DTLsetup`. There is no sub-key corresponding to databib.

All other options will be passed to datatool, if it hasn't already been loaded, or will be set with `\DTLsetup` (if allowed) otherwise.

7.2. BibTeX: An Overview

This document assumes that you have at least some passing familiarity with BibTeX, but here follows a brief refresher.

BibTeX is an external application used in conjunction with L^AT_EX. When you run BibTeX, you need to specify the name of the document's auxiliary (`aux`) file. BibTeX then reads this file and looks for the commands `\bibstyle` (which indicates which bibliography style (`bst`) file to load), `\bibdata` (which indicates which bibliography database (`bib`) files to load) and `\citation` (produced by `\cite` and `\nocite`, which indicates which entries should be

7. Converting a BibTeX database into a datatool database (databib package)

included in the bibliography). BibTeX then creates a file with the extension `bbl` which contains the bibliography, formatted according to the layout defined in the bibliography style file.

In general, given a document called, say, `mydoc.tex`, you will have to perform the following steps to ensure that the bibliography and all citations are up-to-date (replace `latex` with `pdflatex`, `xelatex` or `lualatex`, as applicable):

1. Run L^AT_EX:

```
latex mydoc
```

This writes the citation information to the auxiliary file. The bibliography currently doesn't exist, so it isn't displayed. Citations will appear in the document as `[?]` since the internal cross-references don't exist yet (similar to `??` which marks unknown references).

2. Run BibTeX:

```
bibtex mydoc
```

This reads the auxiliary file, and creates a file with the extension `bbl` which typically contains the typeset bibliography.

3. Run L^AT_EX:

```
latex mydoc
```

Now that the `bbl` file exists, the bibliography can be input into the document. The internal cross-referencing information for the bibliography can now be written to the auxiliary file.

4. Run L^AT_EX:

```
latex mydoc
```

The cross-referencing information can be read from the auxiliary file.

7.2.1. BibTeX database

The bibliographic data required by BibTeX must be stored in a file with the extension `bib`, where each entry is stored in the form:

```
@⟨entry-type⟩ {⟨cite-key⟩,
  ⟨field⟩ = ⟨value⟩,
  ⋮
  ⟨field⟩ = ⟨value⟩
}
```

The $\langle value \rangle$ may be delimited with braces or double-quotes or may be a number or a predefined string. For example:

```
@STRING{TUGBOAT = {TUGboat}}
@article{brown2022,
  author = "Mary-Jane Brown",
  title = {An interesting paper},
  journal = TUGBOAT,
  year = 2022
}
```

The above first defines a string constant (“TUGBOAT”) that may be used instead of a literal string in the field value. Note that the constant isn’t delimited by braces or double-quotes when referenced in the field value.

The entry type, given by $\langle entry-type \rangle$ above, indicates the type of document. This depends on which ones are supported by the `bst` bibliography style file, but the standard ones are: `article`, `book`, `booklet`, `inbook`, `incollection`, `inproceedings` (with synonym `conference`), `manual`, `mastersthesis`, `misc`, `phdthesis`, `proceedings`, `techreport` or `unpublished`.

The $\langle cite-key \rangle$ above is a unique label identifying this entry, and is the label used in the argument of `\cite` or `\nocite`. The available fields depends on the entry type (and the `bst` file), for example, the field `journal` is required for the `article` entry type, but is ignored for the `inproceedings` entry type. The standard fields are: `address`, `author`, `booktitle`, `chapter`, `edition`, `editor`, `howpublished`, `institution`, `journal`, `key`, `month`, `note`, `number`, `organization`, `pages`, `publisher`, `school`, `series`, `title`, `type`, `volume` and `year`.



The `databib` package was developed in 2007 and designed to work with BibTeX, which is now quite old and has little localisation support. The newer `biber`, which was initially released in 2009 and is used with `biblatex`, provides more flexible support and a less rigid name syntax.

With BibTeX, author and editor names must be entered in one of the following ways:

1. $\langle First\ names \rangle \langle von\ part \rangle \langle Surname \rangle, \langle Jr\ part \rangle$

The $\langle von\ part \rangle$ is optional and is identified by the name(s) starting with lowercase letters.

7. Converting a BibTeX database into a datatool database (databib package)

The final comma followed by *⟨Jr part⟩* is also optional. Examples:

```
author = "Henry James de Vere"
```

In the above, the first names are Henry James, the “von part” is “de” and the surname is “Vere”. There is no “junior part”.

```
author = "Mary-Jane Brown, Jr"
```

In the above, the first name is Mary-Jane, there is no von part, the surname is Brown and the junior part is Jr.

```
author = "Peter {Murphy Allen}"
```

In the above, the first name is Peter, and the surname is Murphy Allen. Note that in this case, the surname must be grouped, otherwise Murphy would be considered part of the forename.

```
author = "Maria Eliza {\MakeUppercase{d}e La} Cruz"
```

In the above, the first name is Maria Eliza, the von part is De La, and the surname is Cruz. In this case, the von part starts with an uppercase letter, but specifying:

```
author = "Maria Eliza De La Cruz"
```

would make BibTeX incorrectly classify “Maria Eliza De La” as the first names, and the von part would be empty. Since BibTeX doesn’t understand LaTeX commands, using `{\MakeUppercase{d}e La}` will trick BibTeX into thinking that it starts with a lowercase letter.

2. *⟨von part⟩* *⟨Surname⟩*, *⟨Forenames⟩*

Again the *⟨von part⟩* is optional, and is determined by the case of the first letter. For example:

```
author = "de Vere, Henry James"
```

Multiple authors or editors should be separated by the key word and, for example:

```
author = "Michel Goossens and Frank Mittlebach and Alexander Sam"
```

Below is an example of a book entry:

```
@book{latexcomp,
  title      = "The \LaTeX\ Companion",
  author     = "Michel Goossens and Frank Mittlebach and
               Alexander Samarin",
  publisher  = "Addison-Wesley",
  year       = 1994
}
```

Note that numbers may be entered without delimiters, as in `year = 1994`. There are also some predefined strings, including those for the month names. It's best to use these strings instead of the actual month name, as the way the month name is displayed depends on the bibliography style. For example:

```
@article{Cawley2007b,
  author = "Gavin C. Cawley and Nicola L. C. Talbot",
  title  = "Preventing over-
            fitting in model selection via {B}ayesian
            regularisation of the hyper-parameters",
  journal = "Journal of Machine Learning Research",
  volume  = 8,
  pages   = "841--861",
  month   = APR,
  year    = 2007
}
```

You can concatenate strings using the `#` character, for example:

```
month = JUL # "~31~---~" # AUG # "~4",
```

Depending on the bibliography style, this may be displayed as: July 31 – August 4, or it may be displayed as: Jul 31 – Aug 4.

7.2.2. Column Keys (Fields)

Each row of a *databib* database corresponds to an entry in the *bib* file. The custom *data-bib.bst* file ensures that $\text{\LaTeX}$ creates a file where each selected citation is added as a new row to the database, and each column in that row corresponds to a *bib* field supported by *databib.bst*. Note that although $\text{\LaTeX}$ fields are case-insensitive, the *datatool* column keys are case-sensitive. These case-sensitive column keys are described below.

CiteKey

This column will always be set and corresponds to the entry `\cite-key`, that is the label used in `\cite` that uniquely identifies the entry. The value in this field will match the `\cite-key` given in the bib file.

EntryType

This column will always be set to the BibTeX entry type in lowercase, without the leading `@`. For example, an entry defined with `@ARTICLE` in the bib file will have the `EntryType` column set to `article`.

The remaining columns correspond to the field provided by `databib.bst` and will be null if not set or not supported by the entry type. The columns will only be added to the database if the field was found by BibTeX (that is, the field is present in the bib file and supported by the entry type).

Abstract

Corresponds to the `abstract` bib field.

Address

Corresponds to the `address` bib field.

Author

Corresponds to the `author` bib field.

BookTitle

Corresponds to the `booktitle` bib field.

Chapter

Corresponds to the `chapter` bib field.

Citations

Corresponds to the `citations` bib field. This is intended for use in CV style documents to show the number of times the reference has been cited by others. It's not a count of the number of times the reference has been cited in the current document.

Date

Corresponds to the `date bib` field. This may be used instead of the `year` and `month` fields.

DOI

Corresponds to the `doi bib` field. Note that the DOI can contain awkward characters. Whilst the `\doi` command provided by the `doi` package is designed to support such identifiers, that command can't be used in the argument of another command. Therefore this field will be set with `\DTLnewbiblitem` instead of `\DTLnewbibitem`. Note that any braces within the value must be balanced.

Edition

Corresponds to the `edition bib` field.

Editor

Corresponds to the `editor bib` field.

EID

Corresponds to the `eid bib` field.

Eprints

Corresponds to the `eprints` or `eprint bib` field. Since this may contain problematic characters, this field will be set with `\DTLnewbiblitem` instead of `\DTLnewbibitem`.

EprintType

Corresponds to the `eprinttype bib` field. (Only set if `eprints` or `eprint` is also set.)

File

Corresponds to the `file bib` field. Since this may contain problematic characters, this field will be set with `\DTLnewbiblitem` instead of `\DTLnewbibitem`.

HowPublished

Corresponds to the `howpublished bib` field.

7. Converting a BibTeX database into a datatool database (databib package)

`Institution`

Corresponds to the `institution bib` field.

`ISBN`

Corresponds to the `isbn bib` field.

`ISSN`

Corresponds to the `issn bib` field.

`Journal`

Corresponds to the `journal bib` field.

`Key`

Corresponds to the `key bib` field.

`Month`

The month is obtained from the `month bib` field, but predefined strings are available that will use `\DTLmonthname` to make sorting easier.

`Note`

Corresponds to the `note bib` field.

`Number`

Corresponds to the `number bib` field.

`Organization`

Corresponds to the `organization bib` field.

`Pages`

Corresponds to the `pages bib` field.

7. Converting a BibTeX database into a datatool database (databib package)

Publisher

Corresponds to the `publisher bib` field.

PubMed

Corresponds to the `pubmed bib` field.

School

Corresponds to the `school bib` field.

Series

Corresponds to the `series bib` field.

Title

Corresponds to the `title bib` field.

Type

Corresponds to the `type bib` field.

Url

Corresponds to the `url bib` field. Since this may contain problematic characters, this field will be set with `\DTLnewbibliteralitem` instead of `\DTLnewbibitem`.

UrlDate

Corresponds to the `urldate bib` field. (Only set if `url` is also present.)

Volume

Corresponds to the `volume bib` field.

Year

Corresponds to the `year bib` field.

7.3. Loading a databib database

The databib package always requires the `databib.bst` bibliography style file (which is supplied with datatool). This ensures that the `bbl` file will be in the format required for `\DTLloadbbl`. That is, the `bbl` file will contain the commands that construct the database with the bibliographic data (see §7.4).

You need to use `\cite` or `\nocite` as usual. If you want to add all entries in the `bib` file to the datatool database, you can use `\nocite{*}`.

```
\DTLloadbbl[⟨bbl file⟩]{⟨db-name⟩}{⟨bib-list⟩}
```

The `⟨db-name⟩` argument may be empty to indicate the `default-name`. This command performs several functions:

1. writes the following line in the auxiliary file:

```
\bibstyle{databib}
```

which tells BibTeX to use the `databib.bst` BibTeX style file;

2. writes `\bibdata{⟨bib list⟩}` to the auxiliary file, which tells BibTeX which `bib` files to use;
3. creates a datatool database called `⟨db name⟩`;
4. loads the file `⟨bbl file⟩` if it exists. (If the optional `⟨bbl file⟩` argument is omitted, the value defaults to `\jobname.bbl`, which is the usual name for a `bbl` file.) If the `bbl` file doesn't exist, the database `⟨db name⟩` will remain empty.

You then need to compile your document with L^AT_EX and then run BibTeX on the auxiliary file, as described in §7.2. This will create a `bbl` file which contains all the commands required to add the bibliography information to the datatool database called `⟨db name⟩`. The next time you L^AT_EX your document, this file will be read, and the information will be added to the database `⟨db name⟩`.

Note that `\DTLloadbbl` doesn't generate any text. Once you have loaded the data, you can display the bibliography using `\DTLbibliography` (described in §7.6) or you can iterate through it with `\DTLforeachbibentry` described in §7.8.

Note that the `databib.bst` BibTeX style file provides additional fields (see §7.2.2) that are not supported by the basic BibTeX styles, and so are ignored by the three predefined databib styles (`plain`, `abbrv` and `alpha`). If you want these fields to be displayed in the bibliography you will need to modify the bibliography style (see §7.7.1). The simplest method is to add the extra information in the `\DTLendbibitem` hook.

If you use the standard BibTeX month abbreviations (JAN, FEB, etc), the `databib.bst` style will convert those to:

```
\DTLmonthname{<month num>}
```

The definition varies depending on the `style` and also on whether or not any localisation support has been provided.

7.4. The databib Database Construction Commands

The `bbl` file created by the `databib.bst` BibTeX style file contains database construction commands rather than code to typeset the bibliography. These commands don't have the database name included in an argument (since the `databib.bst` style file doesn't have that information). Instead, they rely on the following placeholder command:

```
\DTLBIBdbname
```

This should expand to the `databib` database name.

This file is input by `\DTLloadbbl` and `\DTLloadmdbl`, but note that there's no database creation command within the `bbl` file. The database creation is performed by `\DTLnewdb` within the definition of `\DTLloadbbl` and `\DTLloadmdbl`.

This means that it's possible to append to an existing database by simply inputting the `bbl` file (with a check for existence). However, you must make sure that the `databib` bibliography style is set and that `\DTLBIBdbname` is defined to expand to the database name.

```
\DTLnewbibrow
```

Appends a new row to the database. This essentially just does:

```
\DTLnewrow*{\DTLBIBdbname}
```

Note that this is the starred form that doesn't test for the database existence.

```
\DTLnewbibitem{<col-key>}{<item>}
```

Adds a new entry to the final row of the database. This essentially just does:

```
\DTLnewdbentry*{\DTLnewbibitem}{<col-key>}{<item>}
```

Again, the starred form is used which doesn't test for the database existence.



```
\DTLnewbibliteralitem{\<col-key>}{\<item>}
```

This command is used for fields with literal values, such as `Url` and `DOI`. The `\<item>` will be detokenized before being added to the final row of the database. Note that any braces within `\<item>` must be balanced.



Remember that the category code of tokens within database items will be lost when a database is written to a file with `\DTLwrite`. This means that these literal fields may cause a problem if you choose to save the database. The best solution in this case is to save to a CSV file which can then be loaded with `csv-content=literal`.

7.5. Sorting a databib database

Once a database has been loaded with `\DTLloadbbl` (or `\DTLloadmbbl`), it can be sorted using `\DTLsortdata` or `\DTLsort` (see §3.14). The `databib` package automatically does:

```
\dtlSortWordCommands{\let\DTLmonthname \@firstofone}
```

This means that `\DTLmonthname` will expand to its numeric argument when the sort values are created, which makes it easier to sort chronologically. Just make sure to use the BibTeX month strings (JAN, FEB, etc) as in the examples in §7.10.

For example, to sort in reverse chronological order:



```
\DTLsortdata[missing-column-action=ignore]
{mybib}% database name
{Date=descending, Year=descending, Month=descending}
```

This allows for dates to be specified in either the `Date` field or in the `Year` and `Month` fields.

If you want to sort by `Author` or `Editor`, remember that those fields will contain comma-separated lists where each element is in the form `{\<von>}{\<surname>}{\<jr>}{\<forenames>}`. For example, if a bib entry has:

```
author = {von Parrot, Jr, Ann and José Arara}
```

then the `Author` field for the corresponding row in the `databib` database will be:

```
{von}{Parrot}{Jr}{Ann}, {}{Arara}{}{José}
```

This means that if the database is sorted by `Author`:

```
\DTLsortdata{mybib}{Author}
```

then the sort value will be:

```
vonParrotJrAnn, AraraJosé
```

This may or may not be sufficient, depending on your requirements.

To make it easier to adjust the sort value for the `Author` and `Editor` fields, `databib` provides:

```
\DTLbibsortencap{<value>}{<col-idx>}{<db-name>}
```

This may be used as the sort `encap` function. This command tests if `<col-idx>` corresponds to the column index for the `Author` or `Editor` keys. If it doesn't, it will simply expand to `<value>`. If it does, it will expand the comma-separated value so that each element is separated with:

```
\DTLbibsortnamesep initial: \datatoolasciend
```

and each element is encapsulated with:

```
\DTLbibsortname{<von>}{<surname>}{<jr>}{<forename>}
```

which, by default, does:

```
\tl_if_empty:nF {<von>} { <von> ~ }
<surname>
\tl_if_empty:nF {<jr>} { , ~ <jr> }
\datatoolpersoncomma
<forename>
```

This means that the earlier example will expand to:

```
von Parrot, Jr\datatoolpersoncomma Ann\DTLbibsort-
namesep
Arara\datatoolpersoncomma José
```

This produces a better result where there are multiple authors. You can redefine `\DTLbibsortname` if a different sort value is required. For example, to ignore the `<von>` and `<jr>` parts:

```
\renewcommand{\DTLbibsorthname}[4]{%
  #2\datatoolpersoncomma #4}
```

Remember that the definition must be expandable.

For example, the following sorts by `Author`, but if the `Author` field hasn't been set it will sort by `Editor`, and if the `Editor` hasn't been set, it will sort by `Title`. In the event that there are duplicate primary values, the secondary sort will be by the `Title`:

```
\DTLsortdata[encap=\DTLbibsorthencap]
{mybib}{Author={replacements={Editor,Title}},Title}
```

7.6. Displaying a databib database

A databib database which has been loaded using `\DTLloaddbbl` (described in §7.3) can be displayed using:

```
\DTLbibliography[⟨condition⟩]{⟨db-name⟩}
```

where `⟨db name⟩` is the name of the database. This command internally iterates over the database, formatting each row that matches `⟨condition⟩` according to the current style. It's provided as a convenient shortcut that does:

```
\begin{DTLthebibliography}[⟨condition⟩]{⟨db-name⟩}
\DTLforeachbibentry*[⟨condition⟩]{⟨db-name⟩}{%
  \DTLbibitem \DTLformatbibentry \DTLendbibitem
}%
\end{DTLthebibliography}
```

Within the optional argument `⟨condition⟩`, you may use any of the commands that may be used within the optional argument of `\DTLforeach` (that is, any condition that may be used in `\ifthenelse`, see §2.4.2). *In addition*, you may use the following commands, which reference values in the current row.

```
\DTLbibfieldexists{⟨field label⟩}
```

This tests whether the field with the given label exists and is not null for the current entry. The field label is the column key of the database.

7. Converting a BibTeX database into a datatool database (databib package)

For example, suppose you have loaded a databib database called “mybib” using `\DTLloadbb1` (described in §7.3) then the following bibliography will only include those entries which have a `Year` field:

```
\DTLbibliography[\DTLbibfieldexists{Year}]{mybib}
```

```
\DTLbibfieldiseq{<field label>}{<value>}
```

This tests whether the value of the field given by `<field label>` equals `<value>`. If the field doesn’t exist for the current entry, this evaluates to false. For example, the following will produce a bibliography which only contains entries which have the `Year` field set to 2004:

```
\DTLbibliography[\DTLbibfieldiseq{Year}{2004}]{mybib}
```

```
\DTLbibfieldcontains{<field label>}{<value>}
```

This tests whether the value of the field given by `<field label>` contains `<value>`. For example, the following will produce a bibliography which only contains entries where the author field contains the name “Knuth”:

```
\DTLbibliography[\DTLbibfieldcontains{Author}{Knuth}]{mybib}
```

```
\DTLbibfieldislt{<field label>}{<value>}
```

This tests whether the value of the field given by `<field label>` is lexicographically less than `<value>`. If the field doesn’t exist for the current entry, this evaluates to false. For example, the following will produce a bibliography which only contains entries whose `Year` field is less than 1983:

```
\DTLbibliography[\DTLbibfieldislt{Year}{1983}]{mybib}
```

```
\DTLbibfieldisle{<field label>}{<value>}
```

This tests whether the value of the field given by $\langle field\ label \rangle$ is lexicographically less than or equal to $\langle value \rangle$. If the field doesn't exist for the current entry, this evaluates to false. For example, the following will produce a bibliography which only contains entries whose `Year` field is less than or equal to 1983:

```
\DTLbibliography[\DTLbibfieldisle{Year}{1983}]{mybib}
```

```
\DTLbibfieldisgt{\langle field label \rangle}{\langle value \rangle}
```

This tests whether the value of the field given by $\langle field\ label \rangle$ is lexicographically greater than $\langle value \rangle$. If the field doesn't exist for the current entry, this evaluates to false. For example, the following will produce a bibliography which only contains entries whose `Year` field is greater than 1983:

```
\DTLbibliography[\DTLbibfieldisgt{Year}{1983}]{mybib}
```

```
\DTLbibfieldisge{\langle field label \rangle}{\langle value \rangle}
```

This tests whether the value of the field given by $\langle field\ label \rangle$ is lexicographically greater than or equal to $\langle value \rangle$. If the field doesn't exist for the current entry, this evaluates to false. For example, the following will produce a bibliography which only contains entries whose `Year` field is greater than or equal to 1983:

```
\DTLbibliography[\DTLbibfieldisge{Year}{1983}]{mybib}
```

Note that `\DTLbibliography` uses `\DTLforeachbibentry` (described in §7.8) so you may also test the value of the counter `DTLbibrow` within $\langle conditions \rangle$ (as in Example 186), but bear in mind that the counter is only incremented after testing the condition (and then only if the condition evaluates to true). You may also use the boolean commands defined by the `ifthen` package, such as `\not`.

7.7. Changing the bibliography style

The style of the bibliography produced using `\DTLbibliography` may be set with the `style` package option, or with:

```
\DTLbibliographystyle{\<name>}
```

Note that this is *not* the same as `\bibliographystyle`, as the *databib* package uses its custom `databib.bst` bibliography style file.

For example:

```
\usepackage[style=plain]{databib}
```

or

```
\DTLbibliographystyle{plain}
```

Both of the above set the `plain` bibliography style. This is, in fact, the default style, so it need not be specified.

Available styles are: `plain`, `abbrv` and `alpha`. These are similar to the standard $\text{\LaTeX}$ styles of the same name, but are by no means identical. The most notable difference is that these styles do not sort the bibliography. It is up to you to sort the bibliography using `\DTLsort-data` or `\DTLsort` (described in §3.14).

7.7.1. Modifying an existing style

You can use `\DTLforeachbibentry` and format each row according to your particular requirements. However, the predefined styles used by `\DTLbibliography`, `\DTLmbibliography`, and `\DTLformatthisbibentry` provide a more convenient method if they are sufficient for your needs. The hooks described in this section allow you to make minor adjustments the selected style.

Some of the commands described here do more than simply typeset the required text. They also perform some internal checks to keep track of the end of a sentence to determine whether or not a period needs to be inserted or if the `spacefactor` needs adjusting.

```
\DTLformatauthor{\<von part>}{\<surname>}{\<jr part>}{\<forenames>}
```

Formats an author's name.

```
\DTLformatteditor{\<von part>}{\<surname>}{\<jr part>}{\<forenames>}
```

Formats an editor's name.

```
\DTLformatsurnameonly{<von part>}{<surname>}{<jr
part>}{<forenames>}
```

Formats the name, omitting the forenames.

The list of authors and editors are stored in the databib database as a comma separated list where each item is in the form {<von part>}{<surname>}{<jr part>}{<forenames>}. This assists with sorting by author or editor (but this may require adjustment, see §7.5). Each element of the list can then be passed to \DTLformatauthor or \DTLformatteditor or \DTLformatsurnameonly for formatting.

Within the definitions of \DTLformatauthor and \DTLformatteditor, you may use the following commands.

```
\DTLformatforenames{<forenames>}
```

This is used by the plain and alpha styles to display the author's forenames.

```
\DTLformatabbrvforenames{<forenames>}
```

This is used by the abbrv style to just show the author's initials, which are obtained from <forenames> by \DTLstoreinitials (see §2.8.2).

```
\DTLformatsurname{<surname>}
```

This is used to format the surname.

```
\DTLformatvon{<von>}
```

If the <von part> is empty, this command does nothing, otherwise it displays its argument followed by:

```
\DTLpostvon
```

initial: ~

This expands to a non-breaking space by default.

```
\DTLformatjr{<jr>}
```

If the <jr part> is empty, this command displays nothing, otherwise it does a comma and space followed by its argument.

For example, suppose you want the author's surname to appear first in small capitals, followed by a comma and the forenames. This can be achieved with:

```
\renewcommand*{\DTLformatauthor}[4]{%
  \textsc{\DTLformatvon{#1}\DTLformatsurname{#2}\DTL-
  formatjr{#3}},
  \DTLformatforenames{#4}%
}
```

DTLmaxauthors

The counter DTLmaxauthors is used to determine the maximum number of authors to display for a given entry. If the author list contains more than that number of authors, \etalname is used.

DTLmaxeditors

The DTLmaxeditors counter is analogous to the DTLmaxauthors counter. It is used to determine the maximum number of editor names to display.

`\DTLformatauthorlist`

Used by styles to format the list of author names (which will be obtained from the [Author](#) field). Each name is formatted with to `\DTLformatauthor`.

`\DTLformatteditorlist`

Used by styles to format the list of editor names (which will be obtained from the [Editor](#) field). Each name is formatted according to `\DTLformatteditor`. The list is followed by a comma and space and either `\editorname` or `\editorsname`, depending on whether the list contains one or more elements.

The above two list commands internally use:

`\DTLformatbibnamelist{<list>}{<max>}{<fmt-cs>}`

This formats a list of names, where each name in the list must be in the form `{<von>}{<surname>}{<jr>}{<forenames>}` and `<fmt-cs>` must have those four arguments for its syntax. If the list has more than `<max>` items, it will be truncated with `\etalname`.

Within a list of names, the separators used are:

`\DTLlandlast`

Placed between two names when the list contains more than two names.

`\DTLandnotlast`

Placed between all except the last two names when the list contains more than two names.

`\DTLtwoand`

Placed between the two names when the list only contains two names. The definitions of `\DTLandlast` and `\DTLtwoand` use `\DTLlistand` for “and”.

`\DTLformatdate`

This is used by the styles to insert the date. It first tests if the `Date` field is set. If so, it will format that field with `\DTLbibdatefield`. Otherwise, it tests if the `Year` and `Month` fields are set.

`\DTLformatpages`

This is used by the styles to insert the page or page range. If the `Pages` field is set, this will be displayed preceded by the language-sensitive `\pagename` or `\pagesname`. The textual tag and the contents of the `Pages` field are separated with:

`\DTLpostpagename`

initial: ~

which expands to a non-breakable space by default.

`\DTLbibitem`

Used at the start of each bibliography item within `\DTLbibliography`. It uses `\bibitem` to provide a marker, such as [1], and writes the citation information to the aux file in order to resolve references on the next L^AT_EX run.

`\DTLmbibitem`

Used at the start of each bibliography item within `\DTLmbibliography`, and is analogous to `\DTLbibitem`.

`\DTLendbibitem`

A hook used by `\DTLbibliography` and `\DTLmbibliography` that does nothing by default, but may be redefined to include additional content at the end of the current entry.

Within the definition command, you may use the commands `\DTLbibfield`, `\DTLifbibfieldexists` and `\DTLifanybibfieldexists`, which are described in §7.8. For example, if you have used the `abstract` field in any of your entries, you can display the abstract as follows:

```
\renewcommand{\DTLendbibitem}{%
  \DTLifbibfieldexists{Abstract}{\DTLpar\textbf
{Abstract}
  \begin{quote}\DTLbibfield{Abstract}\end{quote}}{}%
}
```

```
\DTLbibformatdigital
```

This command is not used by any of the styles but may be added to the definition of `\DTLendbibitem` to include the digital information. The result varies depending on whether or not the `hyperref` or `url` packages have been loaded. The content will be included in the following order:

- If the `PubMed` field is set, this will be displayed with `\DTLbibpubmed`.
- If the `DOI` field is set, this will be displayed with `\DTLbibdoi`.
- If the `Url` field is set, this will be displayed with `\DTLbiburl`, and if the `UrlDate` field is also set, this will be displayed with `\DTLbiburldate`.
- If the `Eprints` field is set, this will be displayed with `\DTLbibeprints`, where the second argument will either be the content of the `EprintType` field (if set) or “eprint” otherwise.

The commands used by `\DTLbibformatdigital`, described below, may be redefined to customize the output.

```
\DTLbibpubmed{<pmid>}
```

This is used to format the `PubMed` field. The field content is passed as the argument. If the `hyperref` package has been loaded, the `<pmid>` will be a hyperlink.

```
\DTLbibpubmedtag
```

The tag used at the start of `\DTLbibpubmed`. By default, this simply expands to `\textsc{pmid}`:_U.

`\DTLbibpubmedhome`

initial: `https://pubmed.ncbi.nlm.nih.gov/`

Used by `\DTLbibpubmed` in the formation of the hyperlink, if applicable.

`\DTLbibdoi{<doi>}`

This is used to format the DOI field. The field content is passed as the argument. Remember that this field is set with `\DTLnewbibliteralitem` which detokenizes the content before adding it to the database. This means that `\DTLbibdoi` doesn't make any category code changes as it expects the argument to be detokenized already.

If the `hyperref` package has been loaded, the DOI will be a hyperlink otherwise it will simply be displayed in a monospaced font with `\url` (if defined) or `\texttt` otherwise. If `hyperref` isn't required but the DOI needs better formatting, try loading the `url` package.

`\DTLbibdoitag`

The tag used at the start of `\DTLbibdoi`. By default, this simply expands to `\textsc{doi}:`.

`\DTLbibdoihome`

initial: `https://doi.org/`

Used by `\DTLbibdoi` in the formation of the hyperlink, if applicable.

`\DTLbibeprints{<url>}{<eprint-type>}`

This command is used to format the Eprints field. The field content is passed as the first argument. The second argument is set by `\DTLbibformatdigital` to either the value of the EprintType field, if set, or "eprint" otherwise.

The content of the Eprints field is assumed to be a URL. If not, you will need to redefine `\DTLbibeprints` as applicable.

If `hyperref` has been loaded, a hyperlink will be created with the second argument as the link text, otherwise the first argument will be displayed in a monospaced font, prefixed by the second argument.

`\DTLbiburl{<url>}`

7. Converting a BibTeX database into a datatool database (databib package)

This command is used to format the `Url` field. The field content is passed as the argument. If `\url` has been defined (for example, by loading `hyperref` or the `url` package) then that will be used to format the argument otherwise it will simply be typeset in a monospaced font.

```
\DTLbiburldate{<date>}
```

This command is used to format the `UrlDate` field. The argument provided by `\DTLbib-formatdigital` will be:

```
\DTLbibdatefield{UrlDate}
```

The definition of `\DTLbiburldate` is:

```
\space (\DTLbibaccessedname: <date>)
```

which places the date in parentheses prefixed with:

```
\DTLbibaccessedname initial: accessed
```

7.8. Iterating through a databib database

The hooks described in §7.7.1 to adjust the format of `\DTLbibliography` may still not meet your needs. For example, you may be required to list journal papers and conference proceedings in separate sections. In which case, you may find it easier to iterate through the bibliography using:

```
\DTLforeachbibentry[<condition>]{<db-name>}{<body>} modifier: *
```

This iterates through the database identified by `<db-name>` and does `<body>` if `<condition>` is met. The starred version uses the read-only `\DTLforeach*` and the unstarred version uses the unstarred `\DTLforeach`. This means that you can use `\dtlbreak` in `<body>` to terminate the loop after the current iteration. Note that you can't have explicit paragraph breaks in `<body>` (that is, `\par` or a blank line in your source code). If you need a paragraph break, use `\DTLpar`.

Although `\DTLforeach` makes global changes, `\DTLforeachbibentry` only makes local assignments for the placeholder commands, which means that it's unsuitable to display the references in a tabular-like environment.

```
\gDTLforeachbibentry[<condition>]{<db-name>}{<body>} modifier: *
```

7. Converting a BibTeX database into a datatool database (databib package)

This is as `\DTLforeachbibentry` but the placeholder commands are globally assigned.

For each row of the database, the following commands are set:

- `\DBIBCitekey` This is the unique label which identifies the current entry (as used in the argument of `\cite` and `\nocite`).
- `\DBIBentrytype` This is the current entry type (converted to lowercase), and will be one of: `article`, `book`, `booklet`, `inbook`, `incollection`, `inproceedings`, `manual`, `mastersthesis`, `misc`, `phdthesis`, `proceedings`, `techreport` or `unpublished`. (Note that even if you used the entry type `conference` in your bib file, its entry type will be set to `inproceedings`).

The remaining fields may be accessed using:

```
\DTLbibfield{<field name>}
```

where *<field label>* may be any of those listed in §7.2.2.

If you want to encapsulate a field with a command, you can simply include `\DTLbibfield{<field name>}` in the command's argument. For example:

```
\strong{\DTLbibfield{Year}}
```

However, it's also possible to use:

```
\DTLencapbibfield{<cs>}{<field name>}
```

This is similar to `\DTLbibfield{<field name>}` but encapsulates the field value with *<cs>*. For example:

```
\DTLencapbibfield{\strong}{Year}
```

The difference is that in this case the field value will be directly passed to the formatting command. Also if the field isn't set, the command won't be used in the second case but will be used in the first. For example, `\DTMdate` (provided by the `datetime2` package) requires its argument to be in the form *<YYYY>-<MM>-<DD>* (there are some other formats as well, see the `datetime2` manual for further details). This means that it's not possible to do, say:

```
\DTMdate{\DTLbibfield{Date}}
```

Instead use:

```
\DTLencapbibfield{\DTMdate}{Date}
```

Both `\DTLbibfield` and `\DTLencapbibfield` expand to nothing if the field isn't set or doesn't exist.

```
\DTLbibdatefield{<field name>}
```

This command is used `\DTLformatdate` and `\DTLbibformatdigital` to format the `Date` and `UrlDate` fields, respectively. The default definition simply uses `\DTLbibfield` but may be redefined to format the date.

Alternatively, you can assign the value of a field to a control sequence `<cs>` using:

```
\DTLbibfieldlet{<cs>}{<field name>}
```

You can determine if a field exists for a given entry using

```
\DTLifbibfieldexists{<field name>}{<true>}{<false>}
```

If the field given by `<field label>` exists for the current bibliography entry, it does `<true part>`, otherwise it does `<false part>`.

```
\DTLifanybibfieldexists{<field list>}{<true>}{<false>}
```

This is similar to `\DTLifbibfieldexists` except that the first argument is a list of field names. If one or more of the fields given in `<field label list>` exists for the current bibliography item, this does `<true part>`, otherwise it does `<false part>`.

```
\DTLformatbibentry
```

This formats the bibliography entry for the current row. It checks for the existence of the command `\DTLformat<entry-type>`, where `<entry-type>` is given by `\DBIBentrytype`. These commands are defined by the bibliography style. There is also a version for use with `\gDTLforeachbibentry`:

```
\gDTLformatbibentry
```

It's also possible to use `\DTLformatbibentry` for a specific key, rather than using it within `\DTLforeachbibentry`.

```
\DTLformatthisbibentry{<db-name>}{<cite key>}
```

This formats the row identified by `<cite key>` in the database `<db-name>` using the same format as `\DTLformatbibentry`.

Note that none of the above three commands use `\bibitem`. You can manually insert `\bibitem{<cite key>}` in front of the command, or you can use:

```
\DTLcumbibitem{<item code>}{<ref text>}{<cite key>}
```

This is like `\bibitem[<text>]{<cite key>}` except that it uses `<item code>` instead of `\item[<text>]` and `<ref text>` instead of `\the\value{<@listctr>}`.

```
\DTLcomputewidestbibentry{<condition>}{<db-name>}{<bib-label>}{<cs>}
```

This computes the widest bibliography entry over all entries satisfying `<condition>` in the database `<db-name>`, where the label is given by `<bib-label>`, and the result is stored in `<cs>`, which may then be used in the argument of the `thebibliography` environment. The optional `<condition>` argument should be suitable for use in `\ifthenelse`, and you may use any of the commands that may be used within the optional argument of `\DTLbibliography`, described in §7.6

DTLbibrow

The counter `DTLbibrow` keeps track of the current bibliography entry. This is reset at the start of each `\DTLforeachbibentry` and is incremented if `<condition>` is met.

7.9. Multiple Bibliographies

It is possible to have more than one bibliography in a document, but it then becomes necessary to have a separate auxiliary (`aux`) file for each bibliography, and each auxiliary file must then be passed to BibTeX. In order to do this, you need to use

```
\DTLmultibibs{<list>}
```

preamble only

where `<list>` is a comma separated list of names. For each `<name>` in the list, this command creates an auxiliary file called `<name>.aux`.



In the following commands, the $\langle mbib \rangle$ argument must be one of the names listed in $\backslash\text{DTLmultibibs}$.

When you want to cite an entry for a given bibliography named in $\backslash\text{DTLmultibibs}$, you must use:



```
 $\backslash\text{DTLcite}[\langle text \rangle]\{\langle mbib \rangle\}\{\langle label-list \rangle\}$ 
```

This is analogous to $\backslash\text{cite}[\langle text \rangle]\{\langle label-list \rangle\}$, but writes the $\backslash\text{citation}$ command to $\langle mbib \rangle.aux$ instead of to the document's main auxiliary file. It also ensures that the cross-referencing labels are based on $\langle mbib \rangle$, to allow you to have the same reference in more than one bibliography without incurring a “multiply defined” warning message. Note that you can still use $\backslash\text{cite}$ to add citation information to the main auxiliary file.

If you want to add an entry to the bibliography without producing any text, you can use



```
 $\backslash\text{DTLnocite}\{\langle mbib \rangle\}\{\langle label-list \rangle\}$ 
```

which is analogous to $\backslash\text{nocite}\{\langle label-list \rangle\}$, where again the citation information is written to $\langle mbib \rangle.aux$ instead of the document's main auxiliary file.



```
 $\backslash\text{DTLloadmbbl}\{\langle mbib \rangle\}\{\langle db-name \rangle\}\{\langle bib-list \rangle\}$ 
```

This is similar to $\backslash\text{DTLloadbbl}$, described in §7.3. This creates a new datatool database called $\langle db-name \rangle$ and inputs $\langle mbib \rangle.bbl$ (if it exists), which should contain the commands required to construct the database.



```
 $\backslash\text{DTLmbibliography}[\langle condition \rangle]\{\langle mbib name \rangle\}\{\langle db-name \rangle\}$ 
```

This is similar to $\backslash\text{DTLbibliography}$, but is required when displaying a bibliography in which elements have been cited using $\backslash\text{DTLcite}$ and $\backslash\text{DTLnocite}$. As $\backslash\text{DTLbibliography}$, it iterates over the database using $\backslash\text{DTLforeachbibentry}^*$.

7.10. Examples

The examples here use one or both of the following sample files. The first file, `sample1.bib`, contains some of my own publications with a mixture of entry types:

```
@STRING{TUGBOAT = {TUGboat}}
@STRING{PRACTEX = {The Prac\TeX\Journal}}
```

```
@ARTICLE{Flom2005,
  author = {Peter Flom and Hans Hagen and Joe Hogg
and Nicola Talbot and Philip Taylor and Christina
Thiele and David Walden},
  title = {What is {\TeX}?},
  journal = PRACTEX,
  year = 2005,
  volume = 3,
  url =
{http://tug.org/pracjourn/2005-3/walden-what-is}
}
@ARTICLE{tugboat2016,
  title = "Localisation of {\TeX} documents:
tracklang",
  author = "Nicola Talbot",
  month = NOV,
  year = 2016,
  volume = 37,
  number = 3,
  journal = TUGBOAT,
  url =
"http://www.tug.org/TUGboat/tb37-3/tb117talbot.pdf"
}
@ARTICLE{tugboat2022,
  title = "bib2gls: Standalone entries and repeated
lists (a little book of poisons)",
  author = "Nicola Talbot",
  month = APR,
  year = 2022,
  volume = 43,
  number = 1,
  journal = TUGBOAT,
  doi = 10.47397/tb/43-1/tb133talbot-bib2gls-reorder,
  url = "https://tug.org/TUGboat/tb43-1/tb133talbot-
bib2gls-reorder.pdf"
}
@BOOK{Talbot2012a,
  title = {{\LaTeX} for Complete Novices},
  publisher = {Dickimaw Books},
  year = 2012,
  month = SEP,
  author = {Nicola L C Talbot},
```

```

volume = 1,
series = {Dickimaw {\LaTeX} Series},
isbn = 978-1-909440-00-5,
url =
{https://www.dickimaw-books.com/latex/novices/}
}
@INPROCEEDINGS{Talbot2020,
author = {Nicola L C Talbot},
title = {Sorting Glossaries with bib2gls},
booktitle = {{\LaTeX}.net},
year = 2020,
month = JUL,
url =
{https://latex.net/sorting-glossaries-with-bib2gls/}
}
@BOOK{Talbot2013a,
title = {Using {\LaTeX} to Write a {PhD} Thesis},
publisher = {Dickimaw Books},
year = 2013,
month = MAR,
author = {Nicola L C Talbot},
volume = 2,
series = {Dickimaw {\LaTeX} Series},
isbn = {978-1-909440-02-9},
url = {http://www.dickimaw-books.com/latex/thesis/}
}
@INPROCEEDINGS{Talbot2012c,
author = {Nicola L C Talbot},
title = {Creating a glossary without using an
external indexing application},
booktitle = {{\LaTeX}.net},
year = 2012,
month = SEP,
url =
{https://latex.net/glossary-without-external-app/},
note={Originally posted on the {\LaTeX} Community's
Know How Section}
}
@BOOK{Talbot2014a,
title = {{\LaTeX} for Administrative Work},
publisher = {Dickimaw Books},
month = SEP,

```

7. Converting a BibTeX database into a datatool database (databib package)

```
year = 2014,
author = {Nicola L C Talbot},
volume = 3,
series = {Dickimaw {\LaTeX} Series},
isbn = {978-1-909440-07-4},
url = {https://www.dickimaw-books.com/latex/admin/}
}
@BOOK{Talbot2013b,
title = {Quack, Quack, Quack. Give My Hat Back!},
publisher = {Dickimaw Books},
year = 2013,
month = MAY,
author = {Nicola L C Talbot and Magdalene
Pritchett},
isbn = {978-1-909440-03-6},
url =
{https://www.dickimaw-books.com/fiction/kids/duck/}
}
@BOOK{Talbot2012b,
title = {The Foolish Hedgehog},
publisher = {Dickimaw Books},
year = 2012,
month = NOV,
author = {Nicola L C Talbot and Magdalene
Pritchett},
isbn = {978-1-909440-01-2},
url = {https://www.dickimaw-
books.com/fiction/kids/hedgehog/}
}
```

The second file, `sample2.bib`, contains fictitious references:

```
@Article{duck2018,
Author = {Dickie Duck and José Arara and Polly
Parrot},
Title = {Avian Friendship},
Journal = {Fowl Times},
Year = 2018,
Volume = 7,
Number = 5,
Pages = "1032--5"
}
```



```
@BOOK{duck2016,
  AUTHOR = {Dickie Duck},
  TITLE = {Feathered stunt doubles: \emph{The Birds}
and other films},
  PUBLISHER = {Duck Duck Goose},
  YEAR = 2016
}
@book{macaw,
  author = {Prof Macaw},
  title = {Annotated notes on the \emph{Duck and
Goose} chronicles},
  publisher = Duck Duck Goose,
  year = 2012
}
@inproceedings{parrot2021a,
  author = {Polly Parrot},
  title = {Who's a Pretty Polly? {T}he surge of avian
chatbots},
  booktitle = {Avian Intelligence},
  month = jan,
  year = 2021
}
@inproceedings{parrot2021b,
  author = {Polly Parrot},
  title = {Pollybot: the next generation in avian
translators},
  booktitle = {Avian Advances},
  month = apr,
  year = 2021
}
@phdthesis{ing2020,
  author = {Bor Ing},
  title = {\emph{Duck and Goose}: an allegory for
modern times?},
  school = {Department of Literature, University of
Somewhere},
  month = mar,
  year = 2010
}
@booklet{parrots2013,
  author = {Polly Parrot and Dickie Duck},
  title = {\emph{Duck and Goose} Cheat Sheet},
```

```

    howpublished = {Limited print run},
    address = {Dubious Student Resources},
    year = 2013
}
@book{parrot2012,
  author = {von Parrot, Jr, Ann},
  title = {My Friend is a Duck},
  publisher = {Duck Duck Goose},
  year = 2012,
}
@book{quackalot,
  author = {Sir Quackalot},
  title = {The Adventures of Duck and Goose},
  publisher = {Duck Duck Goose},
  year = 2011
}
@techreport{zebra2022,
  author = {Zoë Zebra and Mabel Canary},
  title = {Health and Safety when Handling
Mind-Controlling Cookies},
  institution = {Secret Lab of Experimental Stuff},
  month = {22 } # MAR,
  year = 2014
}
@manual{canary2015,
  author = {Mabel Canary},
  title = {Ray Gun User Guide},
  organization = {Secret Lab of Experimental Stuff},
  edition = "2nd",
  year = 2015
}
@book{fan1992,
  author = {Éli-Fant, Nellie},
  title = {The Duckinator},
  publisher = {Duck Duck Goose},
  year = 1992,
  note = {A cyborg from the future travels to the
past to scramble some eggs.}
}
@misc{henpecked,
  title = {Henpecked: Time for a Coup in the Coop!},
  howpublished = {Flyer}

```

```
}
```

Note that the `bib` entry types and field names are case-insensitive. I've used a mixture of cases above.

7.10.1. Sort by Author

Example 183 demonstrates the basic use of `databib`. The `abbrv` style is set with the `style` package option. I've also set the locale to English. This requires `datatool-english`, which should be installed separately.

183

```
\usepackage[style=abbrv,locales=en]{databib}
```

Next is the instruction to BibTeX to select all the citations in the `bib` file:

```
\nocite{*}
```

The following line will add the appropriate code in the `aux` file to instruct BibTeX to fetch the data from `sample2.bib` and create a `bbl` file (according to the format specified by `databib.bst`), which will then be input, if it exists:

```
\DTLloadbbl{mybib}{sample2}
```

This will do the following:

1. Writes the line:

```
\bibstyle{databib}
```

to the auxiliary file. This tells BibTeX to use `databib.bst` (which is supplied with this package). You therefore shouldn't use `\bibliographystyle`.

2. Writes the line:

```
\bibdata{sample2}
```

to the auxiliary file. This tells BibTeX that the bibliography data is stored in the file `sample2.bib`.

3. Creates a datatool database called `mybib` (using `\DTLnewdb`).

4. If the `bb1` file exists, `\DTLloadbb1` will input the file (which contains code to add the bibliography data to the database), otherwise it does nothing further.

The database is then sorted by author (or title, if there's no author) and then by title. This can be done with `\DTLsortdata` (see §3.14.1):

```
\DTLsortdata{mybib}{Author={replacements=Title},Title}
```

However, bear in mind that the `Author` field will contain a comma-separated list of names, with each item in the form `{\von}{\surname}{\jr}{\forenames}`. This means that the names will all be merged together in the sort value. The `encap` option can be used to separate the names:

```
\DTLsortdata[encap=\DTLbibsorthname]{mybib}{Author={replacements=Title},Title}
```

This will encapsulate each name in the author list with `\DTLbibsorthname`. With the default definition, “von Duck” will have a sort value starting with “v”. If the “von” should be ignored when sorting, just redefine `\DTLbibsorthname` to omit it (or place it after the surname).

The bibliography is then displayed with:

```
\DTLbibliography{mybib}
```

Suppose I save this file as `myDoc.tex`, then I need to do:

```
latex myDoc
bibtex myDoc
latex myDoc
```

The result is shown in Example 183.

7.10.2. Tabulate Bib Data

Note that it's not necessary to actually display the bibliography if it's not required (but it would be required if any entries are referenced with `\cite`). If the required entries are simply selected with `\nocite` then this simply provides a convenient way to convert the data from the `bib` file into a format compatible with `datatool`.

Example 184 makes a minor modification to Example 183 so that, instead of using `\DTLbibliography{mybib}`, it displays the `Author` and `Title` values in a table. This can be done with `\DTLdisplaydb` but the `Author` field will be a comma-separated list with each

184

↑ Example 183: Bibliography Sorted by Author



References

- [1] M. Canary. *Ray Gun User Guide*. Secret Lab of Experimental Stuff, 2nd edition, 2015.
- [2] D. Duck. *Feathered stunt doubles: The Birds and other films*. Duck Duck Goose, 2016.
- [3] D. Duck, J. Arara, and P. Parrot. Avian friendship. *Fowl Times*, 7(5):1032–5, 2018.
- [4] N. Éli Fant. *The Duckinator*. Duck Duck Goose, 1992. A cyborg from the future travels to the past to scramble some eggs.
- [5] Henpecked: Time for a coup in the coop! Flyer.
- [6] B. Ing. Duck and Goose: *an allegory for modern times?* PhD thesis, Department of Literature, University of Somewhere, Mar. 2010.
- [7] P. Macaw. *Annotated notes on the Duck and Goose chronicles*. Duck Duck Goose, 2012.
- [8] P. Parrot. Pollybot: the next generation in avian translators. In *Avian Advances*, Apr. 2021.
- [9] P. Parrot. Who’s a pretty polly? The surge of avian chatbots. In *Avian Intelligence*, Jan. 2021.
- [10] P. Parrot and D. Duck. *Duck and Goose* cheat sheet. Limited print run, Dubious Student Resources, 2013.
- [11] S. Quackalot. *The Adventures of Duck and Goose*. Duck Duck Goose, 2011.
- [12] A. von Parrot, Jr. *My Friend is a Duck*. Duck Duck Goose, 2012.
- [13] Z. Zebra and M. Canary. Health and safety when handling mind-controlling cookies. Technical report, Secret Lab of Experimental Stuff, 22 Mar. 2014.

element in the form $\{\langle von \rangle\} \{\langle surname \rangle\} \{\langle jr \rangle\} \{\langle forenames \rangle\}$, which will result in the name elements running together. This can be dealt with by adjusting `\DTLdisplaydbAddItem` so that it encapsulates the `Author` value with `\DTLformatbibnamelist` (if it's not null). This assumes the `Author` column is the first column in the table (which it will be if it's the first item in the `only-keys` list):

```
\RenewDocumentCommand\DTLdisplaydbAddItem
{ m m m m m m m m }
{ %
  \DTLifnull{#2}%
  {\appto#1{---}}% do a dash if null
  {%
    \ifnum#7=1
      \appto#1{\DTLformatbibnamelist{#2}\value{DTLmax-
authors}}{\DTLformatauthor}}%
    \else
      \appto#1{#3{#2}}}%
    \fi
  }%
}
```

Note that the above won't work if `store-datum` has been set, which it isn't by default. The data can now be displayed:

```
\DTLdisplaydb*[only-keys={Author,Title}]{mybib}
```

The document build is the same as before as BibTeX is still needed to fetch the data from the bib file. Since `\DTLformatauthor` is set by the style and the style has been set to `abbrv`, the forenames are abbreviated. The result is shown in Example 184.

7.10.3. Publications Since a Given Year

Suppose my boss has asked me to produce a list of my publications in reverse chronological order, but doesn't want any publications published prior to the year 2013. I have a file which contains all my publications which I keep in my personal TEXMFHOME tree $\langle \text{TEXMFHOME} \rangle$ /`bibtex/bib/`. I could look through this file, work out the labels for all the publications whose year field is greater or equal to 2013, and create a file with a `\nocite` command containing all those labels in a comma separated list in reverse chronological order, but I really can't be bothered to do that. Instead, I can use `databib` to convert my publication data into a `datatool` database:

185

↑ Example 184: Tabulate Bib Data

Author	Title
M. Canary	Ray Gun User Guide
D. Duck	Feathered stunt doubles: <i>The Birds</i> and other films
D. Duck, J. Arara, and P. Parrot	Avian friendship
N. Éli Fant	The Duckinator
—	Henpecked: Time for a coup in the coop!
B. Ing	<i>Duck and Goose</i> : an allegory for modern times?
P. Macaw	Annotated notes on the <i>Duck and Goose</i> chronicles
P. Parrot	Pollybot: the next generation in avian translators
P. Parrot	Who's a pretty polly? The surge of avian chatbots
P. Parrot and D. Duck	<i>Duck and Goose</i> cheat sheet
S. Quackalot	The Adventures of Duck and Goose
A. von Parrot, Jr	My Friend is a Duck
Z. Zebra and M. Canary	Health and safety when handling mind-controlling cookies

```
\documentclass{article}
\usepackage{databib}
```

Next is the instruction to BibTeX to select all the citations in the bib file:

```
\nocite{*}
```

Since my own bib file isn't available (and is too long to include), I'm going to use the `sample1.bib` file:

```
\DTLloadbbl{mybib}{sample1}
```

The database is then sorted in reverse chronological order:

```
\DTLsortdata{mybib}{Year=descending,Month=descending}
```

If the bibliography database is large, sorting and creating the bibliography may take a while. For large bib files or complex requirements, consider using `biblatex` and `biber` instead.

The data is then displayed with filtering applied, so that only the entries where the row is greater than or equal to 2013 are shown:

```
\DTLbibliography[\DTLbibfieldisge{Year}{2013}]{mybib}
```

This is a string comparison, rather than a numerical comparison, but since all the years are four digits it's sufficient. The result is shown in Example 185.

↑ Example 185: List of Publications Since a Given Year

References

- [1] Nicola Talbot. bib2gls: Standalone entries and repeated lists (a little book of poisons). *TUGboat*, 43(1), April 2022.
- [2] Nicola L C Talbot. Sorting glossaries with bib2gls. In *LaTeX.net*, July 2020.
- [3] Nicola Talbot. Localisation of T_EX documents: tracklang. *TUGboat*, 37(3), November 2016.
- [4] Nicola L C Talbot. *L^AT_EX for Administrative Work*, volume 3 of *Dickimaw L^AT_EX Series*. Dickimaw Books, September 2014.
- [5] Nicola L C Talbot & Magdalene Pritchett. *Quack, Quack, Quack. Give My Hat Back!* Dickimaw Books, May 2013.
- [6] Nicola L C Talbot. *Using L^AT_EX to Write a PhD Thesis*, volume 2 of *Dickimaw L^AT_EX Series*. Dickimaw Books, March 2013.

7.10.4. Five Most Recent Publications

Suppose my boss has asked me to produce a list of my five most recent publications (in reverse chronological order). I can create the required document with just a minor modification to Example 185:

```
\DTLbibliography[\value{DTLbibrow}<5]{mybib}
```

This is just a different condition in the optional argument. Note that the condition is evaluated before the DTLbibrow counter is incremented (as it's only incremented if the condition is true).

In addition, to demonstrate the end item hook, I've redefined `\DTLendbibitem` to show the ISBN and URL fields, if they have been set:

186


```
\renewcommand{\DTLendbibitem}{%
\DTLifbibfieldexists{ISBN}%
{ ISBN: \DTLbibfield{ISBN}.}{}}%
\DTLifbibfieldexists{Url}%
{ \DTLencapbibfield{\url}{Url}}{}}%
}
```

Note that this will require a package that defines `\url`. In this case, I've used the `url` package, but this may be replaced with `hyperref` if hyperlinks are required. The result is shown in Example 186.

↑ Example 186: Five Most Recent Publications

References

- [1] Nicola Talbot. `bib2gls`: Standalone entries and repeated lists (a little book of poisons). *TUGboat*, 43(1), April 2022. <https://tug.org/TUGboat/tb43-1/tb133talbot-bib2gls-reorder.pdf>
- [2] Nicola L C Talbot. Sorting glossaries with `bib2gls`. In *LaTeX.net*, July 2020. <https://latex.net/sorting-glossaries-with-bib2gls/>
- [3] Nicola Talbot. Localisation of T_EX documents: `tracklang`. *TUGboat*, 37(3), November 2016. <http://www.tug.org/TUGboat/tb37-3/tb117talbot.pdf>
- [4] Nicola L C Talbot. *L^AT_EX for Administrative Work*, volume 3 of *Dickimaw L^AT_EX Series*. Dickimaw Books, September 2014. ISBN: 978-1-909440-07-4. <https://www.dickimaw-books.com/latex/admin/>
- [5] Nicola L C Talbot & Magdalene Pritchett. *Quack, Quack, Quack. Give My Hat Back!* Dickimaw Books, May 2013. ISBN: 978-1-909440-03-6. <https://www.dickimaw-books.com/fiction/kids/duck/>

Note that, although Example 186 only shows five items, the loop iterates over all entries in the database. It's more efficient (particularly for large databases) to terminate the loop after the fifth row with `\dtlbreak`. This is done by replacing `\DTLbibliography` with:

```
\begin{DTLthebibliography}{mybib}
\DTLforeachbibentry{mybib}
{%
```

```

\DTLbibitem \DTLformatbibentry \DTLendbibitem
\ifthenelse{\value{DTLbibrow}=5}{\dtlbreak}{}
% terminate after row 5
}
\end{DTLthebibliography}

```

7.10.5. Compact Bibliography

Suppose I don't have much space in my document, and I need to produce a compact bibliography. 187
 Firstly, I can use the bibliography style `abbrv`, either through the package option:

```
\usepackage[style=abbrv]{databib}
```

or using:

```
\DTLbibliographystyle{abbrv}
```

Note that in the second case, the style must be set before the bibliography is displayed, otherwise they will have no effect.

Once I have set the style, I can further modify it thus:

```

\renewcommand*{\DTLtwoand}{ \& }
\renewcommand*{\DTLlandlast}{, \& }
\renewcommand*{\editorname}{ed.}
\renewcommand*{\editorsname}{eds.}
\renewcommand*{\pagesname}{pp.}
\renewcommand*{\pagename}{p.}
\renewcommand*{\volumename}{vol.}
\renewcommand*{\numbername}{no.}
\renewcommand*{\editionname}{ed.}
\renewcommand*{\techreportname}{T.R.}
\renewcommand*{\mscthesisname}{MSc thesis}

```

Now I can load the bibliography. For this example, I'm using both sample files (described earlier):

```
\DTLloadbbl{mybib}{sample1,sample2}
```

This time the database is sorted alphabetically by the title:

```
\DTLsortdata{mybib}{Title}
```

Remember that the required references need to be cited with `\cite` or `\nocite`. For simplicity, I have again used:

```
\nocite{*}
```

to select them all. As with the previous examples, the bibliography is displayed with:

```
\DTLbibliography{mybib}
```

The first page is shown in Example 187.

7.10.6. Highlight a Given Author

Suppose my boss wants me to produce a list of all my publications (from the file `sample1.bib`, as in Example 185). In my real file `nlct.bib`, most of my publications have multiple co-authors, but suppose my boss would like me to highlight my name in bold so that when he skims through the document, he can easily see my name in the list of co-authors. I can do this by redefining `\DTLformatauthor` so that it checks if the given surname matches mine. (This assumes that none of the other co-author's share my surname.)

188

```
\renewcommand*{\DTLformatauthor}[4]{%
  {% scope the font change
    \DTLifstringeq{#2}{Talbot}{\bfseries }{}%
    \DTLformatforenames{#4}
    \DTLformatvon{#1}%
    \DTLformatsurname{#2}%
    \DTLformatjr{#3}%
  }%
}
```

I have used `\DTLifstringeq` (described in §2.4.1) to perform the string comparison.

If one or more of my co-authors shared the same surname as me, I would also have had to check the first name, however there is regrettably a lack of consistency in my `bib` file when it comes to my forenames. Sometimes my name is given as “Nicola L. C. Talbot”, sometimes the middle initials are omitted, “Nicola Talbot”, or sometimes, just initials are used, “N. L. C. Talbot”. This can cause problems when checking the forenames, but as long as the other authors who share the same surname as me, don't also share the same first initial, I can use `\DTLifStartsWith`

↑ Example 187: Compact Bibliography



References

- [1] P. Macaw. *Annotated notes on the Duck and Goose chronicles*. Duck Duck Goose, 2012.
- [2] D. Duck, J. Arara, & P. Parrot. Avian friendship. *Fowl Times*, 7(5):1032–5, 2018.
- [3] N. Talbot. bib2gls: Standalone entries and repeated lists (a little book of poisons). *TUGboat*, 43(1), Apr. 2022.
- [4] N.L.C. Talbot. Creating a glossary without using an external indexing application. In *LaTeX.net*, Sept. 2012. Originally posted on the L^AT_EX Community’s Know How Section.
- [5] P. Parrot & D. Duck. *Duck and Goose* cheat sheet. Limited print run, Dubious Student Resources, 2013.
- [6] B. Ing. *Duck and Goose: an allegory for modern times?* PhD thesis, Department of Literature, University of Somewhere, Mar. 2010.
- [7] D. Duck. *Feathered stunt doubles: The Birds and other films*. Duck Duck Goose, 2016.
- [8] Z. Zebra & M. Canary. Health and safety when handling mind-controlling cookies. T.R., Secret Lab of Experimental Stuff, 22 Mar. 2014.
- [9] Henpecked: Time for a coup in the coop! Flyer.
- [10] N.L.C. Talbot. *L^AT_EX for Administrative Work*, vol. 3 of *Dickimaw L^AT_EX Series*. Dickimaw Books, Sept. 2014.
- [11] N.L.C. Talbot. *L^AT_EX for Complete Novices*, vol. 1 of *Dickimaw L^AT_EX Series*. Dickimaw Books, Sept. 2012.
- [12] N. Talbot. Localisation of T_EX documents: tracklang. *TUGboat*, 37(3), Nov. 2016.
- [13] A. von Parrot, Jr. *My Friend is a Duck*. Duck Duck Goose, 2012.
- [14] P. Parrot. Pollybot: the next generation in avian translators. In *Avian Advances*, Apr. 2021.
- [15] N.L.C. Talbot & M. Pritchett. *Quack, Quack, Quack. Give My Hat Back!* Dickimaw Books, May 2013.
- [16] M. Canary. *Ray Gun User Guide*. Secret Lab of Experimental Stuff, 2nd ed., 2015.
- [17] N.L.C. Talbot. Sorting glossaries with bib2gls. In *LaTeX.net*, July 2020.
- [18] S. Quackalot. *The Adventures of Duck and Goose*. Duck Duck Goose, 2011.

7. Converting a BibTeX database into a datatool database (databib package)

or `\DTLisPrefix`, which are described in §2.4.1.2 and §2.4.2, respectively. The following uses the first approach:

```
\renewcommand*{\DTLformatauthor}[4]{%
  {% scope font change
    \DTLifstringeq{#2}{Talbot}{\DTLifStartsWith{#4}{N}
  {\bfseries }{}{}{}%
    \DTLformatforenames{#4}
    \DTLformatvon{#1}%
    \DTLformatsurname{#2}%
    \DTLformatjr{#3}%
  }%
}
```

The data is loaded and sorted as for Example 185:

```
\DTLloadbbl{mybib}{sample1}
\DTLsortdata{mybib}{Year=descending,Month=descending}

\nocite{*}
```

In this case there's no filtering:

```
\DTLbibliography{mybib}
```

The result is shown in Example 188.

7.10.7. Separate Bib Types

Suppose now my boss has decided that I need to produce a list of all my publications, but they need to be separated so that all the journal papers appear in one section, and all the conference papers appear in another section. The journal papers need to be labelled [J1], [J2] and so on, while the conference papers need to be labelled [C1], [C2] and so on. (My boss isn't interested in any of my other publications!) Again, I'm going to use the `sample1.bib` sample file, with the data sorted in reverse chronological order (newest to oldest):

```
\DTLloadbbl{mybib}{sample1}
\DTLsortdata{mybib}{Year=descending,Month=descending}
```

189

↑ Example 188: Highlighting a given author



References

- [1] **Nicola Talbot**. bib2gls: Standalone entries and repeated lists (a little book of poisons). *TUGboat*, 43(1), April 2022.
- [2] **Nicola L C Talbot**. Sorting glossaries with bib2gls. In *LaTeX.net*, July 2020.
- [3] **Nicola Talbot**. Localisation of T_EX documents: tracklang. *TUGboat*, 37(3), November 2016.
- [4] **Nicola L C Talbot**. *L^AT_EX for Administrative Work*, volume 3 of *Dickimaw L^AT_EX Series*. Dickimaw Books, September 2014.
- [5] **Nicola L C Talbot** & Magdalene Pritchett. *Quack, Quack, Quack. Give My Hat Back!* Dickimaw Books, May 2013.
- [6] **Nicola L C Talbot**. *Using L^AT_EX to Write a PhD Thesis*, volume 2 of *Dickimaw L^AT_EX Series*. Dickimaw Books, March 2013.
- [7] **Nicola L C Talbot** & Magdalene Pritchett. *The Foolish Hedgehog*. Dickimaw Books, November 2012.
- [8] **Nicola L C Talbot**. *L^AT_EX for Complete Novices*, volume 1 of *Dickimaw L^AT_EX Series*. Dickimaw Books, September 2012.
- [9] **Nicola L C Talbot**. Creating a glossary without using an external indexing application. In *LaTeX.net*, September 2012. Originally posted on the L^AT_EX Community’s Know How Section.
- [10] Peter Flom, Hans Hagen, Joe Hogg, **Nicola Talbot**, Philip Taylor, Christina Thiele, & David Walden. What is T_EX? *The PracT_EX Journal*, 3, 2005.

7. Converting a BibTeX database into a datatool database (databib package)

All entries are selected by BibTeX:

```
\nocite{*}
```

The journal papers are in the first section. I'm using the article class, which provides `\refname` for the bibliography title, so that can be redefined as applicable:

```
\renewcommand*{\refname}{Journal Papers}
```

Next the widest label is computed for all entries that have article in the `EntryType` column:

```
\DTLcomputewidestbibentry{\equal{\DBIBentrytype}{article}}{mybib}{J\theDTLbibrow}{\widest}
```

This will define my custom `\widest` command to the widest label, which can then be passed to the `thebibliography` environment.

```
\begin{thebibliography}{\widest}
\DTLforeachbibentry[\equal{\DBIBentrytype}{article}]{mybib}{%
\bibitem[J\theDTLbibrow]{\DBIBcitekey} \DTLformatbibentry}
\end{thebibliography}
```

Similarly for the conference papers:

```
\renewcommand*{\refname}{Conference Papers}
\DTLcomputewidestbibentry{\equal{\DBIBentrytype}{inproceedings}}{mybib}{C\theDTLbibrow}{\widest}
\begin{thebibliography}{\widest}
\DTLforeachbibentry[\equal{\DBIBentrytype}{inproceedings}]{mybib}{%
\bibitem[C\theDTLbibrow]{\DBIBcitekey} \DTLformatbibentry}
\end{thebibliography}
```

The result is shown in Example 189. Note that this involves multiple iterations over the database. It would be more efficient to gather the required information (that is, testing and calculating the widest labels) in one iteration. However, that's more complicated to code.

↑ Example 189: Separate List of Journals and Conference Papers

Journal Papers

- [J1] Nicola Talbot. bib2gls: Standalone entries and repeated lists (a little book of poisons). *TUGboat*, 43(1), April 2022.
- [J2] Nicola Talbot. Localisation of T_EX documents: tracklang. *TUGboat*, 37(3), November 2016.
- [J3] Peter Flom, Hans Hagen, Joe Hogg, Nicola Talbot, Philip Taylor, Christina Thiele, & David Walden. What is T_EX? *The PracT_EX Journal*, 3, 2005.

Conference Papers

- [C1] Nicola L C Talbot. Sorting glossaries with bib2gls. In *LaTeX.net*, July 2020.
- [C2] Nicola L C Talbot. Creating a glossary without using an external indexing application. In *LaTeX.net*, September 2012. Originally posted on the L^AT_EX Community's Know How Section.

7.10.8. Multiple Bibliographies

Suppose I need to create a document which contains a section listing all my publications, but I also need to have separate sections covering my fiction and non-fiction work, with a mini-bibliography at the end of each section. As in the earlier examples, I'm using the `sample1.bib` file. Note that there will be some duplication as the references in the mini-bibliographies will also appear in the main bibliography at the end of the document, but using `\DTLcite` and `\DTLmbibliography` ensures that all the cross-referencing labels (and hyperlinks if they are enabled) are unique. As with the previous examples, I'm using the article class to keep it simple. The `auto` option can be used to automatically run BibTeX:

```
\usepackage[auto]{databib}
```

190

7. Converting a BibTeX database into a datatool database (databib package)

The following preamble command will create the files `nonfiction.aux` and `fiction.aux`:

```
\DTLmultibibs{nonfiction,fiction}
```

The `bbl` files may be loaded anywhere before the databases need to be referenced:

```
\DTLloadmbbl{nonfiction}{nonfictionDB}{sample1}  
\DTLloadmbbl{fiction}{fictionDB}{sample1}  
\DTLloadbbl{fullDB}{sample1}
```

The databases can then be sorted, if required. In this case, the full bibliography is sorted chronologically, but the other databases aren't sorted, so they will be in order of citation:

```
\DTLsortdata{fullDB}{Year,Month}
```

The document is as follows:

```
\section{Non Fiction}  
In this section I'm going to describe some \LaTeX work,  
and in the process I'm going to cite some related  
papers \DTLcite{nonfiction}{tugboat2016,Talbot2012a}  
.  
  
\DTLmbibliography{nonfiction}{nonfictionDB}  
  
\section{Fiction}  
In this section I'm going to describe my fiction, and in the pro  
to cite some books \DTLcite{fiction}  
{Talbot2013b,Talbot2012b}  
  
\DTLmbibliography{fiction}{fictionDB}
```

Finally, all entries are selected for the main list:

```
\nocite{*}  
\renewcommand{\refname}  
{Complete List of Publications}  
\DTLbibliography{fullDB}
```

7. Converting a BibTeX database into a datatool database (databib package)

If the document source is in `myDoc.tex`, then the build process with the default `auto=false` is:

```
pdflatex myDoc
bibtex myDoc
bibtex nonfiction
bibtex fiction
pdflatex myDoc
pdflatex myDoc
```

With `auto=true`, the shell escape is required. Since `bibtex` is normally on the restricted list and the restricted shell escape is normally on by default, the build process is then:

```
pdflatex myDoc
pdflatex myDoc
pdflatex myDoc
```

The resulting document is shown in Example 190.

7.11. Localisation

As described in §2.3, the `datatool-base` package (which will automatically be loaded by the `databib` package, if not already loaded) provides localisation support via the `tracklang` interface. The `databib` package uses the same interface to load the file `databib-⟨locale⟩.ldf` for each tracked locale if that file is installed on TeX's path.

The supplementary `datatool-english` package (which needs to be installed separately), described in §2.3.5, includes `databib-english.ldf`, which provides English localisation support for the `databib` package. This file may be used as a template for other languages.

Any localisation options specific to `databib` should be identified by `⟨locale⟩ / databib`. For example, `databib-english.ldf` provides the option `short-month-style`, which may have the value `dotted` (abbreviated month names are followed by a dot) or `dotless` (three letter abbreviation month names with no dot). For example, to switch to `dotless`:

```
\DTLsetLocaleOptions[en]{databib}{short-month-  
style=dotless}
```

Or:

Example 190: Multiple Bibliographies

1

Non Fiction

In this section I'm going to describe some BTeXwork, and in the process I'm going to cite some related papers [1, 2].

References

[1] Nicola Talbot. Localisation of T_EX documents: tracklang. *TUGboat*, 37(3), November 2016.

[2] Nicola L C Talbot. *BTeX for Complete Novices*, volume 1 of *Dickimaw BTeX Series*. Dickimaw Books, September 2012.

2

Fiction

In this section I'm going to describe my fiction, and in the process, I'm going to cite some books [1, 2].

References

[1] Nicola L C Talbot & Magdalene Pritchett. *Quack, Quack, Quack. Give My Hat Back!* Dickimaw Books, May 2013.

[2] Nicola L C Talbot & Magdalene Pritchett. *The Foolish Hedgehog*. Dickimaw Books, November 2012.

Complete List of Publications

[1] Peter Flom, Hans Hagen, Joe Hogg, Nicola Talbot, Philip Taylor, Christina Thiele, & David Walden. What is T_EX? *The PracT_EX Journal*, 3, 2005.

[2] Nicola L C Talbot. *BTeX for Complete Novices*, volume 1 of *Dickimaw BTeX Series*. Dickimaw Books, September 2012.

[3] Nicola L C Talbot. Creating a glossary without using an external indexing application. In *LaTeX.net*, September 2012. Originally posted on the BTeX Community's Know How Section.

[4] Nicola L C Talbot & Magdalene Pritchett. *The Foolish Hedgehog*. Dickimaw Books, November 2012.

[5] Nicola L C Talbot. *Using BTeX to Write a PhD Thesis*, volume 2 of *Dickimaw BTeX Series*. Dickimaw Books, March 2013.

[6] Nicola L C Talbot & Magdalene Pritchett. *Quack, Quack, Quack. Give My Hat Back!* Dickimaw Books, May 2013.

[7] Nicola L C Talbot. *BTeX for Administrative Work*, volume 3 of *Dickimaw BTeX Series*. Dickimaw Books, September 2014.

[8] Nicola Talbot. Localisation of T_EX documents: tracklang. *TUGboat*, 37(3), November 2016.

[9] Nicola L C Talbot. Sorting glossaries with bib2gls. In *LaTeX.net*, July 2020.

[10] Nicola Talbot. bib2gls: Standalone entries and repeated lists (a little book of poisons). *TUGboat*, 43(1), April 2022.

575

7. Converting a *BibTeX* database into a *datatool* database (*datbib* package)



```
\DTLsetLocaleOptions{en/databib}{short-month-  
style=dotless}
```

8. Creating an index, glossary or list of abbreviations (datagidx package)



```
\usepackage[<options>]{datagidx}
```

The datagidx package can be used to generate indexes or glossaries as an alternative to packages such as glossaries. It was written in 2013 before I added `\printnoidxglossary` to the glossaries package in 2014 (which uses $\TeX$ to sort and collate) and `\printunsrtglossary` to the glossaries-extra package in 2016 (which doesn't sort or collate). To avoid duplication of effort, I don't intend providing new features for datagidx. The most flexible solution is to use glossaries-extra and bib2gls, see the Dickimaw Books Gallery¹ for examples.



If you are considering datagidx because you are having difficulty running an external indexing application, see Incorporating makeglossaries or makeglossaries-lite or bib2gls into the document build.^a

^adickimaw-books.com/latex/buildglossaries/

The datagidx package was rewritten in version 3.0 to use $\LaTeX$ 3 commands. The xkeyval and afterpage packages have been dropped and the use of `\DTLforeach` has been replaced with `\DTLmapdata`. A number of bugs have also been fixed, which may cause some differences in the output. You may need to delete the temporary files before rebuilding after upgrading to v3.0. Rollback to version 2.32 is available:



```
\usepackage{datagidx}[=2.32]
```

Note that if datatool hasn't already been loaded, this will also apply rollback to datatool. Problems may occur if a newer release of datatool has already been loaded.



The datagidx package is incompatible with the glossaries package.

¹dickimaw-books.com/gallery

8. Creating an index, glossary or list of abbreviations (*datagidx* package)

The *datagidx* package works by having a predefined database called *datagidx* which contains a list of each index/glossary with its associated settings. Whenever you define a new index/glossary database with `\newgidx` (see §8.2), a new row is added to the *datagidx* database and a new database is created in which to store each entry associated with the particular index or glossary.

Whenever a term is referenced using a command such as `\useentry` the relevant information is looked up in the database. A mapping exists between the entry label and the index/glossary database label to ensure that the correct database is referenced. Unlike some of the other supplementary packages provided with *datatool*, the *datagidx* package doesn't use the `default-name` setting, but instead has its own way of identifying the default index/glossary database, which is independent of `default-name`.

The list of referenced terms can then be displayed with `\printterms`, which firsts sorts the database using `\DTLsortdata`.



Don't expect *datagidx* to perform as efficiently as an application that is designed specifically to sort and collate entries.

8.1. Options

Any options that may be passed to *datatool* can also be passed to *datagidx* (unless *datatool* has already been loaded and the option isn't permitted in `\DTLsetup`). Additionally, the package options described in §8.1.1 are specific to *datagidx*.

The *datagidx* package provides the `\DTLsetup` option `index`:

```
\DTLsetup{index={\options}}
```

This may be used to set any of the options listed §8.1.2.

8.1.1. Package Options

8.1.1.1. Global Options



optimize=*<value>*

default: **high**; initial: **off**

This setting may only be passed as a package option, and sets the optimization mode. The value must be one of: `off` (no optimization), `low` (some optimization), or `high` (highest level). If you want to optimize some glossaries but not others, switch on the `optimize` function and clear the `sort` option for the relevant glossaries and manually sort using `\DTLsortdata` or `\dtlsort` before the glossary is displayed. See §8.8.3.1 for further details.

draft

This valueless setting may only be passed as a package option, and switches on draft mode, which displays additional information, such as target names.

final

This valueless setting (which is the default) may only be passed as a package option, and switches off draft mode.

The following options are also considered global options but may be changed after the package has been loaded.

nowarn= $\langle$ *boolean* $\rangle$

default: **true**; initial: **false**

This setting may only be passed as a package option, and, if true, will switch off datagidx warnings. If you want to change the setting again later, you can use the **warn** option in the **index** sub-setting.

compositor= $\{ \langle$ *character* $\rangle \}$

initial: .

Sets the location compositor (see §8.5.2). This value may also be set after the package has loaded with the **compositor** option.

counter= $\{ \langle$ *counter name* $\rangle \}$

initial: **page**

Sets the location counter (see §8.5.2). This value may also be set after the package has loaded with the **counter** option.

8.1.1.2. Style Options

These options relate to the way the index or glossary is formatted by **\printterms**.

columns= $\langle$ *n* $\rangle$

Sets the number of columns in **\printterms**. This value may also be set after the package has loaded with the **columns** option.

child= $\langle$ *value* $\rangle$

initial: **named**

Sets the child style used in **\printterms**, where the value may be **named** (show the child's name) or **noname** (only show the numeric label, as given by **\DTLgidxChildCountLabel**, not the child's name). This value may also be set after the package has loaded with the

`child` option.

namecase= $\langle value \rangle$

Sets the name case style used in `\printterms`, where the value may be `nochange`, `uc`, `lc`, `firstuc` or `capitalise`. This value may also be set after the package has loaded with the `namecase` or `name-case` option.

postname= $\{ \langle value \rangle \}$

Sets the code to insert after the name used in `\printterms`. This value may also be set after the package has loaded with the `postname` or `post-name` option.

postdesc= $\langle value \rangle$

Sets the post-description style used in `\printterms`, where the value may be one of: `none` or `dot`. This value may also be set after the package has loaded with the `postdesc` or `post-desc` option.

prelocation= $\langle value \rangle$

Sets the style of the pre-location content used in `\printterms`, where the value may be `none`, `enspace`, `space`, `dotfill` or `hfill`. This value may also be set after the package has loaded with the `prelocation` or `pre-location` option.

location= $\langle value \rangle$

Sets the location list style used in `\printterms`, where the value may be one of: `hide`, `list` or `first`. This value may also be set after the package has loaded with the `location` option.

see= $\langle value \rangle$

Sets the “see” style used in `\printterms`, where the value may be one of: `comma`, `brackets`, `dot`, `space`, `nosep`, `semicolon`, or `location`. This value may also be set after the package has loaded with the `see` option.

symboldesc= $\{ \langle value \rangle \}$

Sets the symbol-description style used in `\printterms`, where the value may be one of: `symbol`, `desc`, `(symbol) desc`, `desc (symbol)`, `symbol desc`, or `desc symbol`. This value may also be set after the package has loaded with the `symboldesc` or

`symbol-desc` option.

8.1.2. Post-Package Options

The following options may be set after the *datagidx* package has been loaded with the `index` setting. For example:

```
\DTLsetup{index={style=gloss,columns=1}}
```

8.1.2.1. General

These settings may be localised but they apply to all index or glossary databases. They can't be set in the optional argument of `\newgidx` or `\printterms`. Ideally they should be set in the preamble.

```
compositor={⟨character⟩}
```

initial: .

Sets the location compositor (see §8.5.2). This setting may also be passed as a package option.

```
counter={⟨counter name⟩}
```

initial: page

Sets the location counter (see §8.5.2). This setting may also be passed as a package option.

```
warn=⟨boolean⟩
```

default: **true**; initial: **true**

If true, switches on *datagidx* warnings, otherwise switches them off. The `nowarn` package option is an antonym of this setting.

8.1.2.2. Database Creation and Display

These options are applicable to both `\newgidx` and `\printterms`. They may be set in the optional arguments of those commands or set with `index` in `\DTLsetup`. If used in the optional argument of `\newgidx`, they set the default for that database, which can then be overridden by the optional argument of `\printterms`, if necessary. Any options not explicitly set in `\newgidx` will use the current setting.

```
balance=⟨boolean⟩
```

default: **true**; initial: **true**

If true and the `columns` value is greater than 1, balance columns. That is, the unstarred multicols environment (provided by the *multicol* package) will be used. If `balance=false`,

8. Creating an index, glossary or list of abbreviations (*datagidx* package)

then `\twocolumn` will be used if `columns=2` or, if more than two columns, the `multicols*` environment will be used. The `balance` option has no effect with `columns=1`.

```
heading={⟨title⟩}
```

Sets the index/glossary heading.

```
post-heading={⟨code⟩}
```

Sets the content to insert after the heading.

```
postheading={⟨code⟩} alias: post-heading
```

Synonym of `post-heading`.

```
show-groups=⟨boolean⟩ default: true; initial: true
```

If true, visually separate the letter groups. This requires the `LetterGroup` column to be set, which is done by the default `sort` code.

```
showgroups=⟨boolean⟩ alias: show-groups; default: true; initial: true
```

Synonym of `show-groups`.

```
sort={⟨sort-code⟩}
```

The value should be the code to sort the database at the start of `\printterms`. The default is:

```
\DTLsortdata
[ save-group-key=LetterGroup ]
{ \DTLgidxCurrentdb }
{ HierSort={ replacements={Sort}}, FirstId }
```

The `save-group-key=LetterGroup` option is needed to support letter groups. If you omit it from `\DTLsortdata` (or if you use `\dtlsort` instead) then no letter group information will be available. See §8.10 for the list of column keys, which may be referenced in `\DTLsortdata`.

```
style={⟨name⟩} initial: index
```

Sets the style to be used by `\printterms` (see §8.8.2).

8.1.2.3. Display Only

These options are only applicable to `\printterms`. They can't be passed in the optional argument of `\newgidx`. If not provided in `\printterms`, the current settings will be used.

`child={⟨value⟩}`

initial: **named**

Sets the child style. The value may be one of: `named` (show the child's name) or `noname` (only show the numeric label, as given by `\DTLgidxChildCountLabel`, not the child's name). This setting may also be passed as a package option.

`child-sort=⟨boolean⟩`

default: **true**; initial: **true**

If this boolean option is true, the child sub-lists should follow the ordering from the database (which will be the sort order, if the database has been sorted), otherwise they will follow the order of the comma-separated list in `\Children` (which will be the order of definition).

`childsort=⟨boolean⟩`

alias: `child-sort`; default: **true**; initial: **true**

Synonym of `child-sort`.

`columns=⟨n⟩`

initial: **2**

The number of columns. If the value is greater than 1, the `multicols` or `multicols*` environment will be used, depending on the `balance` setting, except where both `columns=2` and `balance=false`, in which case `\twocolumn` will be used. This setting may also be passed as a package option.

If the `columns` setting is used as a package option or within the `index` setting, it's equivalent to setting the number of columns with `\DTLgidxSetColumns`. If passed as an option to `\printterms` the effect is localised.

`condition={⟨condition⟩}`

Only include rows that satisfy the condition, where `⟨condition⟩` must be suitable for use in `\if-thenelse` (see §2.4.2). Note that `condition={⟨condition⟩}` is equivalent to:

`include-if={\ifthenelse{⟨condition⟩}{#1}{}}`

`database={⟨db-name⟩}`

initial: **empty**

Specifies the datagidx database name. If used within the `index` setting, this is equivalent to

8. Creating an index, glossary or list of abbreviations (*datagidx* package)

setting the default database with `\DTLgidxSetDefaultDB`. If passed as an option to `\printterms` the effect is localised. If no default has been set, `\newgidx` will automatically set the default to its `<db-name>` argument. So if you only have one database, there is no need to explicitly set the default.



If a database is loaded from an external file with `\loadgidx`, then the default database will automatically be set to `\dtllastloadeddb`.



`include-if=<definition>`

An alternative to the `condition` option, this defines the filter command used by `\printterms` to `<definition>`, which should expand to its sole argument (`#1`) if the given row of data should be included in the list and do nothing otherwise. This may be used as an alternative to `include-if-fn` or `condition`.



`include-if-fn=<cs>`

An alternative to the `condition` option, this sets the filter command, which should take one argument, to the given `<cs>`. The definition of `<cs>` should expand to its argument if the given row of data should be included in `\printterms` and do nothing otherwise.



The options `condition`, `include-if` and `include-if-fn` all override each other.



`location={<value>}`

initial: **list**

Sets the location style. The value must be one of: `hide` (don't show the location list), `list` (show the location list), or `first` (only show the first location). This setting may also be passed as a package option.



`location-width=<dim>`

initial: **0pt**

Sets the space to allocate for each location list. If zero or negative, the location list just occupies its natural width. This setting changes the `\datagidxlocationwidth` dimension.



`locationwidth=<dim>`

alias: `location-width`; *initial:* **0pt**

Synonym of `location-width`.

```
name-case={⟨value⟩}
```

initial: **nochange**

Specifies the case-change to apply to the name. The value must be one of: **nochange** (no change), **uc** (convert to uppercase), **lc** (convert to lowercase), **firstuc** (convert to sentence case with `\xmakefirstuc`), or **capitalise** (capitalise each word with `\xcapitalisewords`). See the **mfirstuc** documentation for further details of the last two. This setting redefines `\DTLgidxNameCase` to use the applicable command.

```
namecase={⟨value⟩}
```

alias: **name-case**; *initial:* **nochange**

Synonym of **name-case**. This setting (but not **name-case**) may also be passed as a package option.

```
name-font={⟨code⟩}
```

The font declaration or text block command to apply to the name. The code may contain multiple declarations. Only the final token may be a text block command. This setting redefines `\DTLgidxNameFont`:

```
\renewcommand*{\DTLgidxNameFont}[1]{\{⟨code⟩\#1\}}
```

```
namefont={⟨code⟩}
```

alias: **name-font**

Synonym of **name-font**. This setting (but not **name-font**) may also be passed as a package option.

```
post-name={⟨value⟩}
```

initial: `\`

Content to insert after the name. This setting redefines `\DTLgidxPostName` to the given *⟨value⟩*.

```
postname={⟨value⟩}
```

alias: **post-name**

Synonym of **post-name**. This setting (but not **post-name**) may also be passed as a package option.

```
post-desc={⟨value⟩}
```

initial: **none**

The value may be one of: **none** (no content to insert after the description) or **dot** (insert a period/full stop after the description). This setting redefines `\DTLgidxPostDescrip-`

tion according to the given value.

`postdesc={⟨value⟩}`

alias: `post-desc`

Synonym of `post-desc`. This setting (but not `post-desc`) may also be passed as a package option.

`pre-location={⟨value⟩}`

initial: **enspace**

Indicates the type of padding that should be inserted before the location list. The value may be one of: `none` (nothing), `enspace` (an en-space, created with `\enspace`), `space` (a normal space), `dotfill` (dotted leader, created with `\dotfill`), `hfill` (filled space, created with `\hfill`). This setting redefines `\DTLgidxPreLocation` according to its value.

`prelocation={⟨value⟩}`

alias: `pre-location`

Synonym of `pre-location`. This setting (but not `pre-location`) may also be passed as a package option.

`see={⟨value⟩}`

initial: **location**

Sets the “see” cross-reference style. This setting may also be passed as a package option. The value must be one of: `comma` (insert a comma and space, followed by the cross-reference), `dot` (insert a period/full stop and space, followed by the cross-reference), `semicolon` (insert a semi-colon and space, followed by the cross-reference), `brackets` (insert a space followed by the cross-reference in parentheses), `space` (insert a space followed by the cross-reference), `location` (insert `pre-location` content followed by the cross-reference), or `nosep` (insert the cross-reference without any leading punctuation or space). In all cases, the cross-reference is obtained with:

`\DTLgidxFormatSee{⟨seename⟩}{⟨See⟩}`

`symbol-desc={⟨value⟩}`

Specifies how the symbol and description should be displayed. The value must be one of the following:

- `symbol`: symbol only;
- `desc`: description only;
- `(symbol) desc`: the symbol in parentheses followed by the description;

8. Creating an index, glossary or list of abbreviations (*datagidx* package)

- `desc (symbol)`: the description followed by the symbol in parentheses;
- `symbol desc`: the symbol followed by the description;
- `desc symbol`: the description followed by the symbol.

`symboldesc={⟨value⟩}`

alias: `symbol-desc`

Synonym of `symbol-desc`. This setting (but not `symbol-desc`) may also be passed as a package option.

`symbol-width=⟨dim⟩`

initial: **0pt**

Sets the space to allocate for each symbol. If zero or negative, the symbol just occupies its natural width. This setting changes the `\datagidxsymbolwidth` dimension.

`symbolwidth=⟨dim⟩`

alias: `symbol-width`; initial: **0pt**

Synonym of `symbol-width`.

8.2. Defining Index/Glossary Databases

Before you define any terms, you need to create the database in which to store the information. This is done with:

`\newgidx[⟨options⟩]{⟨db-name⟩}{⟨title⟩}`

preamble only

This creates a new database called `⟨db-name⟩`, which has an associated title (`⟨title⟩`) for use with `\printterms`. For example:

```
\newgidx{index}{Index of Terms}
\newgidx{abbr}{Abbreviations}
```

The optional argument is a `⟨key⟩=⟨value⟩` list of options, which may be any of those described in §8.1.2.2.

This does more than simply create the database with `\DTLnewdb` as it also adds a line to the cataloguing database with the provided `⟨title⟩` and all applicable settings (see §8.10 for further details). It also takes into account the `optimize` setting (see §8.8.3.1).



If no *datagidx* default database has yet been specified, `\newgidx` will automatically set the default to `<db-name>` (so you won't need to specify `database` in `\printterms` if you only have one database). Note that this isn't the same as the *datatool* `default-name` setting.

8.3. Loading Data Created by *datatooltk*

If you have edited and sorted a *datagidx* database in *datatooltk*, you can then just load it using:



```
\loadgidx[<options>]{<filename>}{<title>}
```

where `<filename>` is the name of the DBTEX file saved by *datatooltk*. The remaining arguments `<options>` and `<title>` are the same as for `\newgidx`, described in §8.2. This command automatically sets the default database to the loaded database. You can change the default database using `\DTLgidxSetDefaultDB` or with the `database` setting.

Since `\loadgidx` is intended for use with presorted databases, this implements `sort={}` which means that `\printterms` won't sort the database.



Both *datatooltk* and *bib2gls* are Java applications. If you're going to use an external helper application, you're better off using *bib2gls* with *glossaries-extra* than using *datatooltk* with *datagidx*.

8.4. Defining Terms

A new term may be defined in the preamble with:



```
\newterm[<options>]{<name>}
```

preamble only

(For abbreviations, see §8.7.) This defines a term with the given `<name>`, which is the text to display in the list typeset with `\printterms`. The optional argument is a `<key>=<value>` list of options, described below. (See §8.6 if you want additional fields with associated options.)

If *datatool*-base's `verbose` mode is on, `\newterm` will write a confirmation line in the transcript. This may be used to double-check the default values for the `label`, `database` and `sort` settings.

database= $\{ \langle db-name \rangle \}$

The label of the database in which this term should be stored. If omitted, the *datagidx* default database is assumed.

label= $\{ \langle label \rangle \}$

The term's label. Each term must have a unique label that identifies it, so that it can be referenced by commands like `\useentry` and `\gls`. If hyperlinks are required, the label is also used to construct the anchor. If a label is not provided in the optional argument, the label will be created as described in §8.4.2.

parent= $\{ \langle parent-label \rangle \}$

The label of the term's parent, if a hierarchical structure is required. The parent must belong to the same database as the child entry.

text= $\{ \langle text \rangle \}$

The text to use by commands like `\gls`. If omitted, the $\langle name \rangle$ is used.

plural= $\{ \langle text \rangle \}$

The text to use by commands like `\glspl`. If omitted, the plural is constructed by appending “s” to the `text` value.

symbol= $\{ \langle text \rangle \}$

The text to use by commands like `\glsym`. It may also be shown in `\printterms`, depending on the `symbol-desc` setting.

description= $\{ \langle text \rangle \}$

The term's description to be displayed in `\printterms`, if applicable.

see= $\{ \langle xr-labels \rangle \}$

If the term should have a “see” cross-reference in `\printterms`, the label of the cross-reference term should be set with the `see` option. The value may also be a comma-separated list of cross-referenced labels.

```
seealso= { <xr-labels> }
```

If the term should have a “see also” cross-reference in `\printterms`, the label of the cross-reference term should be set with the `seealso` option. The value may also be a comma-separated list of cross-referenced labels.

```
sort= { <text> }
```

The term’s sort value. If omitted, the value is obtained by processing the *<name>*, as described in §8.4.3.

The sort value is stored in the database’s `Sort` column, which is referenced in the default `sort` code at the start of `\printterms`. If you change the `\printterms sort` code, use `Sort` as the column key to reference this value. However, if you have child entries, you may prefer to sort by the `HierSort` column first with the `Sort` column as the replacement.

The term’s hierarchical sort value, which is stored in the `HierSort` column and only set for child entries, is obtained by appending the sort value to the parent’s `HierSort` value (or the parent’s `Sort` value, if the parent doesn’t have a parent), separated by:

```
\datatoolctrlboundary \datatoolasciistart
```

The following options are provided for use by `\newacro`, which internally uses `\newterm`. Whilst they can be explicitly used in `\newterm`, `\newacro` is a convenient wrapper that will encapsulate the short and long forms in the provided abbreviation style commands. See §8.7 for further details.

```
short= { <text> }
```

The short (abbreviated) form.

```
shortplural= { <text> }
```

The plural short form. If omitted, it will be obtained by appending “s” to the `short` value.

```
long= { <text> }
```

The long form.

```
longplural= { <text> }
```

The plural long form. If omitted, it will be obtained by appending “s” to the `long` value.

8.4.1. Markup Commands for Terms

The commands described below may be used in the term's name but expand differently depending on the context. Remember that if the `text`, `label` or `sort` values aren't provided, they will be obtained from the name. These commands may be nested. For example:

```
\newterm{%
  \DTLgidxOffice
  {bishop \DTLgidxParticle{of}{Bayeux}}% office
  {\DTLgidxName{Henry}\DTLgidxParticle{de}{Beaumont}}
% name
}
```

This is equivalent to:

```
\newterm[
  label={deBeaumont},
  sort={Beaumont.\datatoolpersoncomma
    Henry\datatoolpersoncomma bishop Bayeux.}
]{%
  \DTLgidxOffice
  {bishop \DTLgidxParticle{of}{Bayeux}}% office
  {\DTLgidxName{Henry}\DTLgidxParticle{de}{Beaumont}}
% name
}
```

```
\DTLgidxName{<forename>}{<surname>}
```

Normally this expands to `<forename> <surname>`, but it expands differently within `\print-`terms and also in the construction of the default `label` and `sort` values. For example:

```
\newterm{\DTLgidxName{John}{Smith}}
```

This is essentially like:

```
\newterm[
  label={Smith},
  sort={Smith\datatoolpersoncomma John},
  text={John Smith}
]{Smith, John}
```



```
\DTLgidxNameNum{⟨num⟩}
```

Designed to markup a name ordinal (such as for a monarch). This normally expands to the uppercase Roman numeral of the given number, but will expand to *⟨num⟩* zero-padded to two digits when creating the default `sort` value. For example:



```
\newterm{James~\DTLgidxNameNum{6}}
```

This is essentially like:

```
\newterm[
  label={James VI},
  sort={James 06},
  text={James~VI}
]{James~VI}
```



```
\DTLgidxPlace{⟨country⟩}{⟨town/city⟩}
```

Used to format a place, this normally expands to just its second argument *⟨town/city⟩*, but it expands differently within `\printterms` and also in the construction of the default `label` and `sort` values. For example:



```
\newterm{\DTLgidxPlace{USA}{New York}}
```

This is essentially like:

```
\newterm[
  label={New York USA},
  sort={New York\datatoolplacecomma USA},
  text={New York}
]{New York, USA}
```



```
\DTLgidxSubject{⟨main⟩}{⟨category⟩}
```

Used to format a subject, this normally expands to just its second argument *⟨category⟩*, but it expands differently within `\printterms` and also in the construction of the default `label` and `sort` values. For example:

```
\newterm{\DTLgidxSubject{population}{New York}}
```

This is equivalent to:

```
\newterm[
  label={New York population},
  sort={New York\datatoolsubjectcomma population}
]{\DTLgidxSubject{population}{New York}}
```

which is essentially like:

```
\newterm[
  label={New York population},
  sort={New York\datatoolsubjectcomma population},
  text={New York}
]{New York, population}
```

In this case, this leads to a long label, so you may prefer to set a shorter label explicitly. If you redefine `\DTLgidxSubject`, it will only affect how the text appears in the main body of the document, not within `\printterms`.

```
\DTLgidxOffice{<office>}{<name>}
```

Used to markup a person's office, this normally expands to `<name> (<office>)`, but it expands differently within `\printterms` and also in the construction of the default label and sort values. For example:

```
\newterm{\DTLgidxOffice{Rollo}{ruler of Normandy}}
```

This is equivalent to:

```
\newterm[
  label={ruler of Normandy},
  sort={ruler of Normandy\datatoolpersoncomma Rollo}
]{\DTLgidxOffice{Rollo}{ruler of Normandy}}
```

which is essentially like:

8. Creating an index, glossary or list of abbreviations (*datagidx* package)

```
\newterm[
  label={ruler of Normandy},
  sort={ruler of Normandy\datatoolpersoncomma Rollo}
,
  text={Rollo (ruler of Normandy)}
]{ruler of Normandy, Rollo}
```

If you redefine `\DTLgidxOffice`, it will only affect how the text appears in the main body of the document, not within `\printterms`.

```
\DTLgidxRank{<title>}{<forename(s)/initial(s)>}
```

Used to markup a person's rank, this normally expands to `<rank>~<forename(s)/initial(s)>`, but it expands differently within the construction of the default `label` and `sort` values. For example:

```
\newterm{\DTLgidxRank{Sir}{John}}
```

This is equivalent to:

```
\newterm[
  label={Sir John},
  sort={John.}
]{\DTLgidxRank{Sir}{John}}
```

Note that this doesn't have a different definition within `\printterms`.

```
\DTLgidxParticle{<particle>}{<surname>}
```

Used to markup a surname with a particle (such as “of” or “de”), this normally expands to `<particle>~<surname>`, but it expands differently within the construction of the default `label` and `sort` values. For example:

```
\newterm{\DTLgidxParticle{de}{Winter}}
```

This is equivalent to:

8. Creating an index, glossary or list of abbreviations (*datagidx* package)

```
\newterm[
  label={deWinter},
  sort={Winter.}
]{\DTLgidxParticle{de}{Winter}}
```

Note that this doesn't have a different definition within `\printterms`.

```
\DTLgidxMac{<text>}
```

Used to markup “Mac”/“Mc” prefix at the start of a surname. Normally this simply expands to its argument, but will expand to “Mac” when creating the sort value, regardless of the argument. For example:

```
\newterm{\DTLgidxMac{Mc}Coy}
```

This is equivalent to:

```
\newterm[
  label={McCoy},
  sort={MacCoy}
]{\DTLgidxMac{Mc}Coy}
```

```
\DTLgidxSaint{<text>}
```

Used to markup “Saint”/“St” prefix. Normally this simply expands to its argument, but will expand to “Saint” when creating the sort value, regardless of the argument. For example:

```
\newterm{\DTLgidxSaint{St}~James}
```

This is equivalent to:

```
\newterm[
  label={St James},
  sort={Saint James}
]{\DTLgidxSaint{St}~James}
```

```
\DTLgidxParen{<text>}
```

8. Creating an index, glossary or list of abbreviations (*datagidx* package)

Used to markup parenthetical material, this normally expands to a space followed by its argument in parentheses, but it discard its argument within the construction of the default `label`. For example:

```
\newterm{0\DTLgidxParen{zero}}
```

This is equivalent to:

```
\newterm[
  label={0},
  sort={0\datatoolparenstart zero}
]{0\DTLgidxParen{zero}}
```

```
\DTLgidxIgnore{<text>}
```

Used to markup content that should be stripped from the default `sort` value. Expands to its argument normally. For example:

```
\newterm{de\DTLgidxIgnore{-}escalate}
```

This is equivalent to:

```
\newterm[
  label={de-escalate},
  sort={deescalate},
]{de\DTLgidxIgnore{-}escalate}
```

Note that in this case, the letter sort handler `\DTLsortletterhandler` can be used to strip hyphens.

```
\DTLgidxStripBackslash{<cs>}
```

This expands to the command name without the leading backslash. As from version 3.0, this simply uses `\cs_to_str:N`.

8.4.2. Commands to Assist Label Creation

All defined terms must be supplied a unique label in order to reference them. For example:


```
\newterm[label=duck]{duck}
```

If the name doesn't contain any markup, as in the above example, it can be a bit repetitive to have to explicitly set the label. To avoid this repetition, `\newterm` will create a label based on the name if the `label` option isn't set. If you want to double-check the label value, switch on `verbose` mode and `\newterm` will write information, including the label, to the transcript.

The automatic label creation locally redefines certain commands, performs a protected expansion, strips awkward characters and purifies the result. This means that the above example can simply be written as:

```
\newterm{duck}
```

If the name contains fragile commands, either explicitly set the `label` value or redefine `\newtermlabelhook` to locally change the fragile commands to expand to nothing or to something suitable for a label.

The steps are described in more detail below.

1. Local redefinitions of common formatting commands.

The command `\glsadd` will be redefined to ignore its argument. Various commands are redefined to `\DTLgidxNoFormat`, which simply expands to its argument. This is largely redundant now, since the purification step will strip any non-expandable commands. Commands such as `\&` are replaced via `\datagidxconvertchars`.

2. Local redefinitions of special markup commands.

Some of the markup commands described in §8.4.1 are redefined. These are: `\DTLgidxParen` (discards its argument), `\DTLgidxName` and `\DTLgidxOffice` (expands to just the second argument), `\DTLgidxPlace` and `\DTLgidxSubject` (inverted), and `\DTLgidxParticle` (expands both arguments without any separator).

3. Local redefinitions of standard math-Greek commands (such as `\alpha`) to words (via `\datagidxwordifygreek`).

4. User hook.

5. Protected expansion of the supplied name.

6. Commas and equal signs are stripped (as they can interfere with syntax parsing).

7. The remaining content is expanded and purified.

The user hook is:

```
\newtermlabelhook
```

This does nothing by default. It may be redefined to perform any additional local redefinitions.

8.4.3. Commands to Assist Sorting

If the `sort` value isn't set, it will be obtained from the name. For example:

```
\newterm{duck}
```

is equivalent to:

```
\newterm[sort=duck]{duck}
```

Some processing is performed on the name to assist the creation of a missing `sort` value. If the name contains fragile commands, either explicitly set the `sort` value or redefine `\newtermsorthook` to locally change the fragile commands to expand to nothing or to something suitable for sorting. If you want to double-check the sort value, switch on `verbose` mode and `\newterm` will write information, including the sort value, to the transcript.

The sort handler function may perform additional changes to the sort value, regardless of whether `sort` is set explicitly or obtained from the name.

The steps to create the missing `sort` value are similar to those for the label, but there are some slight differences.

1. Local redefinitions of common formatting commands.

The command `\glsadd` will be redefined to ignore its argument. Various commands are redefined to `\DTLgidxNoFormat`, which simply expands to its argument. This is largely redundant now, since the purification step will strip any non-expandable commands. Commands such as `\&` are replaced via `\datagidxconvertchars`.

2. Local redefinitions of special markup commands.

The markup commands described in §8.4.1 are redefined:

- `\DTLgidxName{⟨arg1⟩}{⟨arg2⟩}` and `\DTLgidxOffice{⟨arg1⟩}{⟨arg2⟩}` expand to `⟨arg2⟩\datatoolpersoncomma ⟨arg1⟩`;
- `\DTLgidxPlace{⟨arg1⟩}{⟨arg2⟩}` expands to `⟨arg2⟩\datatoolplacecomma ⟨arg1⟩`;
- `\DTLgidxSubject{⟨arg1⟩}{⟨arg2⟩}` expands to `⟨arg2⟩\datatoolsubjectcomma ⟨arg1⟩`;
- `\DTLgidxParen{⟨arg⟩}` expands to `\datatoolparenstart ⟨arg⟩`;
- `\DTLgidxMac{⟨arg⟩}` expands to “Mac”, ignoring its argument;
- `\DTLgidxSaint{⟨arg⟩}` expands to “Saint”, ignoring its argument;

8. Creating an index, glossary or list of abbreviations (*datagidx* package)

- `\DTLgidxIgnore{⟨arg⟩}` expands to nothing;
 - `\DTLgidxRank{⟨arg1⟩}{⟨arg2⟩}` and `\DTLgidxParticle{⟨arg1⟩}{⟨arg2⟩}` expand to `⟨arg2⟩.` (that is, they ignore their first argument and insert “.” after the second argument);
 - `\DTLgidxNameNum{⟨n⟩}` expands to its numeric argument zero-padded to two digits.
3. Local redefinitions of standard math-Greek commands (such as `\alpha`) to words (via `\datagidxwordifygreek`).
 4. User hook.
 5. Protected expansion of the supplied name.

The user hook is:

```
\newtermsorthook
```

This does nothing by default. It may be redefined to perform any additional local redefinitions.

8.5. Referencing Terms

Terms that have been defined with `\newterm` can be referenced in the document with:

```
\useentry{⟨label⟩}{⟨col-key⟩}
```

This robust command fetches the given field (the value in the column labelled `⟨col-key⟩`) for the term identified by `⟨label⟩`, displays it (in a hyperlink, if applicable), and marks the term as having been used.

See §8.10.2 for a list of the default column keys. Additional fields (see §8.6) can also be referenced if they were present when the term was defined.

For example, suppose I have previously (in the preamble) defined the term “reptile” using:

```
\newterm{reptile}
```

I can now reference this term in the document:

```
\useentry{reptile}{Text}
```

or if I want the plural, I can use:

```
\useentry{reptile}{Plural}
```

There is also a sentence case command:

```
\Useentry{<label>}{<col-key>}
```

This makes the first letter of the field value uppercase (using the *mfirstuc* package) or for all caps:

```
\USEentry{<label>}{<col-key>}
```

This converts all the text obtained from the field to uppercase.

If the *hyperref* package is loaded, the above commands will automatically create hyperlinks to the relevant entry in the index/glossary (produced with `\printterms`). You can suppress this action for individual cases by using one of the following analogous commands instead. (To suppress all hyperlinks, use `\DTLgidxDisableHyper`.)

```
\useentrynl{<label>}{<col-key>}
```

As `\useentry` but with no hyperlink. If the *hyperref* package hasn't been loaded, this is equivalent to `\useentry`.

The sentence case version is:

```
\Useentrynl{<label>}{<col-key>}
```

This is as `\Useentry` but with no hyperlink. If the *hyperref* package hasn't been loaded, this is equivalent to `\Useentry`.

The all caps version is:

```
\USEentrynl{<label>}{<col-key>}
```

This is as `\USEentry` but with no hyperlink. If the *hyperref* package hasn't been loaded, this is equivalent to `\USEentry`.

You can also specify your own custom text:

```
\glslink{<label>}{<text>}
```

This behaves like `\useentry` but displays the provided text instead of the value of a field.

In all the above commands, the `<label>` argument may optionally start with `[<format>]`, where `<format>` is the name of a control sequence name *without* the preceding backslash. This command

8. Creating an index, glossary or list of abbreviations (*datagidx* package)

will be applied to this location in the entry's location list when it's displayed in the index/glossary (produced with `\printterms`).

For example:

```
\useentry{[textbf]reptile}{Text}
```

Note that the command (`\textbf` in the above example) should take one argument (the location). If you attempt to use a declaration (such as `\bfseries`) the effect won't be localised.

You can simply display the value of a field using:

```
\glsdispenry{<label>}{<col-key>}
```

This robust command is like `\useentry` but it doesn't index or create a hyperlink.

The sentence case version is:

```
\Glsdispenry{<label>}{<col-key>}
```

This robust command is like `\Useentry` but it doesn't index or create a hyperlink.

The above commands aren't expandable. If you want to fetch a value without displaying or using it, you can use:

```
\DTLgidxFetchEntry{<cs>}{<label>}{<col-key>}
```

where `<cs>` is a control sequence, `<label>` is the label that uniquely identifies the entry and `<col-key>` is column key identifying the required field. The command `<cs>` is defined to expand to the value of that field. If the value isn't found, an error will be triggered and `<cs>` will be null (see §3.10).

Remember that you can also simply use the general purpose database commands or actions, such as `\DTLassignfirstmatch` or `select row`, to fetch information. However, you will need to specify the database name, which isn't required with `\DTLgidxFetchEntry` (since it uses the label to database mapping created by `\newterm`).

You can add an entry to the index/glossary without displaying any text using:

```
\glsadd{<label>}
```

This simply indexes the term. As with `\useentry`, `<label>` maybe in the form `[<format>]{<label>}` where `<format>` is the name of a control sequence *without* the leading backslash.

You can also add all entries from a particular database using:



```
\glsaddall{<db-name>}
```

This essentially iterates over all rows of the database identified by `<db-name>`, indexing each entry with a blank location.



Unlike the commands of the same name provided by the `glossaries` package, here there is a difference between `\glsaddall` and using `\glsadd` on all entries in the database. In the case of `\glsadd` a location is added to the location list for that entry. However in the case of `\glsaddall` no location is added to each entry's location list, but the location list is set to non-null so the entry will appear in the index/glossary.

8.5.1. Shortcut Commands

There are some shortcuts to common fields (if you are used to the `glossaries` package, note that these commands have different syntax to the commands provided by `glossaries` with the same name):



```
\gls{<label>}
```

This is equivalent to `\useentry{<label>}{Text}`.



```
\glspl{<label>}
```

This is equivalent to `\useentry{<label>}{Plural}`.



```
\glsnl{<label>}
```

This is equivalent to `\useentrynl{<label>}{Text}`.



```
\glsplnl{<label>}
```

This is equivalent to `\useentrynl{<label>}{Plural}`.



```
\Gls{<label>}
```

This is equivalent to `\Useentry{<label>}{Text}`.

`\Glspl{<label>}`

This is equivalent to `\Useentry{<label>}{Plural}`.

`\Glsnl{<label>}`

This is equivalent to `\Useentrynl{<label>}{Text}`.

`\Glsplnl{<label>}`

This is equivalent to `\Useentrynl{<label>}{Plural}`.

`\glssym{<label>}`

This is equivalent to `\useentry{<label>}{Symbol}`.

`\Glsym{<label>}`

This is equivalent to `\Useentry{<label>}{Symbol}`.

8.5.2. Locations

Each location shown in `\printterms` is, by default, the page number on which the term was referenced. The counter used for the location is obtained by expanding:

`\DTLgidxCounter`

initial: page

If redefined, this must expand to a valid counter name. The `counter` option redefines `\DTLgidxCounter` to the given value (with a check to determine that the supplied name is valid).

It's not possible to have different counters for different databases. If you want that flexibility, use the `glossaries` package instead. The most flexible solution for esoteric numbering systems that need to work with `hyperref` is to use `glossaries-extra` with the `record-nameref` package option and `bib2gls`.

Each location is saved in the `aux` file and added to the `Location` field list for the relevant row of the database on the next run. The location list is then displayed (if supported by the style) at the end of the relevant row in `\printterms` using `\DTLgidxLocation` (which is

modified by the `location` option). The entire content of the `Location` field is encapsulated with the special marker `\dtlspecialvalue`.

A location may be in the form `<prefix><c><num>` where `<c>` is a character used to separate components of the location. This character is the *compositor*. The default compositor is a period/full stop but may be changed with the `compositor` option or with:

```
\DTLgidxSetCompositor{<character>}
```

The argument should be a single character. For most documents, the page numbers are a simply unprefix number, in which case this setting may be ignored. However, if you change the location counter, for example to section, you may need to take the compositor into account.

8.6. Adding Extra Fields

If you want additional fields, you need to create a new column in the database and a corresponding option for use in `\newterm`. This can be done with:

```
\newtermaddfield[<db list>]{<column key>}[<placeholder cs>]{<new term
key>}[<data type>]{<default value>}
```

For each database listed in the comma-separated list of database labels `<db list>`, this will create a new column with the key `<column key>` (with the given `<data type>` if set, see §2.2). If the `<db list>` argument is missing, all databases that have been defined with `\newgidx` will be iterated over.

The `<placeholder cs>` argument should be a command to use as the placeholder. This will be added to the assignment list `\DTLgidxAssignList`. If `<placeholder cs>` is omitted, it will be formed from the column key. Note that the command must not already be defined.

The `<new term key>` argument is the option name for use in `\newterm` (or `\newacro`) to set the new field, and `<default value>` is the default value to use if the new option isn't provided. Note that this will only affect subsequent instances of `\newterm` or `\newacro`.

Within `<default value>`, you can reference another value with:

```
\field{<key>}
```

The `<key>` argument should be the `\newterm` option name (not the database column key).

For example, suppose I want to be able to specify an alternative plural. I can add a new field like this:

```
\newtermaddfield{AltPlural}{altplural}{}{}
```


8. Creating an index, glossary or list of abbreviations (*datagidx* package)

This adds a new column with the label `AltPlural` to each defined index/glossary database and adds a new key called `altplural` that I can now use in the optional argument of `\newterm`. The default is set to empty. Now I can define terms with an alternative plural:

```
\newterm[altplural=kine]{cow}
```

In the document, I can use `\gls{cow}` to display “cow”, `\glspl{cow}` to display “cows” and `\useentry{cow}{AltPlural}` to display “kine”. To make life a little easier, I can define a new command to save typing:

```
\newcommand*{\glsaltpl}[1]{\useentry{#1}{AltPlural}}
```

Now I can just do `\glsaltpl{cow}` to display “kine”.

Here’s another example. Suppose I want to add a field that produces the past tense of a verb. In this case, the default should be formed by appending “ed” to the `text` value. The new field can be defined as follows:

```
\newtermaddfield{Ed}{ed}{\field{text}ed}
```

This adds a new column labelled `Ed` and defines a new key called `ed` that can be used with `\newterm`. Now I can define some verbs:

```
\newterm{jump}  
\newterm[ed=went]{go}
```

Let’s define a shortcut command to access this field:

```
\newcommand*{\glsed}[1]{\useentry{#1}{Ed}}
```

This new field can now be referenced in the document:

```
He \glsed{jump}  
over the gate.  
She \glsed{go}  
to the shop.
```

He jumped over the gate. She went to the shop.

8.7. Abbreviations

You may have noticed that you can specify `short` and `long` values when you define a new term. There is a convenient shortcut command which uses `\newterm` to define an abbreviation:

`\newacro[⟨options⟩]{⟨short⟩}{⟨long⟩}`

This is a shortcut for:

```
\newterm
[
  description={\capitalisewords{⟨long⟩}},
  short={\acronymfont{⟨short⟩}},
  long={⟨long⟩},
  text={\DTLgidxAcrStyle{⟨long⟩}{\acronymfont{⟨short⟩}}},
  plural={\DTLgidxAcrStyle{⟨long⟩s}{\acronymfont{⟨short⟩s}}},
],
  sort={⟨short⟩},
  ⟨options⟩
]
{⟨SHORT⟩}
```

where `⟨SHORT⟩` is `⟨short⟩` converted to uppercase, and `\capitalisewords` is defined in `mfirstuc` (which is automatically loaded by `datagidx`).

`\acronymfont{⟨text⟩}`

By default this just typesets its argument but can be redefined if the abbreviations need to be typeset in a certain style (such as small caps).

`\DTLgidxAcrStyle{⟨long⟩}{⟨short⟩}`

This governs how the abbreviation is typeset in the `text` value. This defaults to: `⟨long⟩ (⟨short⟩)`.

The plural form is obtained by appending “s” to the `⟨long⟩` and `⟨short⟩` form. If this is inappropriate you will need to set the `plural`, `shortplural` and `longplural` values in the optional argument of `\newacro`.

The `datagidx` package only provides a very basic abbreviation style. For more complex requirements, consider using the `glossaries-extra` package. See [Gallery: Abbreviation Styles²](https://dickimaw-books.com/gallery/index.php?label=sample-abbr-styles) for

²dickimaw-books.com/gallery/index.php?label=sample-abbr-styles

examples.

8.7.1. Using Abbreviations

You can use terms that represent abbreviations via commands such as `\useentry`. For example, if you define the following in the preamble:

```
\newacro{css}{cascading style sheet}
```

then later in the text you can use:

```
\useentry{css}{Short}
```

to access the short form and

```
\useentry{css}{Long}
```

to access the long form. You can also use

```
\useentry{css}{Text}
```

(or `\gls{css}`) to access the full version. However with abbreviations, you may prefer to have only the full form on first use and just the short form on subsequent use. The following commands are provided to do that. The singular form is obtained using:

```
\acr{<label>}
```

This robust command does:

```
\ifentryused{<label>}{\useentry{<label>}{Short}}
{\DTLgidxFormatAcr{<label>}{Long}{Short}}
```

The plural form is obtained using:

```
\acrpl{<label>}
```

8. Creating an index, glossary or list of abbreviations (*datagidx* package)

This robust command does:

```
\ifentryused{⟨label⟩}{\useentry{⟨label⟩}{ShortPlural}}
{\DTLgidxFormatAcr{⟨label⟩}{LongPlural}{ShortPlural}}
```

Note that, unlike the *glossaries* package, `\acr` isn't the same as `\gls`. With *datagidx*, `\gls` always references the `Text` value. There is no “first” field.



Take care when using `\acr` and `\acrpl` with *beamer*. Using overlays can cause problems with first use expansions.

As a general rule, it's not consider appropriate to capitalise the first letter of an abbreviation (especially if it is displayed in small caps) but if you need to you, for example, first use occurs at the start of a sentence, can use the following commands.



```
\Acr{⟨label⟩}
```

This robust command does:

```
\ifentryused{⟨label⟩}{\Useentry{⟨label⟩}{Short}}
{\DTLgidxFormatAcrUC{⟨label⟩}{Long}{Short}}
```



```
\Acrpl{⟨label⟩}
```

This robust command does:

```
\ifentryused{⟨label⟩}{\Useentry{⟨label⟩}{ShortPlural}}
{\DTLgidxFormatAcrUC{⟨label⟩}{LongPlural}{ShortPlural}}
```

The commands used for formatting and indexing the first use form are:



```
\DTLgidxFormatAcr{⟨label⟩}{⟨long-field⟩}{⟨short-field⟩}
```

This does:

```
\DTLgidxAcrStyle
{ \glsdispenry{⟨label⟩}{⟨long-field⟩} }
{ \useentry{⟨label⟩}{⟨short-field⟩} }
```

```
\DTLgidxFormatAcrUC{⟨label⟩}{⟨long-field⟩}{⟨short-field⟩}
```

This does:

```
\DTLgidxAcrStyle  
{ \Glsdispenry{⟨label⟩}{⟨long-field⟩} }  
{ \useentry{⟨label⟩}{⟨short-field⟩} }
```

Note that `\DTLgidxFormatAcrUC` will need to be redefined if `\DTLgidxAcrStyle` is redefined to show the short form first.

8.7.2. Unsetting and Resetting Abbreviations

You can reset a term so it's marked as not used with:

```
\glsreset{⟨label⟩}
```

or you can unset a term so it's marked as used with:

```
\glsunset{⟨label⟩}
```

These commands change the value stored in the `Used` field (see §8.10.2).

You can reset all the terms defined in a given database using:

```
\glsresetall{⟨db-name⟩}
```

or unset all the terms defined in a given database using:

```
\glsunsetall{⟨db-name⟩}
```

where `⟨db-name⟩` is the name of the database as supplied when the database was defined using `\newgidx`.

8.8. Displaying the Index or Glossary

A database created by `\newgidx` can be displayed with:

```
\printterms[⟨options⟩]
```

8. Creating an index, glossary or list of abbreviations (*datagidx* package)

The optional argument is a $\langle key \rangle = \langle value \rangle$ list that may be used to override the default settings (available options are described in §§8.1.2.2 & 8.1.2.3). To allow the hooks to reference the current database, `\printterms` defines:



```
\DTLgidxCurrentdb
```

This will expand to the database's name (as identified by `database`).

The heading and style is obtained from the special `datagidx` catalogue database (see §8.10), but this may be overridden by the options provided to `\printterms`. The style is set with `\datagidxsetstyle`. Note that this means that any redefinitions of style commands made before `\printterms` will be overridden.

The code supplied by the `sort` option is used to sort the database (which should be referenced with `\DTLgidxCurrentdb`). The special markup commands `\DTLgidxName`, `\DTLgidxPlace`, `\DTLgidxSubject`, and `\DTLgidxOffice` are redefined to invert their arguments (see §8.4.1).

Then `\printterms` will then iterate over the database using `\DTLgidxForeachEntry` where the $\langle body \rangle$ argument is simply `\datagidxitem`, which is set by the style.

The placeholder commands set by `\DTLgidxForeachEntry` (which are obtained by expanding `\DTLgidxAssignList`) may be referenced in the `condition` option or the `include-if` or `include-if-fn` definitions to filter rows from the database. However, note that `\DTLgidxForeachEntry` automatically filters out any row that has a non-null `\Parent` regardless of the condition. Other rows will also be automatically filtered if they have null `Location`, `See` and `SeeAlso` fields (that is, the term hasn't been indexed).



The `LetterGroup` column is filled by the default `sort` code, which uses `save-group-key=LetterGroup` to obtain and save the letter group. If the `sort` code is changed to omit this option then `\datagidxcurrentgroup` will be null (see §3.10 for information about null values).

If the `show-groups` (or `showgroups`) setting is on, `\datagidxcurrentgroup` will be expanded at the start of each iteration and, at the end of the iteration (after $\langle body \rangle$), the expanded value will be saved in `\datagidxprevgroup`.

This allows the styles to compare `\datagidxcurrentgroup` with `\datagidxprevgroup` to determine if the letter group has changed. If the values have changed, a letter group heading can be inserted where the group title is obtained with:



```
\DTLgidxGroupHeaderTitle{ $\langle group \rangle$ }
```

This will expand to `\datagidx $\langle group \rangle$ name`, if it exists, or simply $\langle group \rangle$ otherwise.

Note that `\printterms` does:

```
\DTLgidxForeachEntry{\datagidxitem}
```

This will format each entry that both satisfies `condition` (or `include-if` or `include-if-fn`) and has a null `\Parent` according to the current style. It's up to the style to determine whether or not to display the child entries along with their parent entry. (The list of child labels can be obtained by expanding `\Children`).

Finally, `\printterms` does `\datagidxend`, which is again defined by the style and should close any groups that were opened with `\datagidxstart`. If `\twocolumn` was issued by a combination of `columns=2` and `balance=false`, and two-column mode wasn't already in effect before `\printterms`, then one column mode will be restored with:

```
\printtermsrestoreonecolumn
```

This is simply defined to do `\onecolumn`.

8.8.1. Hooks and Associated Commands

The commands described here are independent of the style, that is the style may use them but should not redefine them. They may be redefined explicitly, but some of these commands are redefined by a setting, such as `child` or `post-name`.

```
\DTLgidxSetColumns{<n>}
```

Locally sets the default number of columns in the index/glossary. The `columns` option uses this command. The value must be greater than 0.

```
\DTLgidxNameCase{<text>}
```

This command encapsulates the `\Name` placeholder, and is intended for case change. If redefined, it should take into account that its argument will be a command not the actual text. The `name-case` option redefines `\DTLgidxNameCase` to use the applicable command. The default definition simply expands to its argument (that is, not case-change is applied).

```
\DTLgidxNameFont{<text>}
```

This command encapsulates `\DTLgidxNameCase{\Name}` and is used to apply a font change. The `name-font` option redefines `\DTLgidxNameFont`. The default definition is to use `\textnormal`.

The name, including the case-changing and font-changing commands, will be encapsulated with `\datagidxtarget`, which will create a hyper-target, if supported. After that will be the post-name hook.

`\DTLgidxPostName`

initial: `\`

This command is the post-name hook inserted after the name for entries that have a null `\Parent`. The `post-name` option redefines `\DTLgidxPostName`.

`\DTLgidxPostChildName`

initial: `\DTLgidxPostName`

This command is the post-name hook inserted after the name for child entries (that is, entries that have a non-null `\Parent`). The default definition is simply `\DTLgidxPostName`.

`\DTLgidxChildStyle{<name>}`

Encapsulates the child's name (including `\DTLgidxNameFont` and `\DTLgidxName-Case`) and post-name hook (`\DTLgidxPostChildName`). The default definition simply expands to its argument. The `child=noname` setting redefines `\DTLgidxChildStyle` to ignore its argument and expand to `\DTLgidxChildCountLabel`. The `child=noname` setting redefines `\DTLgidxChildStyle` to expand to its argument (that is, back to its original definition).

`\DTLgidxChildCountLabel`

Expands to `\theDTLgidxChildCount)` (that is, the value of the child counter, followed by a closing parenthesis and space).

The symbol and description (if not null and not empty) may be shown, depending on the `symbol-desc` setting.

`\DTLgidxSymDescSep`

initial: `\space`

Separator to place between the symbol and description if both are present.

`\DTLgidxFormatDesc{<text>}`

Encapsulates the description, if applicable. Note that the argument will be the `\Description` placeholder.

`\DTLgidxPostDescription`

initial: `empty`

Inserted after the description, if applicable.

`\DTLgidxEndInitItem`

Inserted at the end of each item (after the location list, child list and cross-references, if applicable). The default definition is `\par \smallskip` (which creates a paragraph break and a small vertical gap).

The location list may or may not be shown, depending on the `location` setting. The `location=hide` setting will omit the location list and the pre and post location hooks.

`\DTLgidxPreLocation`

initial: `\enspace`

Inserted before the location, if applicable. The `pre-location` setting redefines this command.

`\DTLgidxPostLocation`

initial: `\quad`

Inserted after the location, if applicable.

`\DTLgidxFormatSee{<tag>}{<label list>}`

Used to format the “see” cross-reference list. The `<tag>` is the textual prefix, and the `<label-list>` argument is the comma-separated list of labels, obtained from expanding the `\See` placeholder.

`\DTLgidxFormatSeeAlso{<tag>}{<label list>}`

Used to format the “see also” cross-reference list. This has a similar definition to `\DTLgidxFormatSee` but a pre hook before the tag and a post hook after the list, which are determined by the style.

`\DTLgidxSeeTagFont{<text>}`

Encapsulates the `<tag>` part in the definition of `\DTLgidxFormatSee` and `\DTLgidxFormatSeeAlso`. By default, this uses `\emph`.

Each item label in the cross-reference list is encapsulated with:

`\DTLidxFormatSeeItem{<label>}`

This fetches the corresponding name from the database and, if supported, creates a hyperlink to the referenced label.

`\DTLidxSeeLastSep`

initial: `\&`

Inserted between the final two items in the cross-reference list.

`\DTLidxSeeSep`

initial: `,`

Inserted between all except the final two items in the cross-reference list.

`\seename`

Used for the tag in `\DTLgidxFormatSee`, this command is defined by language packages, such as `babel`, but will be provided by `datagidx` if not already defined.

`\seealso`

Used for “see also” references, if not already defined it will be defined to `\also`, if that command exists, or to “see also” otherwise.

`\datagidxsymbolwidth`

initial: `0pt`

This dimension is set with the `symbol-width` option, but may also be changed with `\setlength`. If zero or negative, the symbol will just occupy its natural space, otherwise the styles that support this setting will allocate this width for the location list.

`\datagidxlocationwidth`

initial: `0pt`

This dimension is set with the `location-width` option, but may also be changed with `\setlength`. If zero or negative, the location list will just occupy its natural space, otherwise the styles that support this setting will allocate this width for the location list.

`\datagidxsymalign`

initial: `\centering`

Alignment of the symbol if the symbol width dimension has been set to a positive value.

`\datagidxlocalign`

initial: `\raggedleft`

Alignment of the location list if the location width dimension has been set to a positive value.

8.8.2. Index or Glossary Styles

The style used by `\printterms` is specified with the `style` option. The default is `style=index`.

8.8.2.1. index

The default `index` style is a basic style for indexes, but the symbol and description will be shown if set, and it will follow the `symbol-width` and `location-width` settings.

8.8.2.2. indexalign

The `indexalign` style is similar to the `index` style but aligns the descriptions. This requires an initial iteration over the database to calculate the alignment.

8.8.2.3. align

The `align` style aligns the fields. It will follow the `symbol-width` and `location-width` settings, but will require an initial iteration over the database to calculate the alignment.

8.8.2.4. gloss

The `gloss` style is a basic style for glossaries.

<code>\DTLgidxChildSep</code>	<i>initial:</i> <code>□</code>

Separator between child entries.

<code>\DTLgidxPostChild</code>	<i>initial:</i> <code>empty</code>

Hook inserted at the end of the list of child entries.

8.8.2.5. dict

The `dict` style is a dictionary style, for a hierarchical structure where the top level entries have a name. The next level is used to indicate a category, such as “adjective” or “noun”. If there is only one meaning this level also has a description. If there is more than one meaning, each meaning should be a child of the category entry. Only third level entries are numbered. The `Child` column is ignored by this style. The symbol is ignored. The location and symbol widths are also ignored.

If `show-groups=true`, the group headers will be inserted with `\DTLgidxDict-Head`.

`\DTLgidxDictPostItem`

initial: `\DTLgidxEndItem`

Inserted at the end of each item.

`\datagidxdictindent`

initial: `1em`

Indentation used by the `dict` style. Note that this isn't a register, so it should be changed with `\renewcommand`.

`\DTLgidxDictHead`

Inserts the group header. This will use `\chapter` if it has been defined or `\section` otherwise. The title is obtained with `\DTLgidxGroupHeaderTitle`.

`\DTLgidxCategoryNameFont{<text>}`

Encapsulates the “category” child names. Just expands to its argument by default.

`\DTLgidxCategorySep`

initial: `\space`

Separator used between the category entries.

`\DTLgidxSubCategorySep`

initial: `\space`

Separator used between child entries whose parent entry is a category.

8.8.3. Sorting the Index or Glossary Database

When displaying the list, `\printterms` will automatically sort the database according to the `sort` option. The default code is:

```
\DTLsortdata
[save-group-key=LetterGroup]
{\DTLgidxCurrentdb}% current database
{HierSort={replacements=Sort},FirstId}
```

This sorts by the `HierSort` column first. If the value isn't set, the value from the `Sort` column will be used. In the event that two rows being compared have identical sort values, the `FirstId` column will be compared. Note that `\DTLgidxCurrentdb` is used to identify the database.



Remember that `\DTLsortdata` is influenced by the available localisation support.

Prior to version 3.0, the default `sort` code was:

```
\dtlsort{Sort,FirstId}{\DTLgidxCurrentdb}{\dtlword-
indexcompare}
```

Note that this is less efficient than using the newer `\DTLsortdata` and there was no `HierSort` column. However, if you want to revert back to this behaviour without using `rollback`, you can do:



```
\printterms[sort=
\dtlsort{Sort,FirstId}{\DTLgidxCurrentdb}{\dtlword-
indexcompare}
]
```

If you don't want the database sorted, just set `sort` to an empty value:



```
\printterms[sort=]
```

8.8.3.1. Optimization

Version 3.0 has improved sorting times by switching to the more efficient `l3seq` sorting function. However, for a large database, you may still want to switch on the optimization settings.

If you have used indexing applications, such as `makeindex`, you'll be familiar with the document creation process. The document is first compiled, then the indexing application is run to sort and collate the entries, then the document is compiled again (and possibly once more). This involves two (or three) runs and one sort and collate run (with the process possibly repeated in the event of complex documents with shifting page numbers). With the *datagidx* package, the sorting and collation is done every $\LaTeX$ run. For a large index, this can be quite slow. If you're not editing the index or glossary, you might prefer not to have to keep sorting the database whenever you update the document. To assist this, *datagidx* provides the `optimize` package option. This may take the following values:



```
optimize=off
```

Don't use the `optimize` facility. The index/glossary databases will be sorted every run, unless the sorting is switched off by setting the `sort=key` to empty.



```
optimize=low
```

Use the “low” optimize setting. This only sorts the index/glossary databases on every run. This is assuming that the sorting is done via the `\printterms sort=key`, rather than by explicitly sorting the database with commands like `\DTLsortdata` somewhere else in the document.



Don’t use the `optimize=low` setting if sorting the databases makes the document out of date. For example, the group headers use sectioning commands.



```
optimize=high
```

Use the “high” optimize setting. This sorts the index/glossary databases on the first run, then writes the sorted databases to external files, which are read in on subsequent runs. Again this assumes that sorting is done via the `\printterms sort=key`. Don’t use this option if you want to edit the index/glossary database.

8.9. Supplementary Commands



```
\DTLgidxEnableHyper
```

Enables hyperlinks, if supported. This redefines `\datagidxtarget` and `\datagidxlink` back to their default behaviour, which tests for the existence of `\hypertarget` and `\hyperlink`.



```
\DTLgidxDisableHyper
```

Disables hyperlinks. This redefines `\datagidxtarget` and `\datagidxlink` to simply expand to their `<text>` argument.

8.9.1. Conditionals and Loops



```
\iftermexists[<label>]{<true>}{<false>}
```

Expands to `<true>` if a term has been defined with the given label, otherwise expands to `<false>`.

```
\ifentryused[⟨label⟩]{⟨true⟩}{⟨false⟩}
```

This is a robust command that does *⟨true⟩* if term identified by the given *⟨label⟩* has been marked as used. (That is, the value for the *Used* column for the associated row is set to 1.)

```
\DTLgidxForeachEntry{⟨body⟩}
```

Iterates over the current database (obtained by expanding *\DTLgidxCurrentdb*) using *\DTLmapdata* with *read-only=true*. This command is primarily provided for use within *\printterms*. If you want to use it explicitly, make sure that *\DTLgidxCurrentdb* expands to the required database name.

The *⟨body⟩* argument is done for each row where all of the following apply:

- the *Parent* field is null;
- the *Location* or *See* or *SeeAlso* fields is non-null and non-empty;
- the entry has been marked as used;
- the condition evaluated by the current *condition* or *include-if* or *include-if-fn* setting is true.

If *⟨body⟩* is done, the value of *\Label* will be added to a global list of referenced labels, which is used at the end of the document to check if a rerun is required.

After *⟨body⟩* is done, the current letter group information is saved in:

```
\datagidxprevgroup
```

This allows the styles to compare *\datagidxcurrentgroup* with *\datagidxprevgroup* to determine if the letter group has changed.

The placeholder commands are assigned by expanding:

```
\DTLgidxAssignList
```

The default expansion is:

```
\Name=Name,
\Description=Description,
\Used=Used,
\Symbol=Symbol,
\Long=Long,
\Short=Short,
\LongPlural=LongPlural,
```

```

\ShortPlural=ShortPlural,
\Location=Location,
\See=See,
\SeeAlso=SeeAlso,
\Text=Text,
\Plural=Plural,
\CurrentLocation=CurrentLocation,
\Label=Label,
\Parent=Parent,
\Children=Child,
\FirstId=FirstId,
\HierSort=HierSort,
\Sort=Sort,
\datagidxcurrentgroup=LetterGroup

```

If new fields are added with `\newtermaddfield`, the supplied placeholder command will be added to the assignment list. If you manually add columns, using commands like `\DTLaddcolumn`, you can append your own custom placeholders to this list if they need to be referenced.



If you redefine `\DTLgidxAssignList` so that it omits the essential placeholder commands referenced by `\printterms` and its underlying hooks, errors will occur.



```
\datagidxmapdata{<body>}
```

This command is similar to `\DTLgidxForeachEntry` but is provided for use by the styles to iterate over all entries in the current database (typically to calculate the width of certain field values). Unlike `\DTLgidxForeachEntry`, the letter group commands aren't updated and the list of referenced labels isn't updated.

8.9.2. New Terms

There is a hook implemented at the end of `\newterm`:



```
\postnewtermhook
```

This expands to nothing by default. Just before this hook, `\newterm` will define:



```
\datagidxlastlabel
```


8. Creating an index, glossary or list of abbreviations (*datagidx* package)

to expand to the term's label. This may be used to reference the term within the hook.

When `\newterm` automatically generates the `label` and `sort` values (if omitted, see §§8.4.2 & 8.4.3) certain commands are locally set to:

```
\DTLgidxNoFormat{<text>}
```

This simply expands to its argument by default (equivalent to `\@firstofone`).

```
\DTLgidxGobble{<text>}
```

This simply expands to nothing (equivalent to `\@gobble`).

Certain commands, such as `\&`, are converted with:

```
\datagidxconvertchars
```

This is largely redundant with the new $\text{\LaTeX}$ 3 commands and `datatool-base`'s localisation support, but is retained for backward-compatibility. There is a similar command for converting commands such as `\alpha`:

```
\datagidxwordifygreek
```

8.9.3. Styles

```
\datagidxsetstyle{<style-name>}
```

This is implemented by `\printterms` to set the current style, which means that any redefinitions of the style commands listed below will be overridden. If you want to make any adjustments to these commands, you will need to define a new style.

```
\datagidxnewstyle{<style-name>}{<definitions>}
```

This defines a new style called *<style-name>*. The *<definitions>* argument may start with `\datagidxsetstyle{<other-style>}` if only minor modifications to an existing style (*<other-style>*) are required.

In order to support hyperlinks, the entry name (including the case-changing and font changing commands) should be placed in the second argument of:

`\datagidxtarget{<target-name>}{<text>}`

The first argument will be the `\Label` placeholder. This will generate a hypertarget with the label as the target name, if supported, and display `<text>`.

`\datagidxlink{<target-name>}{<text>}`

If hyperlinks are support, creates a hyperlink with `<text>` as the link text, otherwise just does `<text>`.

If the labels used are likely to cause a conflict with other hyperlinks in the document, redefine `\datagidxtarget` and `\datagidxlink` to insert a prefix.

DTLgidxChildCount

This counter should typically not be explicitly changed. The provided styles reset the `DTLgidxChildCount` counter to 0 at the start of each child list and increment it for each child entry. The `child=noname` setting redefines `\DTLgidxChildStyle` to use `\DTLgidxChildCountLabel`, ignoring its argument.

8.10. Database Structures

There are two types of databases recognised by *datagidx*: the catalogue database, and the databases used to store the terms.

8.10.1. The Catalogue Database

The *datagidx* package automatically creates a database called *datagidx*. This database keeps track of all the databases that are created by `\newgidx` along with their particular settings. The *datagidx* database has the following column keys:

- **Glossary:** this column stores the label of the database created with *datagidx* (the `<db-name>` argument of `\newgidx`);
- **Title:** this column stores the title to be used by `\printterms` (the `<title>` argument of `\newgidx`);
- **Heading:** this column stores the heading to be used by `\printterms` (which can be set with the `heading` option);

8. Creating an index, glossary or list of abbreviations (*datagidx* package)

- `PostHeading`: this column stores the post-heading content to be used by `\printterms` (which can be set with the `post-heading` option);
- `MultiCols`: this column stores the environment name to use by `\printterms` for multiple columns (`balance=true` sets this value to `multicols*` and `balance=false` sets this value to `multicols`);
- `Sort`: this column stores the code that `\printterms` should use to sort the data before displaying it (which can be set with the `sort` option);
- `Style`: this column stores the style name that `\printterms` should use to displaying the data (which can be set with the `style` option);
- `ShowGroups`: this column stores the `show-groups` setting (that is, `true` or `false`).

8.10.2. The Term Databases

Each database created with `\newgidx` contains each defined term or abbreviation associated with that database. The column keys used to store the information are listed below. The placeholder commands used in the column value assignments in the expansion text of the default definition of `\DTLgidxAssignList` are shown in parentheses.

- `Label (\Label)`: the unique label that identifies the entry;
- `Text (\Text)`: the entry's text for use by commands like `\gls`;
- `Plural (\Plural)`: the entry's plural text for use by commands like `\glspl`;
- `Symbol (\Symbol)`: the entry's symbol for use by commands like `\glsym`, if applicable;
- `Short (\Short)`: the entry's short (abbreviated) form for use by commands like `\acr`, if applicable;
- `ShortPlural (\ShortPlural)`: the entry's short plural form for use by commands like `\acrpl`, if applicable;
- `Long (\Long)`: the entry's long form for use by commands like `\acr`, if applicable;
- `LongPlural (\LongPlural)`: the entry's long plural form for use by commands like `\acrpl`, if applicable;

The following columns are primarily intended for use in `\printterms`:

- `Name (\Name)`: the entry's name, as it should appear in `\printterms`;
- `Description (\Description)`: the entry's description, as it should appear in `\printterms`, if applicable;

8. Creating an index, glossary or list of abbreviations (*datagidx* package)

- `See (\See)`: the entry’s “see” cross-reference, if applicable;
- `SeeAlso (\SeeAlso)`: the entry’s “see also” cross-reference, if applicable;
- `Parent (\Parent)`: the entry’s parent label, if applicable;
- `Child (\Children)`: a comma-separated list of child labels, which is updated whenever a new entry is defined identifying this entry as its parent;
- `Location (\Location)`: the location list obtained from the previous run;
- `LetterGroup (\datagidxcurrentgroup)`: the entry’s letter group, as obtained by `\DTLsortdata` with the default `sort` setting.

The following are considered internal fields, but may be referenced in the sorting or filtering code.

- `Sort (\Sort)`: the entry’s sort value (which normally defaults to the name but may be explicitly set with the `sort` option);
- `HierSort (\HierSort)`: the entry’s hierarchical sort value, if applicable, which is obtained from the sort value appended the parent’s hierarchical sort value;
- `FirstId (\FirstId)`: a numeric value corresponding to the first reference (indexing) of the term, so sorting by this column will normally result in an order of use listing;
- `Used (\Used)`: a column that will contain either 0 (unused) or 1 (used) marked up as a special value (see §3.11) to indicate whether or not the entry has been used in the current run.

The following are considered worker fields that will typically not be needed in hooks or styles.

- `CurrentLocation (\CurrentLocation)`: the location list obtained from the current run, which may be one off due to the delayed output routine;
- `UnsafeLocation`: the value of the `CurrentLocation` from the previous run, which may be one off due to the delayed output routine (the current and previous unsafe locations are compared to determine if the locations have changed). This field isn’t included in `\DTLgidxAssignList` as it’s only used to check if the location list has changed from the previous run to determine whether or not to issue a rerun warning.

8.11. Examples

Example 191 uses the `datagidx` package to create an index. Note that I’ve specified English as the locale. This means that I also need to have `datatool–english` installed.

191

```
\usepackage[locales=en]{datagidx}
```

The `hyperref` package is also used:

```
\usepackage[colorlinks]{hyperref}
```

The database is created and populated in the preamble:

```
\newgidx{index}
Index% define a database for the index
\DTLgidxSetDefaultDB{index}% set this as the default
\newterm{macédoine}
\newterm{macramé}
\newterm{élite}
\newterm{reptile}
\newterm[seealso=reptile]crocodylian
\newterm[parent=crocodylian]crocodile
\newterm[parent=crocodylian]alligator
\newterm[
  parent=crocodylian,
  description={ (also cayman) }
]
caiman
\newterm[see=caiman]cayman
```

The document text consists of:

```
Here are some words containing accents: \gls
{macédoine},
\gls{macramé} and \gls{élite}. \Gls{élite}
starts with an uppercase
letter. A \gls{crocodylian} is the family of
\glspl{reptile} that includes \glspl{crocodile}
, \glspl{alligator}
and \glspl{caiman}.
```

The index is displayed with:

```
\printterms
[
  heading={\section*},
  database=index,
```

```
prelocation=dotfill,
showgroups
]
```

This requires two $\text{\LaTeX}$ runs to ensure that the index is up to date.

↑ Example 191: Creating an Index

Here are some words containing accents: **macédoine**, **macramé** and **élite**. **Élite** starts with an uppercase letter. A **crocydlian** is the family of **reptiles** that includes **crocodiles**, **alligators** and **caimans**.

Index

C		M	
cayman	see caiman	macédoine	1
crocydlian	1	macramé	1
alligator	1		
caiman (also cayman).....	1		
crocodile	1		
E		R	
élite	1	reptile	1

Example 192 uses the *datagidx* package to create a list of abbreviations. The *hyperref* package is also used:

192

```
\usepackage{datagidx}
\usepackage[colorlinks]{hyperref}
```

The database is created and populated in the preamble:

```
\newgidx{abbreviations}Abbreviations
\DTLgidxSetDefaultDB{abbreviations}
\newacro{html}{hyper-text markup language}
\newacro{css}{cascading style sheet}
```

The following overrides the default description:

```
\newacro[description={eXtensible Markup Language}]
{xml}{extensible markup language}
```

The document text is:

First use: `\acr{xml}` and `\acr{css}`.
 Next use: `\acr{xml}` and `\acr{css}`.
 Full form: `\gls{xml}` and `\gls{css}`.

The list of abbreviations is displayed with:

```
\printterms
[
  postdesc=
dot,% put a full stop after the description
columns=1,% one column page layout
namefont={\textbf}
,% put the name (i.e. the abbreviation) in bold
namecase=uc,% make the name upper case
style=align% use the 'align' style
]
```

↑ Example 192: Creating a List of Abbreviations

First use: extensible markup language (**xml**) and cascading style sheet (**css**).

Next use: **xml** and **css**.

Full form: **extensible markup language (xml)** and **cascading style sheet (css)**.

Abbreviations

CSS Cascading Style Sheet. **1**

XML eXtensible Markup Language. **1**

9. Referencing People (person package)



```
\usepackage[options]{person}
```

The `person` package can be used to define (§9.3) and reference people (§9.5). This package automatically loads the `datatool` package, since it's primarily intended for use with databases (for example, for mail merging). However, you can simply define an individual or a list of people without needing a database. If you don't require `datatool`, use the `base-only` package option, which will only load `datatool-base` rather than `datatool`. Examples are provided in §9.6.

The `person` package was rewritten in version 3.0 to use L^AT_EX3 commands. The internal list of labels is now stored as a sequence variable, which means that loop commands, such as `\foreachperson`, no longer use `\@for`. This will only make a difference if you have used the `xfor` commands to break the loop.



Some commands that have the potential to conflict with commands from other packages have been replaced. In most cases, the deprecated command will be provided in v3.0 (if not already defined) with a warning, but may be removed in future. Two commands have been completely removed: `\malelabels` and `\femalelabels`. They have been replaced with comma-separated list variables. Use `\PersonSetMaleLabels` and `\PersonSetFemaleLabels` to replace the lists if required.

If there are any backward-compatibility issues, rollback to version 2.32 is available:



```
\usepackage{person}[=2.32]
```

Note that if `datatool` hasn't already been loaded, this will also apply rollback to `datatool`. Problems may occur if a newer release of `datatool` has already been loaded.

The `person` package now has the provision for tracklang localisation support, using the same localisation settings as for the underlying `datatool-base` package. The `datatool-english` package (distributed separately) includes the file `person-english.ldf` which provides the English localisation support for the `person` package. This simply needs to be installed on T_EX's path and will be loaded automatically if required (see §9.7.3).

Any options recognised by `datatool-base` may also be passed to the `person` package. Similarly for `datatool` if that should also be loaded. For example:


```
\usepackage[locales=en-GB]{person}
```

9.1. Package Options

The following options may be passed to the person package when it loads. They can't be used in `\DTLsetup`.

base-only

Only load `datatool-base`, not `datatool` (if not already loaded). You may additionally pass any options to `person` that are recognised by `datatool-base`.

datatool

Load `datatool` (unless it has already been loaded). You may additionally pass any options to `person` that are recognised by `datatool`. This is the default.

shortcuts

Defines the shortcut commands listed in Table 9.1. If not passed as a package option, this option may also be used within the `person` option of `\DTLsetup`:

```
\DTLsetup{person={shortcuts}}
```

9.2. Other Options

These options may be used within the `person` option of `\DTLsetup`. For example:

```
\DTLsetup{person={global}}
```

local=*<boolean>*

default: true; initial: true

A boolean option that determines whether commands such as `\newperson` have a local effect.

global= $\langle$ boolean $\rangle$ default: **true**; initial: **false**

Antonym of `local`. That is, `global=true` is equivalent to `local=false`.

shortcuts

Defines the shortcut commands listed in Table 9.1. If this option or the `shortcuts` package option has already used, this does nothing.

9.3. Defining and Undefining People

The effect of these commands may be scoped `local=true`. Otherwise they will be global.

 $\backslash$ newperson [$\langle$ person-label $\rangle$] { $\langle$ full name $\rangle$ } { $\langle$ name $\rangle$ } { $\langle$ gender-label $\rangle$ }

Defines a new person with the given full name ($\langle$ full name $\rangle$) and familiar or first name ($\langle$ name $\rangle$) who can be identified with the given label $\langle$ person-label $\rangle$. The mandatory arguments will all be fully expanded and trimmed, so this is equivalent to:

```
 $\backslash$ newperson* [ $\langle$ person-label $\rangle$ ] {
  expand-fullname={  $\langle$ full name $\rangle$  } ,
  expand-name={  $\langle$ name $\rangle$  } ,
  gender={  $\langle$ gender-label $\rangle$  }
}
```

See the descriptions of the `fullname`, `name` and `gender` options for further details.

If the $\langle$ person-label $\rangle$ optional argument is omitted, `anon` will be used. This is the default label for commands such as $\backslash$ personname so with the `local` setting on, this is useful in a scoped context to create a temporary person without the need to assign a new label.

 $\backslash$ newperson* [$\langle$ person-label $\rangle$] { $\langle$ key=value list $\rangle$ }

The starred version is new to version 3.0 and has a $\langle$ key $\rangle$ = $\langle$ value $\rangle$ list argument, which makes it more flexible and allows for additional information to be supplied. The optional argument $\langle$ person-label $\rangle$ is as for the unstarred $\backslash$ newperson. There are no required settings. If all are omitted, then the person will have an empty name and unknown gender.

Available options:

fullname= { $\langle$ full name $\rangle$ }initial: *empty*

The person's full name. If omitted or empty, the full name will be constructed as follows:

9. Referencing People (person package)

- if `forenames` is set then, if the `surname` is also set, the full name will be set to the forenames followed by a space and then the surname, otherwise the full name will be just the forenames;
- if `name` is set then, if the `surname` is also set, the full name will be set to the name followed by a space and then the surname, otherwise the full name will be just the name;
- if the `surname` is set then, if the `title` is also set, the full name will be the title followed by `\persontitlesurnamesep` and then the title, otherwise the full name will be set to just the surname.

expand-fullname= { *⟨full name⟩* }

initial: empty

As `fullname` but fully expands the value.

expand-once-fullname= { *⟨full name⟩* }

initial: empty

As `fullname` but expands the first token once.

name= { *⟨name⟩* }

initial: empty

The person's familiar or first name.

expand-name= { *⟨name⟩* }

initial: empty

As `name` but fully expands the value.

expand-once-name= { *⟨name⟩* }

initial: empty

As `name` but expands the first token once.

forenames= { *⟨forenames⟩* }

initial: empty

The person's forenames. If omitted or empty, this will be set to the `name`.

expand-forenames= { *⟨forenames⟩* }

initial: empty

As `forenames` but fully expands the value.

expand-once-forenames= { *⟨forenames⟩* }

initial: empty

As `forenames` but expands the first token once.

surname= { *⟨surname⟩* }

initial: empty

The person's surname.

expand-surname= { *⟨surname⟩* }

initial: empty

As `surname` but fully expands the value.

expand-once-surname= { *⟨surname⟩* }

initial: empty

As `surname` but expands the first token once.

title= { *⟨title⟩* }

initial: empty

The person's title (form of address).

expand-title= { *⟨title⟩* }

initial: empty

As `title` but fully expands the value.

expand-once-title= { *⟨title⟩* }

initial: empty

As `title` but expands the first token once.

gender= { *⟨gender-label⟩* }

initial: empty

The person's gender. The value *⟨gender-label⟩* will first be tested for null (see §3.10) using `\datatool_if_null_or_empty:nTF`. If it's null, `unknown` will be assumed, otherwise *⟨gender-label⟩* will be fully expanded and tested against the recognised gender labels (see §9.4). If *⟨gender-label⟩* is omitted or expands to empty, `unknown` will be assumed.

`\PersonTotalCount`

Expands to the total number of defined people.

`\removeperson[⟨person-label⟩]`

Removes the person identified by *⟨person-label⟩* from the list of known people and undefines the underlying commands.

```
\removepeople{<list>}
```

Removes each person identified by their label in the given comma-separated list.

```
\removeallpeople
```

Removes all defined people.

9.4. Genders

If you have localisation support for the person package, be aware that changing the language may result in resetting the gender labels back to the default for that language.

When defining a new person, you can identify the person's gender using a label. There are four categories:

Male

The internal label used to identify the male gender is `male`. This label, which is used in the construction of underlying language-sensitive commands, may always be used in `\newperson` or for the `gender` option in `\newperson*` (regardless of whether or not it is included in the allowed list of alternative male labels).

Alternative male labels, which may be used to reference the gender in `\newperson`, may be provided as a comma-separated list in the argument of:

```
\PersonSetMaleLabels{<label list>}
```

The default list is: `Male`, `MALE`, `M` and `m`. You can append to this list using: `\PersonAddMaleLabel`. This will expand the label before adding it to the internal list. Both commands have a global effect, regardless of the `local` setting.

The list may be set by a localisation file or you can set or append to it before you define new people.

Command names have changed in version 3.0 to reduce conflict with other packages. The older command `\addmalelabel` is now deprecated and will do `\PersonAddMaleLabel`. The older command `\malelabels` which stored the list has been replaced with `\g_person_male_label_clist`. If you have a pre-v3.0 document that redefined `\malelabels`, replace the

redefinition with `\PersonSetMaleLabels`.

Female

The internal label used to identify the female gender is `female`. This label, which is used in the construction of underlying language-sensitive commands, may always be used in `\newperson` or for the `gender` option in `\newperson*` (regardless of whether or not it is included in the allowed list of alternative female labels).

Alternative female labels, which may be used to reference the gender in `\newperson`, may be provided as a comma-separated list in the argument of:

`\PersonSetFemaleLabels{<label list>}`

The default list is: `Female`, `FEMALE`, `F` and `f`. You can append to this list using: `\PersonAddFemaleLabel` This will expand the label before adding it to the internal list. Both commands have a global effect, regardless of the `local` setting.

The list may be set by a localisation file or you can set or append to it before you define new people.

Command names have changed in version 3.0 to reduce conflict with other packages. The older command `\addfemalelabel` is now deprecated and will do `\PersonAddFemaleLabel`. The older command `\femalelabels` which stored the list has been replaced with `\g_person_female_label_clist`. If you have a pre-v3.0 document that redefined `\femalelabels`, replace the redefinition with `\PersonSetFemaleLabels`.

Non-Binary

The internal label used to identify non-binary is `nonbinary`. This label, which is used in the construction of underlying language-sensitive commands, may always be used in `\newperson` or for the `gender` option in `\newperson*` (regardless of whether or not it is included in the allowed list of alternative non-binary labels).

Alternative non-binary labels, which may be used to reference the gender in `\newperson`, may be provided as a comma-separated list in the argument of:

`\PersonSetNonBinaryLabels{<label list>}`

The default list is: `non-binary`, `Nonbinary`, `Non-Binary`, `NONBINARY`, `N` and `n`. You can append to this list using: `\PersonAddNonBinaryLabel` This

will expand the label before adding it to the internal list. Both commands have a global effect, regardless of the `local` setting.

The list may be set by a localisation file or you can set or append to it before you define new people.

Unknown

The internal label used to identify an unknown gender is `unknown`. This label is used in the construction of language-sensitive commands. When defining a person with `\newperson`, the gender label may be set to `unknown` or empty. There are no alternative labels as this category is provided for instances where the information isn't provided.

If you want to test if a label is a recognised gender label, you can use the following commands. The `<gender-label>` argument will have the first token expanded once. The expansion will first be compared against the internal label and then tested if it's in the relevant comma-separated list. These commands are robust.

`\PersonIfMaleLabel{<gender-label>}{<true>}{<false>}`

Does `<true>` if the one-level expansion of `<gender-label>` is recognised as a male gender identifier. (This replaces the older, deprecated `\ifmalelabel`.)

`\PersonIfFemaleLabel{<gender-label>}{<true>}{<false>}`

Does `<true>` if the one-level expansion of `<gender-label>` is recognised as a female gender identifier. (This replaces the older, deprecated `\iffemalelabel`.)

`\PersonIfNonBinaryLabel{<gender-label>}{<true>}{<false>}`

Does `<true>` if the one-level expansion of `<gender-label>` is recognised as a non-binary gender identifier. Note that this does not include “unknown”.

`\PersonMaleCount`

Expands to the total number of people defined with the gender set to male.

`\PersonFemaleCount`

Expands to the total number of people defined with the gender set to female.

`\PersonNonBinaryCount`

Expands to the total number of people defined with the gender set to non-binary.



```
\PersonUnknownGenderCount
```

Expands to the total number of people defined with the gender set to unknown.

9.5. Displaying Information

The commands below that start with `\person` show the applicable language-sensitive text according to the gender of the person identified by the label. If the label is omitted, `anon` is assumed. The commands that start with `\Person` are the sentence case equivalents.

The commands that start with `\people` show the plural form according to the gender of the group of all defined people. If all people were defined with the same gender, then the plural form for that gender is used, otherwise the plural form for the unknown gender will be used. If only one person has been defined, then resulting text will be equivalent to the analogous `\person` command. The commands that start with `\People` are the sentence case equivalents.

The text produced by these commands can be changed with `\PersonSetLocalisation`. For example:



```
\PersonSetLocalisation{unknown}{pronoun2}{thou}
\PersonSetLocalisation{unknown}{objpronoun2}{thee}
\PersonSetLocalisation{unknown}{possadj2}{thy}
\PersonSetLocalisation{unknown}{posspronoun2}{thine}
```

9.5.0.1. Grammar

Some of the commands listed here are a bit cumbersome. The `shortcuts` option will define shorter synonyms for some of the commands. These are listed in Table 9.1.



```
\personpronoun[<person-label>]
```

Displays the third person singular subjective pronoun according to the gender of the person identified by the label. The subjective pronoun is the pronoun used as the subject of the sentence. For example, “she sees the duck” starts with the female third person singular subject pronoun:



```
\newperson*{name=Zoë,gender=female}
\personpronoun\ sees the duck
```



```
\Personpronoun[<person-label>]
```


Table 9.1.: Synonyms provided by the `shortcuts` package option

Shortcut Command	Equivalent Command
<code>\they</code>	<code>\peoplepronoun</code>
<code>\They</code>	<code>\Peoplepronoun</code>
<code>\them</code>	<code>\peopleobjpronoun</code>
<code>\Them</code>	<code>\Peopleobjpronoun</code>
<code>\their</code>	<code>\peoplepossadj</code>
<code>\Their</code>	<code>\Peoplepossadj</code>
<code>\theirs</code>	<code>\peopleposspronoun</code>
<code>\Theirs</code>	<code>\Peopleposspronoun</code>
<code>\you</code>	<code>\peoplepronounii</code>
<code>\You</code>	<code>\Peoplepronounii</code>
<code>\thee</code>	<code>\peopleobjpronounii</code>
<code>\Thee</code>	<code>\Peopleobjpronounii</code>
<code>\your</code>	<code>\peoplepossadjii</code>
<code>\Your</code>	<code>\Peoplepossadjii</code>
<code>\yours</code>	<code>\peopleposspronounii</code>
<code>\Yours</code>	<code>\Peopleposspronounii</code>
<code>\children</code>	<code>\peoplechild</code>
<code>\Children</code>	<code>\Peoplechild</code>
<code>\parents</code>	<code>\peopleparent</code>
<code>\Parents</code>	<code>\Peopleparent</code>
<code>\siblings</code>	<code>\peoplesibling</code>
<code>\Siblings</code>	<code>\Peoplesibling</code>

As `\personpronoun` but starts with a capital.

`\peoplepronoun`

Displays the third person plural subjective pronoun according to the gender of all defined people. If only one person has been defined, this produces the same text as `\personpronoun`. With the `shortcuts` option, you can use `\they` instead.

`\Peoplepronoun`

As `\peoplepronoun` but starts with a capital. With the `shortcuts` option, you can use `\They` instead.

`\personpronounii[⟨person-label⟩]`

Displays the second person singular subjective pronoun according to the gender of the person identified by the label. In English, this is “you” regardless of gender.

`\Personpronounii[⟨person-label⟩]`

As `\personpronounii` but starts with a capital.

`\peoplepronounii`

Displays the second person plural subjective pronoun according to the gender of all defined people. If only one person has been defined, this produces the same text as `\personpronounii`. With the `shortcuts` option, you can use `\you` instead.

`\Peoplepronounii`

As `\peoplepronounii` but starts with a capital. With the `shortcuts` option, you can use `\You` instead.

`\personobjpronoun[⟨person-label⟩]`

Displays the third person singular objective pronoun according to the gender of the person identified by the label. The objective pronoun is the pronoun used as the object of the sentence. For example, “the duck sees her” ends with the female third person singular objective pronoun:

```
\newperson*{name=Zoë, gender=female}
the duck sees \personobjpronoun
```

```
\Personobjpronoun[⟨person-label⟩]
```

As `\personobjpronoun` but starts with a capital.

```
\peopleobjpronoun
```

Displays the third person plural objective pronoun according to the gender of all defined people. If only one person has been defined, this produces the same text as `\personobjpronoun`. With the `shortcuts` option, you can use `\them` instead.

```
\Peopleobjpronoun
```

As `\peopleobjpronoun` but starts with a capital. With the `shortcuts` option, you can use `\Them` instead.

```
\personobjpronounii[⟨person-label⟩]
```

Displays the second person singular objective pronoun according to the gender of the person identified by the label. In English, this is “you” regardless of gender.

```
\Personobjpronounii[⟨person-label⟩]
```

As `\personobjpronounii` but starts with a capital.

```
\peopleobjpronounii
```

Displays the second person plural objective pronoun according to the gender of all defined people. If only one person has been defined, this produces the same text as `\personobjpronounii`. With the `shortcuts` option, you can use `\thee` instead (as opposed to `\you` for the subjective pronoun).

```
\Peopleobjpronounii
```

As `\peopleobjpronounii` but starts with a capital. With the `shortcuts` option, you can use `\Thee` instead (as opposed to `\You` for the subjective pronoun).



```
\personpossadj[\langle person-label \rangle]
```

Displays the third person singular possessive adjective according to the gender of the person identified by the label. A possessive adjective indicates that something belongs to someone. For example, “her book” starts with the female third person singular possessive adjective:



```
\newperson*{name=Zoë,gender=female}  
\personpossadj\ book
```



```
\Personpossadj[\langle person-label \rangle]
```

As `\personpossadj` but starts with a capital.



```
\peoplepossadj
```

Displays the third person plural possessive adjective according to the gender of all defined people. If only one person has been defined, this produces the same text as `\personpossadj`. In English, there’s no difference between singular and plural possessive adjectives. With the `shortcuts` option, you can use `\their` instead.



```
\Peoplepossadj
```

As `\peoplepossadj` but starts with a capital. With the `shortcuts` option, you can use `\Their` instead.



```
\personpossadjii[\langle person-label \rangle]
```

Displays the second person singular possessive adjective according to the gender of the person identified by the label.



```
\Personpossadjii[\langle person-label \rangle]
```

As `\personpossadjii` but starts with a capital.



```
\peoplepossadjii
```

Displays the second person plural possessive adjective according to the gender of all defined people. If only one person has been defined, this produces the same text as `\personpossadjii`.

In English, there's no difference between singular and plural possessive adjectives. With the `shortcuts` option, you can use `\your` instead.

```
\Peoplepossadjii
```

As `\peoplepossadjii` but starts with a capital. With the `shortcuts` option, you can use `\Your` instead.

```
\personposspronoun[⟨person-label⟩]
```

Displays the third person singular possessive pronoun according to the gender of the person identified by the label. A possessive pronoun indicates ownership. For example, “the book is hers” ends with the female third person singular possessive pronoun:

```
\newperson*{name=Zoë,gender=female}
the book is \personposspronoun
```

```
\Personposspronoun[⟨person-label⟩]
```

As `\personposspronoun` but starts with a capital.

```
\peopleposspronoun
```

Displays the third person plural possessive pronoun according to the gender of all defined people. If only one person has been defined, this produces the same text as `\personposspronoun`. With the `shortcuts` option, you can use `\theirs` instead.

```
\Peopleposspronoun
```

As `\peopleposspronoun` but starts with a capital. With the `shortcuts` option, you can use `\Theirs` instead.

```
\personposspronounii[⟨person-label⟩]
```

Displays the second person singular possessive pronoun according to the gender of the person identified by the label.

```
\Personposspronounii[⟨person-label⟩]
```

As `\personposspronounii` but starts with a capital.

`\peopleposspronounii`

Displays the second person plural possessive pronoun according to the gender of all defined people. If only one person has been defined, this produces the same text as `\personposspronounii`. With the `shortcuts` option, you can use `\yours` instead.

`\Peopleposspronounii`

As `\peopleposspronounii` but starts with a capital. With the `shortcuts` option, you can use `\Yours` instead.

9.5.0.2. Relationships

`\personchild[⟨person-label⟩]`

Displays the person's relationship to their parents (son, daughter or child).

`\Personchild[⟨person-label⟩]`

As `\personchild` but starts with a capital.

`\peoplechild`

Displays the relationship of the defined people to their collective parents (sons, daughters or children). If only one person is defined, the result is the same as `\personchild`. With the `shortcuts` option, you can use `\children` instead.

`\Peoplechild`

As `\peoplechild` but starts with a capital. With the `shortcuts` option, you can use `\Children` instead.

`\personparent[⟨person-label⟩]`

Displays the person's relationship to their child (father, mother or parent).

`\Personparent[⟨person-label⟩]`

As `\personparent` but starts with a capital.

`\peopleparent`

Displays the relationship of the defined people to their collective children (fathers, mothers or parents). If only one person is defined, the result is the same as `\personparent`. With the `shortcuts` option, you can use `\parents` instead.

`\Peopleparent`

As `\peopleparent` but starts with a capital. With the `shortcuts` option, you can use `\Parents` instead.

`\personsibling[⟨person-label⟩]`

Displays the person's relationship to their sibling (brother, sister or sibling).

`\Personsibling[⟨person-label⟩]`

As `\personsibling` but starts with a capital.

`\peoplesibling`

Displays the relationship of the defined people to their collective siblings (brothers, sisters or siblings). If only one person is defined, the result is the same as `\personsibling`. With the `shortcuts` option, you can use `\siblings` instead.

`\Peoplesibling`

As `\peoplesibling` but starts with a capital. With the `shortcuts` option, you can use `\Siblings` instead.

9.5.1. Accessing Individual Information

The information provided when defining a person can be accessed with these commands. Where the label identifying the person is optional, `anon` will be used if omitted.

`\personfullname[⟨person-label⟩]`

Displays the person's full name (corresponding to the `fullname` option). Use `\peoplefullname` if you want a list of all defined people showing their full name.

`\personname [⟨person-label⟩]`

Displays the person's familiar or first name (corresponding to the `name` option). Use `\people-name` if you want a list of all defined people showing their name.

`\personforenames [⟨person-label⟩]`

Displays the person's forenames (corresponding to the `forenames` option). Use `\people-forenames` if you want a list of all defined people showing their forenames.

`\personsurname [⟨person-label⟩]`

Displays the person's surname (corresponding to the `surname` option). Use `\peoplesur-name` if you want a list of all defined people showing their surname.

`\persontitlesurname [⟨person-label⟩]`

Displays the person's title and surname separated by

`\persontitlesurnamesep`

initial: `\_`

If the person has no title then the full name will be used instead. Use `\peopletitlesur-name` if you want a list of all defined people showing their title and surname.

`\persongender {⟨person-label⟩}`

Displays the given person's gender using localisation. This maps the internal gender label to the corresponding language-sensitive text.

`\getpersongender {⟨cs⟩} {⟨person-label⟩}`

Defines `⟨cs⟩` to expand to the language-sensitive text identifying the given person's gender.

`\getpersongenderlabel {⟨cs⟩} {⟨person-label⟩}`

Defines `⟨cs⟩` to the internal gender label associated with the given person.

`\getpersonname {⟨cs⟩} {⟨person-label⟩}`

Defines $\langle cs \rangle$ to expand to the person's name.

```
\getpersonforenames{ $\langle cs \rangle$ }{ $\langle person-label \rangle$ }
```

Defines $\langle cs \rangle$ to expand to the person's forenames.

```
\getpersonsurname{ $\langle cs \rangle$ }{ $\langle person-label \rangle$ }
```

Defines $\langle cs \rangle$ to expand to the person's surname.

```
\getpersonfullname{ $\langle cs \rangle$ }{ $\langle person-label \rangle$ }
```

Defines $\langle cs \rangle$ to expand to the person's full name.

```
\getpersontitle{ $\langle cs \rangle$ }{ $\langle person-label \rangle$ }
```

Defines $\langle cs \rangle$ to expand to the person's title.

9.5.2. List People

The commands in this section produce a list of all the defined people. The separators are given by:

```
\twopeoplesep initial: \DTLlistformatlastsep
```

Inserted between the people when the list only contains two elements. This is simply defined to `\DTLlistformatlastsep`.

```
\personlastsep
```

Inserted between the final pair when the list contains more than two elements. This is defined to:

```
\DTLlistformatoxford  
\DTLlistformatlastsep
```

```
\personsep initial: \DTLlistformatsep
```

Inserted between people for all but the last pair. This is simply defined to `\DTLlistformatsep`.

`\peoplefullname`

Displays all the defined people, showing the full name for each person. Use `\personfullname` for an individual.

`\peoplename`

Displays all the defined people, showing the name for each person. Use `\personname` for an individual.

`\peopleforenames`

Displays all the defined people, showing the forenames for each person. Use `\personforenames` for an individual.

`\peoplesurname`

Displays all the defined people, showing the surname for each person. Use `\personsurname` for an individual.

`\peopletitlesurname`

Displays all the defined people, showing the title and surname for each person. Use `\person-titlesurname` for an individual.

9.6. Examples

9.6.1. Mail Merging

Sometimes when mail-merging, it may be necessary to reference a person by their pronoun, but it's inappropriate to make an assumption and the impersonal “he/she” construct is not only cumbersome but also dated. By defining a person with a gender (male, female or non-binary), the commands described in §9.5 may be used to insert a language-sensitive pronoun.

Example 193 uses the “scores” database (see §3.2.2) to write a letter to the parent of each student in the database, informing them of the student’s score and award. The letter environment implicitly adds scoping, so with the default `local=true` setting, the `anon` label can be reused for each iteration.

For simplicity, the letter class is used:

193

```
\documentclass{letter}
```

The `shortcuts` option is used to provide the simpler shortcut commands:

```
\usepackage[shortcuts]{person}
```

The data is iterated over using `\DTLmapdata`. I've used a mixture of `\DTLmapget-values` and `\DTLmapget`. You may prefer to get the parent, score and award values within `\DTLmapgetvalues` along with the other values.

```
\begin{DTLenvmapdata}
\begin{letter}{}
\DTLmapgetvalues{
  \Forename=forename, \Surname=surname, \Gender=gender
}
\newperson*{
  expand-once-name=\Forename,
  expand-once-surname=\Surname,
  gender=\Gender
}
\opening{Dear \DTLmapget{key=parent}}

Your \personchild\_\personfullname\_\_
received a score of \DTLmapget{key=score}
and was awarded a scholarship of
\DTLmapget{key=award}. We look forward to seeing
\them\_\_on \their\_\_arrival.

\closing{Yours Sincerely}
\end{letter}
\end{DTLenvmapdata}
```

The `\personfullname` command can be replaced with `\Forename\_\Surname`. The advantage of the `person` package is that the gender-dependent text can more easily be produced.

Example 193 shows the first four pages. Download the PDF to view the complete document. Note that the student Quinn doesn't have the gender field set in the database so the placeholder `\Gender` command will be null on that iteration. The `gender` option will detect this and set the gender to `unknown`. By way of contrast, the student Evelyn has the gender set to one of the non-binary labels, which means that the gender will be set to `nonbinary`. This doesn't

make a noticeable difference between the two letters (aside from their names, the names of their parents, the score and award) as the language-sensitive text is the same in both cases.

↑ Example 193: Mail Merging

August 23, 2026

Dear Ms Brown

Your daughter Jane Brown received a score of 75 and was awarded a scholarship of \$1,830. We look forward to seeing her on her arrival.

Yours Sincerely

August 23, 2026

August 23, 2026

Dear Mr and Mrs Smith

Your son John Smith, Jr received a score of 68 and was awarded a scholarship of \$1,560. We look forward to seeing him on his arrival.

Yours Sincerely

August 23, 2026

Dear Mr and Mrs Ó Coinn

Your child Quinn Ó Coinn received a score of 91 and was awarded a scholarship of \$3,280. We look forward to seeing them on their arrival.

Yours Sincerely

Dear Prof O'Leary

Your child Evelyn O'Leary received a score of 81.5 and was awarded a scholarship of \$2,460. We look forward to seeing them on their arrival.

Yours Sincerely

9.6.2. Order of Service

Defining a person with a name and gender can also be useful for other non-database documents, such as an order of service for a baptism or funeral. Since the document is much the same from one person to the next, documents of this nature are frequently simply copied and a search and replace edit is used to change the relevant text. However this can lead to errors (especially if the previous person's name was Mary!) With the person package, you need only change the definition of the person by modifying the arguments of `\newperson`.

Example 194 doesn't use a database so there's no need for the datatool package so the `base-only` option is used so that only `datatool-base` not `datatool` is loaded:

194

```
\usepackage[base-only]{person}
```

Only one person is defined so the default `anon` label can be used, which means there's no need to supply the label.

```
\newperson*{
  forenames=Mary Jane,
  name=Mary,
  surname=Doe,
  gender=f
}
```

9. Referencing People (person package)

This is a simple document for demonstration purposes. A real document would benefit from a custom class that provides better styling. This is just to illustrate the `person` package commands:

```
\begin{center}
\Large
In Memory of \personfullname
\end{center}
```

We are gathered here to remember our
\personsibling_\personname.
\Personpronoun_\will be much missed, and
\personpossadj_\family are in our prayers.

↑ Example 194: Memorial Order of Service

In Memory of Mary Jane Doe

We are gathered here to remember our sister Mary. She will be much missed, and her family are in our prayers.

Remember that the plural commands will behave like the singular ones if only one person has been defined so, for example, `\Peoplepronoun` may be used instead of `\Personpronoun` even if only one person has been defined.

Example 195 produces the same result as Example 194 but it uses the `shortcuts` package option:

```
\usepackage[base-only,shortcuts]{person}
```

This allows the use of the plural shortcut commands, which makes the code slightly more readable:

```
\begin{center}
\Large
In Memory of \peoplefullname
\end{center}
```

```
We are gathered here to remember our
\siblings\_\peoplename.
\They\_will be much missed, and
\their\_family are in our prayers.
```

↑ Example 195: Memorial Order of Service (Shortcuts)



In Memory of Mary Jane Doe

We are gathered here to remember our sister Mary. She will be much missed, and her family are in our prayers.

Example 196 also doesn't use a database:

196

```
\usepackage[base-only]{person}
```

In this case, two people are defined. This means that unique labels need to be supplied:

```
\newperson*[john]{
  forenames=John Joseph,
  name=John,
  gender=male
}

\newperson*[jane]{
  forenames=Jane Mary,
  name=Jane,
  gender=female
}
```

Again, this is a simple document for demonstration purposes where I've used `\title` and `\author` with `\maketitle` to show the title. A real document would benefit from a custom class that provides better styling. This is just to illustrate the *person package* commands:

```
\title{Baptism of}
\author{\peopleforenames}
\maketitle
```

```
Today we welcome \peoplename\
into God's family, may He guide
and protect \peopleobjpronoun.
```

Note that `\` (backslash space) is required after `\peoplename` to force a space.

↑ Example 196: Baptism Order of Service



Baptism of
John Joseph & Jane Mary
August 23, 2026

Today we welcome John & Jane into God's family, may He guide and protect them.

Note that this has the ampersand character (&) rather than the textual “and”. Add localisation support if text is preferred. If localisation support is available but the ampersand character is preferred, then use `and=symbol` to switch.

Again the `shortcuts` option may be used to enable the shorter, more readable, commands.

197

```
\usepackage[
  base-only, % no datatool.sty
  locales=en-GB, % (requires datatool-english)
  shortcuts
]{person}
```

The `datetime2` package also uses `tracklang` for its localisation support. This means that if it's loaded as well, and `datetime2-english` is also installed, then it can also pick up the localisation but beware of the ordering.

For example, load `datetime2` first with regional support:

```
\usepackage[en-GB]{datetime2}
\usepackage[
  base-only, % no datatool.sty
  shortcuts
]{person}
```

or load `person` first with regional support:

```
\usepackage[
  base-only, % no datatool.sty
  locales=en-GB, % (requires datatool-english)
  shortcuts
]{person}
\usepackage[useregional]{datetime2}
```

Alternatively:

```
\documentclass[en-GB]{article}
\usepackage[useregional]{datetime2}
\usepackage[
  base-only, % no datatool.sty
  shortcuts
]{person}
```

Or:

```
\documentclass{article}
\usepackage[british]{babel}
\usepackage[useregional]{datetime2}
\usepackage[
  base-only, % no datatool.sty
  shortcuts
]{person}
```

Example 197 makes this modification to Example 196, along with:

```
Today we welcome \peoplename\_\_
into God's family, may He guide
and protect \them.
```


↑ Example 197: Baptism Order of Service (Shortcuts and Localisation)






Baptism of

John Joseph and Jane Mary

23rd August 2026

Today we welcome John and Jane into God's family, may He guide and protect them.

9.7. Advanced Commands

You can alter or set an attribute for a given person using:

```
\person_set_attribute:nnn {<person-label>} {<attribute>} {<value>}
variant: nnV
```

Note that this doesn't test for existence of either the label or the attribute name, nor does it test the validity of the value. The change is governed by the `local` setting. The following attributes are set by `\newperson`:

- `name`: the person's familiar or first (corresponding to the `name` option);
- `surname`: the person's surname (corresponding to the `surname` option);
- `title`: the person's title or form of address (corresponding to the `title` option);
- `fullname`: the person's formal or full name (corresponding to the `fullname` option);
- `gender`: the internal gender label (which is obtained from the `gender` option and mapped to the applicable constant).

You can get an attribute with:

```
\person_get_attribute:nn {<person-label>} {<attribute>}
```

which expands to the value of the attribute, or

```
\person_get_attribute:Nnn <tl var> {<person-label>} {<attribute>}
```

which sets the token list variable to the value of the attribute. Again, there's no test if the label or attribute exists.

You can undefine an attribute with:

```
\person_unset_attribute:nn {<person-label>} {<attribute>}
```

The change is governed by the `local` setting. This command is primarily provided for use in the `\person_remove_appto:n` code.

If you undefine the name attribute, the existence test will fail.

9.7.1. Conditionals

```
\ifpersonexists{<person-label>}{<true>}{<false>}
```

If a person has been identified with the given label, this does `<true>` otherwise it does `<false>`. The argument is trimmed and expanded before testing.

```
\PersonIfMale{<person-label>}{<true>}{<false>}
```

If the person identified by the label is male, this does `<true>`, otherwise it does `<false>`.

```
\PersonIfAllMale[<person-label-list>]{<true>}{<false>}
```

If all the listed people are male, this does `<true>`, otherwise it does `<false>`. If the option argument is missing, this will simply compare `\PersonTotalCount` with `\PersonMaleCount`.

```
\PersonIfFemale{<person-label>}{<true>}{<false>}
```

If the person identified by the label is female, this does `<true>`, otherwise it does `<false>`.

```
\PersonIfAllFemale[<person-label-list>]{<true>}{<false>}
```

If all the listed people are female, this does `<true>`, otherwise it does `<false>`. If the option argument is missing, this will simply compare `\PersonTotalCount` with `\PersonFemaleCount`.

```
\PersonIfNonBinary{<person-label>}{<true>}{<false>}
```

If the person identified by the label is non-binary, this does *<true>*, otherwise it does *<false>*. Note that this does *<false>* if the gender is unknown.

```
\PersonIfAllNonBinary[<person-label-list>]{<true>}{<false>}
```

If all the listed people are non-binary, this does *<true>*, otherwise it does *<false>*. If the option argument is missing, this will simply compare `\PersonTotalCount` with `\PersonNonBinaryCount`.

```
\PersonIfUnknownGender{<person-label>}{<true>}{<false>}
```

If the person identified by the label has the gender set to unknown, this does *<true>*, otherwise it does *<false>*.

```
\PersonIfAllUnknownGender[<person-label-list>]{<true>}{<false>}
```

If all the listed people have the gender set to unknown, this does *<true>*, otherwise it does *<false>*. If the option argument is missing, this will simply compare `\PersonTotalCount` with `\PersonUnknownGenderCount`.

If you have $\LaTeX$ 3 syntax enabled, you can use the following commands.

```
\person_if_exist:N $\TeX$  {<person-label>}{<true>}{<false>}
\person_if_exist_p:n {<person-label>}
```

Conditional and predicate that tests if a person has been defined with the given label.

The following commands test a label, which may be supplied as a token list variable or as a token list, against the recognised internal gender labels.

```
\person_gender_case:Nnnnnn <tl-var> {<invalid-case>}
{<male-case>} {<female-case>} {<non-binary-case>} {<unknown-case>}
```

Compares the given token list variable (using `\tl_if_eq:NNTF`) against the constants `\c_person_male_label_tl`, `\c_person_female_label_tl`, `\c_person_nonbinary_label_tl`, and `\c_person_unknown_label_tl` and does the applicable case or *<invalid-case>* if no match.

```
\person_gender_case:nnnnn <token-list> {<invalid-case>}
{<male-case>} {<female-case>} {<non-binary-case>} {<unknown-case>}
```

As above but compares a token list.

```
\person_gender_case:Nnnnn <tl-var> {<male-case>} {<female-case>}
{<non-binary-case>} {<unknown-case>}
```

A shortcut that uses `\person_gender_case:Nnnnn`. The invalid case triggers an error and does the same as the unknown case.

```
\person_gender_case:nnnnn <token-list> {<male-case>}
{<female-case>} {<non-binary-case>} {<unknown-case>}
```

As above but compares a token list.

In the event that you only need to know if the label is valid, you can use:

```
\person_do_if_valid_gender:nT {<token-list>} {<code>}
```

This will do the true part if the label is valid, otherwise it will trigger an error.

The following command simply compares `\PersonTotalCount` with `\PersonMaleCount`, `\PersonFemaleCount` and `\PersonNonBinaryCount`.

```
\person_all_gender_case:nnnn {<all-male-case>}
{<all-female-case>} {<all-non-binary-case>} {<other-case>}
```

Tests if all defined people have the same gender. This will do `<all-male-case>` if all defined people have been identified as male, `<all-female-case>` if all defined people have been identified as female, `<all-non-binary-case>` if all defined people have been identified as non-binary, and `<other-case>` otherwise (that is, a mixture of genders or all defined people have no gender specified).

9.7.2. Iterating Through Defined People

The loop commands have changed in version 3.0 to map over a sequence variable instead of using `\@for`. This means that you can no longer break out of the loop using the methods provided by `xfor`. Instead you can break the loop with:

```
\foreachpersonbreak
```

This simply uses `\seq_break:` or `\clist_break:` depending on the context.



The following commands will need to be scoped if nested.



```
\forallpeople[⟨people-label-list⟩]{⟨label-cs⟩}{⟨body⟩}
```

Loops through the list of people and, at each iteration, defines $\langle label-cs \rangle$ to the current label and does $\langle body \rangle$. If the optional argument is omitted, the list of all defined people is assumed.



```
\foreachperson(⟨name-cs⟩,⟨full-name-cs⟩,⟨gender-cs⟩,⟨label-cs⟩)\in
{⟨label-list⟩}\do{⟨body⟩}
```

Iterates through the list of people and, at each iteration, assigns $\langle name-cs \rangle$ to the person's name, $\langle full-name-cs \rangle$ to the person's full name, $\langle gender-cs \rangle$ to the language-sensitive gender text, and $\langle label-cs \rangle$ to the person's label, and then does $\langle body \rangle$. The $\backslash in \{ \langle label-list \rangle \}$ is optional. If omitted the list of all defined people is assumed.

9.7.3. Localisation

As described in §2.3, the `datatool-base` package (which will automatically be loaded by the `person` package, if not already loaded) provides localisation support via the `tracklang` interface. The `person` package uses the same interface to load the file `person-⟨locale⟩.ldf` for each tracked locale if that file is installed on $\text{\TeX}$'s path.

The supplementary `datatool-english` package (which needs to be installed separately), described in §2.3.5, includes `person-english.ldf`, which provides English localisation support for the `person` package. This file may be used as a template for other languages.

If the localisation support provides options (which `person-english.ldf` currently doesn't), the sub-module should be "person" when using `\DTLsetLocaleOptions`.



```
\PersonSetLocalisation{⟨gender-label⟩}{⟨type⟩}{⟨value⟩}
```

Sets the localisation text for the given gender label for the given type. The $\langle gender-label \rangle$ must be one of the internal labels: `male`, `female`, `nonbinary` or `unknown`. The $\langle value \rangle$ argument is the localised text. For example, `person-english.ldf` defines the subjective third person pronouns as follows:



```
\PersonSetLocalisation{male}{pronoun}{he}
\PersonSetLocalisation{female}{pronoun}{she}
\PersonSetLocalisation{nonbinary}{pronoun}{they}
\PersonSetLocalisation{unknown}{pronoun}{they}
```

9. Referencing People (*person package*)

```
\PersonSetLocalisation{male}{pluralpronoun}{they}  
\PersonSetLocalisation{female}{pluralpronoun}{they}  
\PersonSetLocalisation{nonbinary}{pluralpronoun}  
{they}  
\PersonSetLocalisation{unknown}{pluralpronoun}{they}
```

The *<type>* argument is essentially a property name where the plural alternative is as the singular but prefixed with “plural”.



If there is no value set for a particular gender, commands that use localisation text will fallback first on “unknown” and then on “nonbinary”. This means that the *person package* only defines “unknown” in certain cases to avoid redundancy. However, localisation files need to take multilingual documents into account and should therefore set the values for all genders, even if they are duplicates, otherwise only the ones that are set will change when the language changes.

Recognised values of *<type>* are listed below.

- *pronoun*: The third person singular subjective pronoun.
- *pluralpronoun*: The third person plural subjective pronoun.
- *pronoun2*: The second person singular subjective pronoun.
- *pluralpronoun2*: The second person plural subjective pronoun.
- *objpronoun*: The third person singular objective pronoun.
- *pluralobjpronoun*: The third person plural objective pronoun.
- *objpronoun2*: The second person singular objective pronoun.
- *pluralobjpronoun2*: The second person plural objective pronoun.
- *possadj*: The third person singular possessive adjective.
- *pluralpossadj*: The third person plural possessive adjective.
- *possadj2*: The second person singular possessive adjective.
- *pluralpossadj2*: The second person plural possessive adjective.
- *posspronoun*: The third person singular possessive pronoun.
- *pluralposspronoun*: The third person plural possessive pronoun.
- *posspronoun2*: The second person singular possessive pronoun.

9. Referencing People (*person* package)

- `pluralposspronoun2`: The second person plural possessive pronoun.
- `child`: Noun indicating the person's relationship to their parents (that is, son, daughter or child).
- `pluralchild`: The plural form of the above (that is, sons, daughters or children).
- `parent`: Noun indicating the person's relationship to their children (that is, father, mother or parent).
- `pluralparent`: The plural form of the above (that is, fathers, mothers or parents).
- `sibling`: Noun indicating the person's relationship to their sibling (that is, brother, sister or sibling).
- `pluralsibling`: The plural form of the above (that is, brothers, sisters or siblings).
- `gender`: The person's gender. Note that this isn't a gender identifying label for `\newperson` but is used by `\foreachperson`, `\getpersongender`, `\persongender` and `\Persongender`.

The localisation file should also use `\PersonSetMaleLabels`, `\PersonSetFemaleLabels` and `\PersonSetNonBinaryLabels` to the locale gender labels for use in `\newperson`.

You can add support for other types, but make sure you document that those types are only available for your localisation support. The language text can be obtained with the following commands. In each case, the `<person-label>` argument is used to fetch the gender label for the given person in order to obtain the correct text.



```
\person_language_text:nn {<person-label>} {<type>}
```

Expands to the localisation text for the given `<type>` for the gender label associated with the given person. For example, `\personpronoun` is defined as:

```
\NewDocumentCommand \personpronoun { O{anon} }
{
  \person_language_text:nn { #1 } { pronoun }
}
```



```
\person_Language_text:nn {<person-label>} {<type>}
```

As above, but converts the text to sentence case. For example, `\Personpronoun` is defined as:

```
\NewDocumentCommand \Personpronoun { O{anon} }
{
  \person_Language_text:nn { #1 } { pronoun }
}
```



```
\person_language_all_text:n {<type>}
```

The `<type>` argument should be the singular type. If more than one person has been defined, this uses the text for the type given by `plural<type>` according to the group gender (unknown for a mixture); if only one person is defined, this uses the text for `<type>` according to the gender of that person; otherwise it triggers a warning and uses the plural. For example, `\peoplepronoun` is defined as:

```
\NewDocumentCommand \peoplepronoun { }
{
  \person_language_all_text:n { pronoun }
}
```



```
\person_Language_all_text:n {<type>}
```

As above, but converts the text to sentence case. For example, `\Peoplepronoun` is defined as:

```
\NewDocumentCommand \Peoplepronoun { }
{
  \person_Language_all_text:n { pronoun }
}
```

9.7.4. Hooks

There are hooks available when defining a new person.



```
\l_person_label_tl
```

This token list variable will expand to the current label in the following hooks.



```
\person_new_appto_start:n {<code>}
```

Appends `<code>` to a hook used at the start of `\newperson`.



```
\person_new_appto_end:n {<code>}
```

Appends *<code>* to a hook used at the end of `\newperson` (after the label has been added to the internal sequence, and after the internal integer variables used to keep track of totals are incremented). Within *<code>* you can set an attribute with `\person_set_attribute:nnn`.



```
\person_remove_appto:n {<code>}
```

Appends *<code>* to the hook used by `\removeperson` and also for each iteration in `\removepeople` and `\removeallpeople`. You can use `\person_unset_attribute:nn` in *<code>* to undefine a custom attribute. Note that `\removeallpeople` zeroes the internal integer variables and clears the internal sequence at the end, whereas `\removepeople` decrements the variables and pops the label off the sequence at each iteration.

If you want to add any extra keys for use with `\newperson*`, the keys are defined with `\keys_define:nn` (l3keys) and module `datatool/person`. For example, to add a key to set the person's date of birth (replace `mypkg` as applicable):



```
\tl_new:N \l__mypkg_dob_tl
\person_new_appto_start:n
{
  \tl_clear:N \l__mypkg_dob_tl
}
\person_new_appto_end:n
{
  \person_set_attribute:nnV
    \l_person_label_tl { dob } \l__mypkg_dob_tl
}
\person_remove_appto:n
{
  \person_unset_attribute:nn
    \l_person_label_tl { dob }
}
\keys_define:nn { datatool/person }
{
  dob .tl_set:N = \l__mypkg_dob_tl
}
```

10. Acknowledgements

Many thanks to Morten Høgholm for providing a much more efficient way of storing the information in databases which has significantly improved the time it takes to $\text{\LaTeX}$ documents containing large databases.

Part II.

Summaries and Index

Symbols

Symbol	Description
	A counter is being described.
	The syntax and usage of a command, environment or option etc.
	A command, environment or option that is now deprecated.
	A command, environment or option that should not be used with datatool.
	An important message.
	Prominent information.
	L ^A T _E X code to insert into your document.
	The definition of an option value.
	Problematic code which should be avoided.
	How the example code should appear in the PDF.
	An option that takes a value.
	A command-line application invocation that needs to be entered into a terminal or command prompt.
	A boolean option that is initially false.
	A boolean option that is initially true.
	Text in a transcript or log file or written to STDOUT or STDERR.
	An option that doesn't take a value.
	A warning.

Glossary

0x $\langle XX \rangle$

A hexadecimal value. Used in this manual to denote a character by its codepoint, particularly control characters that don't have a visible symbol (glyph). For example, 0x0A denotes the line feed character. When entering the character into L^AT_EX code you can use the double caret notation, such as $\wedge\wedge\text{J}$, but you may first need to set its category code appropriately.

American Standard Code for Information Interchange (ASCII)

A single-byte character encoding. Related blog article: Binary Files, Text Files and File Encodings.¹

$\langle Assign-list \rangle$

Indicates a comma-separated list of $\langle cs \rangle = \langle key \rangle$ assignments, where $\langle cs \rangle$ is a placeholder command (token list variable) and $\langle key \rangle$ is a column key or property name. The $\langle cs \rangle$ command will be assigned to the applicable value identified by $\langle key \rangle$.

Comma Separated Values (CSV)

A CSV file is a text field that uses a comma to separate fields (see §3.15.1.1). A CSV list is a string (rather than specifically the content of a file) where values are separated by a comma, so each row in a CSV file is a CSV list. Some commands that allow a CSV list as the argument may allow the argument to be a command whose definition is a CSV list (see §2.9).

Datum control sequence

A control sequence whose replacement text is specially formatted to include the formatted/string content, numerical value (if applicable), currency symbol (if applicable), and data type (string, integer, real or currency).

Datum item

Data that's formatted according to the replacement text of a datum control sequence. See §2.2.

Expansion

Single expansion (expand once) is where a command is replaced by its definition. Full expansion is where all the commands within the definition are also expanded recursively. Robust commands don't expand and fragile commands need to be protected from expansion to prevent an error from occurring. Expansion is important in certain situations, such as in the argument of section titles where the title also needs to be in the PDF bookmark, which requires just text and no formatting or assignments.

¹dickimaw-books.com/blog/binary-files-text-files-and-file-encodings/

Formatted number

A number that uses the number group character and (if a decimal) the decimal character according to the current setting of `\DTLsetnumberchars`, optionally prefixed with a currency symbol. Note that the number group character is optional but, if present, it must be at intervals of three digits.

Julian Date (JD)

A decimal that's the sum of the `jd` and the `time`.

Julian Day Number (JDN)

An integer assigned to a whole solar day in the Julian day count starting from noon Universal Time.

Julian Time Fraction (JF)

The time of day since noon UT as a decimal fraction of one day, with 0.5 representing midnight UT.

Plain number

A number without a currency symbol and without number group characters. If the number is a decimal, a decimal point is used, regardless of `\DTLsetnumberchars`.

Purify

Fully expand and remove remaining non-expandable commands. This is implemented by $\text{\LaTeX}$ 3's `\text_purify:n` function. See “The $\text{\LaTeX}$ 3 Interfaces” documentation for further details.

Sorted element

An element of a comma-separated list obtained with `\DTLsortwordlist` consisting of a marker, the original element, the sort value used for sorting, and the corresponding letter group.

Structured Query Language (SQL)

A language used to manage data stored within a database management system.

Tab Separated Values (TSV)

As CSV but uses a tab character as the separator.

Unicode Transformation Format (8-bit) (UTF-8)

A variable-width encoding that uses 8-bit code units. This means that some characters are represented by more than one byte. $\text{\LaTeX}$ and $\text{\LaTeX}$ treat the multi-byte sequence as a single token, but the older $\text{\LaTeX}$ formats have single-byte tokens, which can cause complications, although these have mostly been addressed with the newer kernels introduced over the past few years. Related blog article: [Binary Files, Text Files and File Encodings](https://dickimaw-books.com/blog/binary-files-text-files-and-file-encodings/).²

²dickimaw-books.com/blog/binary-files-text-files-and-file-encodings/

Command Summary

A

\Acr{*⟨label⟩*}

datagidx

§8.7.1;
608

As `\acr` but sentence case.

\acr{*⟨label⟩*}

datagidx

§8.7.1;
607

Displays the short form if the abbreviation identified by *⟨label⟩* has been marked as used, otherwise shows both the long and short form.

\acronymfont{*⟨text⟩*}

datagidx

§8.7; 606

Used to encapsulate the short form of an abbreviation defined with `\newacro`.

\Acrpl{*⟨label⟩*}

datagidx

§8.7.1;
608

As `\acrpl` but sentence case.

\acrpl{*⟨label⟩*}

datagidx

§8.7.1;
607

Displays the plural short form if the abbreviation identified by *⟨label⟩* has been marked as used, otherwise shows both the plural long and short form.

 **\addfemalelabel**{*⟨label⟩*}

person

§9.4; 634

This deprecated command has been replaced with `\PersonAddFemaleLabel` and may be removed in future.

 **\addmalelabel** { *<label>* }

person

§9.4; 633

This deprecated command has been replaced with `\PersonAddMaleLabel` and may be removed in future.

C

\c_datatool_apostrophe_regex

datatool-base v3.0+

§2.3.2; 43

Constant regular expression that matches either a straight apostrophe character or a closing single quote character.

\c_datatool_empty_datum_tl

datatool-base v3.0+

Constant datum control sequence representing an empty value with the unknown data type.

\c_datatool_string_int

datatool-base v3.0+

Constant integer (0) used to identify the string data type. Note that datatool provides `\DTL-stringtype`, which is the older command.

\children

(requires `shortcuts` option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for `\peoplechild`.

\Children

datagidx v2.15+

§8.9.1;
620

Placeholder in `\printterms` that expands to the current entry's `Child` value.

\Children

(requires `shortcuts` option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for `\Peoplechild`.

\c_person_female_label_t1

person v3.0+

Constant that expands to `female`, which is the internal label used to identify the female gender.

\c_person_male_label_t1

person v3.0+

Constant that expands to `male`, which is the internal label used to identify the male gender.

\c_person_nonbinary_label_t1

person v3.0+

Constant that expands to `nonbinary`, which is the internal label used to identify the nonbinary gender.

\c_person_unknown_label_t1

person v3.0+

Constant that expands to `unknown`, which is the internal label used to identify the unknown gender.

\CurrentLocation

datagidx v2.15+

§8.9.1;
620

Placeholder in `\printterms` that expands to the current entry's `CurrentLocation` value.

D

\datagidxconvertchars

datagidx

§8.9.2;
621

Used by `\newterm` when creating a label or sort value.

\datagidxcurrentgroup

datagidx v2.15+

§8.9.1;
620

Placeholder in `\printterms` that expands to the current entry's `LetterGroup` value.

`\datagidxdictindent`

initial: 1em datagidx

§8.8.2.5;
616

Indentation used by the `dict` style.

`\datagidxend`

datagidx

Style command used by `\printterms` at the end of the list.

`\datagidxitem`

datagidx

Style command used to display the current entry within `\printterms`.

`\datagidxlastlabel`

datagidx

§8.9.2;
620

Defined at the end of `\newterm` to expand to the new term's label.

`\datagidxlink` { *⟨target-name⟩* } { *⟨text⟩* }

datagidx

§8.9.3;
622

If hyperlinks are support, creates a hyperlink with *⟨text⟩* as the link text, otherwise just does *⟨text⟩*.

`\datagidxlocalign`

initial: `\raggedleft` datagidx

§8.8.1;
614

Alignment of the location list when the location width has been set.

`\datagidxlocationwidth`

initial: 0pt datagidx

§8.8.1;
614

Dimension that indicates the space to allocate for each location list. If zero or negative, the location list will simply occupy its natural space.

`\datagidxmapdata` { *⟨body⟩* }

datagidx v3.0+

§8.9.1;
620

Used by styles to iterate over the database.

\datagidxnewstyle { *⟨style-name⟩* } { *⟨definitions⟩* }

datagidx

§8.9.3;
621

Defines a new style called *⟨style-name⟩* for use with the `style` setting.

\datagidxprevgroup

datagidx

§8.9.1;
619

Assigned by `\DTLgidxForeachEntry` to the letter group for the previous term.

\datagidxsetstyle { *⟨style-name⟩* }

datagidx

§8.9.3;
621

Used by `\printterms` to set the current style.

\datagidxstart

datagidx

Style command used by `\printterms` at the start of the list.

\datagidxsymalign

initial: `\centering` datagidx

§8.8.1;
614

Alignment of the symbol when the symbol width has been set.

\datagidxsymbolwidth

initial: `0pt` datagidx

§8.8.1;
614

Dimension that indicates the space to allocate for each symbol. If zero or negative, the symbol will simply occupy its natural space.

\datagidxtarget { *⟨target-name⟩* } { *⟨text⟩* }

datagidx

§8.9.3;
622

Creates a hyperlink target, if supported, and does *⟨text⟩*.

\datagidxwordifygreek

datagidx

§8.9.2;
621

Used by `\newterm` when creating a label or sort value.

\dataplot_apply_pgfttransform:

dataplot v3.0+

§6.3; 511

For use within `\DTLplotatbegin tikz` this may be used to restore `\DTLplot`'s transformation matrix if the hook has altered it.

\dataplot_db_count:

dataplot v3.0+

§6.3.1;
512

May be used with `\DTLplot` hooks to return the total number of database names.

\dataplot_get_default_legend:Nnnnn $\langle tl-var \rangle$ $\{ \langle db-index \rangle \}$
 $\{ \langle db-name \rangle \}$ $\{ \langle x-key \rangle \}$ $\{ \langle y-key \rangle \}$ dataplot v3.0+

§6.3.2;
513

Gets the default legend label (if not provided by `legend-labels`). The label should be stored in the provided token list variable $\langle tl-var \rangle$, which will initially be empty.

\dataplot_legend_add_begin:

dataplot v3.0+

§6.3.3;
514

Adds the beginning of the legend code to `\l_dataplot_legend_tl`.

\dataplot_legend_add_end:

dataplot v3.0+

§6.3.3;
515

Adds the end of the legend code to `\l_dataplot_legend_tl`.

\dataplot_x_key_count:

dataplot v3.0+

§6.3.1;
512

May be used with `\DTLplot` hooks to return the total number of items given in the `x` list.

\dataplot_y_key_count:

dataplot v3.0+

§6.3.1;
512

May be used with `\DTLplot` hooks to return the total number of items given in the `y` list.

\datatool_adjust_sign_fmt:n $\{ \langle sign \rangle \}$

datatool-base v3.0+

§2.6; 128

Used by `\datatool_prefix_adjust_sign:nnn` and `\datatool_suffix_adjust_sign:nnn` to format the sign.

\datatoolasciient

datatool-base v3.0+

§2.9.5.3;
166

Expands to nothing normally but expands to the delete character (0x7F, the highest ASCII character) inside \dtlwordindexcompare, \dtlletterindexcompare and \DTLsortwordlist.

\datatoolasciistart

datatool-base v3.0+

§2.9.5.3;
166

Expands to nothing normally but expands to null character inside \dtlwordindexcompare, \dtlletterindexcompare and \DTLsortwordlist.

\datatoolctrlboundary

datatool-base v3.0+

§2.9.5.3;
166

Expands to nothing normally but expands to the control character 0x1F, inside \dtlwordindexcompare, \dtlletterindexcompare and \DTLsortwordlist.

\datatoolcurrencysymbolprefixfmt { *tag* }

datatool-base v3.0+

§2.3.2; 45

Used by \datatool_currency_symbol_region_prefix:n to format the tag. Simply expands to *tag* by default.

\datatool_currency_symbol_region_prefix:n { *tag* }

datatool-base v3.0+

§2.3.2; 44

For use by regions that support a prefix for the currency symbol. The *tag* should typically be the region code. This command will encapsulate the *tag* with \datatoolcurrencysymbolprefixfmt.

\DataToolDateFmt { *year* } { *month* } { *day* } { *dow* }

datatool-base v3.0+

§2.7; 138

Inserted by `parse=auto-format` to format a date. The arguments should all be integers, but *dow* may be empty. By default, simply uses \DTLCurrentLocaleFormatDate.

\DataToolDateTimeFmt { *date-specs* } { *time-specs* } { *offset-specs* }

datatool-base v3.0+

§2.7; 137

Inserted by `parse=auto-format` to format timestamps. The arguments may either be

empty or the appropriate arguments to pass to `\DataToolDateFmt` ($\langle date-specs \rangle$ should be $\{\langle year \rangle\}\{\langle month \rangle\}\{\langle day \rangle\}\{\langle dow \rangle\}$), `\DataToolTimeFmt` ($\langle time-specs \rangle$ should be $\{\langle hour \rangle\}\{\langle minute \rangle\}\{\langle second \rangle\}$) and `\DataToolTimeZoneFmt` ($\langle offset-specs \rangle$ should be $\{\langle tzh \rangle\}\{\langle tzm \rangle\}$).

`\datatool_datum_currency:Nnnnn` $\langle marker-cs \rangle$ $\{\langle string \rangle\}$ $\{\langle value \rangle\}$
 $\{\langle currency \rangle\}$ $\{\langle type \rangle\}$ datatool–base v3.0+

§2.2.4.1;
23

Expands to $\langle currency \rangle$.

`\datatool_datum_fp:nnn` $\{\langle fp-value \rangle\}$ $\{\langle fp-var-content \rangle\}$ $\{\langle decimal \rangle\}$
datatool–base v3.0+

§2.2.4; 21

Used to markup a decimal value.

`\datatool_datum_show:N` $\{\langle cs \rangle\}$ datatool–base v3.0+

§2.2.3; 18

Interrupts the $\text{\LaTeX}$ run and shows the component parts of the given datum control sequence.

`\datatool_datum_string:Nnnnn` $\langle marker-cs \rangle$ $\{\langle string \rangle\}$ $\{\langle value \rangle\}$
 $\{\langle currency \rangle\}$ $\{\langle type \rangle\}$ datatool–base v3.0+

§2.2.4.1;
22

Expands to $\langle string \rangle$.

`\datatool_datum_type:Nnnnn` $\langle marker-cs \rangle$ $\{\langle string \rangle\}$ $\{\langle value \rangle\}$
 $\{\langle currency \rangle\}$ $\{\langle type \rangle\}$ datatool–base v3.0+

§2.2.4.1;
23

Expands to $\langle type \rangle$.

`\datatool_datum_value:Nnnnn` $\langle marker-cs \rangle$ $\{\langle string \rangle\}$ $\{\langle value \rangle\}$
 $\{\langle currency \rangle\}$ $\{\langle type \rangle\}$ datatool–base v3.0+

§2.2.4.1;
22

Expands to $\langle value \rangle$.

\datatool_db_state:nnnn {*<db-name>*} {*<not empty>*} {*<empty>*} {*<not exists>*}

datatool v3.0+

§3.6; 236

If the database identified by *<db-name>* exists and is not empty, this does *<not empty>*, if it exists but is empty, this does *<empty>*. If the database doesn't exist, this does *<not exists>*.

\datatool_def_currency:nnn {*<ISO>*} {*<symbol>*} {*<string>*}

variants: nnV nne datatool-base v3.0+

§2.6; 125

Shortcut that uses **\datatool_def_currency:nnnn** with the first argument set to **\dtlcurrdefaultfmt**.

\datatool_def_currency:nnnn {*<fmt>*} {*<ISO>*} {*<symbol>*} {*<string>*}

variants: nnnV nnne datatool-base v3.0+

§2.6; 125

Used by **\DTLdefcurrency** to define a new currency. Note that, unlike **\DTLdefcurrency**, no category code change is performed so make sure that the final argument contains the appropriate category code.

\datatool_extract_timestamp:NN *<datum-cs>* *<result-tl>*

datatool-base v3.0+

§2.2.4; 21

Extracts the date/time data stored in the given datum control sequence and stores the result in the token list register *<result-tl>*.

\datatoolGBcurrencyfmt {*<symbol>*} {*<value>*}

datatool-GB.ldf

Currency format for GBP (requires **datatool-regions**, which is not included with **datatool**).

\datatool_get_first_grapheme:nN {*<text>*} *<tl-var>*

datatool-base v3.0+

§2.8.3;
148

Obtains the first grapheme of *<text>* and defines *<tl-var>* to expand to that grapheme.

\datatool_get_first_letter:nN {*<text>*} *<tl-var>*

datatool-base v3.0+

§2.8.3;
148

Similar to **\datatool_get_first_grapheme:nN** but will skip leading punctuation.

```
\datatool_if_any_int_datum_type:nTF {<n>} {<true>}
{<false>}
\datatool_if_any_int_datum_type_p:n {<n>} datatool-base v3.0+
```

§2.2.4; 22

Tests if the integer $\langle n \rangle$ represents a temporal data type.

```
\datatool_if_has_key:nnTF {<db-name>} {<key>} {<true>}
{<false>}
\datatool_if_has_key_p:nn {<db-name>} {<key>} datatool v3.0+
```

§3.6; 237

Tests if the database with the label $\langle db-name \rangle$ exists and has a column with the given $\langle key \rangle$ (label).

```
\datatool_if_letter:nTF {<grapheme>} {<true>} {<false>}
datatool-base v3.0+
```

§2.8.3;
148

Tests if $\langle grapheme \rangle$ is a letter.

```
\datatool_if_null:nTF <tl var> {<true>} {<false>}
\datatool_if_null_p:N <tl var> datatool-base v3.0+
```

§3.10.2;
316

Tests if $\langle tl var \rangle$ represents null (see §3.10).

```
\datatool_if_null:nTF {<tl>} {<true>} {<false>}
datatool-base v3.0+
```

§3.10.2;
317

Tests if $\langle tl \rangle$ represents null (see §3.10).

```
\datatool_if_null_or_empty:nTF <tl var> {<true>} {<false>}
\datatool_if_null_or_empty_p:N <tl var> datatool-base v3.0+
```

§3.10.2;
316

Tests if $\langle tl var \rangle$ is empty or represents null (see §3.10).

```
\datatool_if_null_or_empty:nTF {<tl>} {<true>} {<false>}
datatool-base v3.0+
```

§3.10.2;
317

Tests if $\langle tl \rangle$ is empty or represents null (see §3.10).


```
\datatool_if_number_only_datum_type:nTF {<n>} {<true>}
{<false>}
\datatool_if_number_only_datum_type_p:n {<n>}
datatool-base v3.0+
```

§2.2.4; 22

Tests if the integer $\langle n \rangle$ represents a temporal data type.

```
\datatool_if_numeric_datum_type:nTF {<n>} {<true>}
{<false>}
\datatool_if_numeric_datum_type_p:n {<n>} datatool-base v3.0+
```

§2.2.4; 22

Tests if the integer $\langle n \rangle$ represents a numeric data type.

```
\datatool_if_row_start:nnTF {<row-num>} {<col-num>} {<true>}
{<false>}
\datatool_if_row_start_p:nn {<row-num>} {<col-num>}
datatool v3.0+
```

§3.7.2;
256

For use with display hooks such as `\DTLdisplaydbAddItem`, this tests if the given combination of $\langle row-num \rangle$ and $\langle col-num \rangle$ corresponds to the first column of the tabular or longtable environment, taking the `per-row` setting into account.

```
\datatool_if_temporal_datum_type:nTF {<n>} {<true>}
{<false>}
\datatool_if_temporal_datum_type_p:n {<n>} datatool-base v3.0+
```

§2.2.4; 22

Tests if the integer $\langle n \rangle$ represents a temporal data type.

```
\datatool_if_valid_datum_type:nTF {<n>} {<true>} {<false>}
\datatool_if_valid_datum_type_p:n {<n>} datatool-base v3.0+
```

§2.2.4; 21

Tests if the integer $\langle n \rangle$ represents a valid data type (including unknown).

```
\datatool_if_value_eq:NNTF <tl var1> <tl var2> {<true>} {<false>}
datatool-base v3.0+
```

§2.2.4.2;
23

Tests for equality where one or other variable may be a datum control sequence. If both are

numeric datum control sequences they will be compared numerically, otherwise they will be compared by their string values.

```
\datatool_if_value_eq:NnTF <tl var> {<tl>} {<true>} {<false>}
datatool-base v3.0+
```

§2.2.4.2;
23

Tests for equality where *<tl var>* may be a datum control sequence and *<tl>* may be a datum item. If both are a numeric datum control sequence and datum item they will be compared numerically, otherwise they will be compared by their string values.

```
\datatool_if_value_eq:nNTF {<tl>} <tl var> {<true>} {<false>}
datatool-base v3.0+
```

§2.2.4.2;
23

Tests for equality where *<tl var>* may be a datum control sequence and *<tl>* may be a datum item. If both are a numeric datum item and datum control sequence they will be compared numerically, otherwise they will be compared by their string values.

```
\datatool_if_value_eq:nnTF {<tl1>} {<tl2>} {<true>} {<false>}
datatool-base v3.0+
```

§2.2.4.2;
23

Tests for equality where one or other or the token lists may be a datum item. If both are numeric datum items they will be compared numerically, otherwise they will be compared by their string values.

```
\datatool_locale_define_keys:nn {<module>} {<key=value list>}
datatool-base v3.0+
```

Define keys for localisation support.

```
\datatool_map_keys_function:nN {<db-name>} {function}
datatool v3.0+
```

§3.16.2;
375

Maps over all the columns in the given database, applying the function to each set of column meta data. The function should have four arguments {<key>} {<col-idx>} {<type>} {<header>} where *<key>* is the column key, *<col-idx>* is the column index, *<type>* is the data type (−1 for unknown) and *<header>* is the column header.

\datatool_map_keys_inline:nn { *<db-name>* } { *<def>* } datatool v3.0+

§3.16.2;
375

Maps over all the columns in the given database, applying the inline function to each set of column meta data. Within *<def>*, #1 references the column key, #2 references the column index, #3 references the data type (−1 for unknown) and #4 references the column header.

\datatool_max_known_type: datatool-base v3.0+

§2.2.4; 21

Expands to the current maximum known data type identifier.

\datatool_measure:NNNn *<wd-dim>* *<ht-dim>* *<dp-dim>* { *<text>* }
datatool-base v3.0+

§2.8.3;
147

Measures the width, height and depth of *<text>* but first implements `\l_datatool_measure_hook_tl` to locally disable problematic commands.

\datatool_measure_depth:Nn *<dim>* { *<text>* } datatool-base v3.0+

§2.8.3;
147

A shortcut that uses `\settodepth` to measure the depth of *<text>* but first implements `\l_datatool_measure_hook_tl` to locally disable problematic commands.

\datatool_measure_height:Nn *<dim>* { *<text>* } datatool-base v3.0+

§2.8.3;
147

A shortcut that uses `\settoheight` to measure the height of *<text>* but first implements `\l_datatool_measure_hook_tl` to locally disable problematic commands.

\datatool_measure_ht_plus_dp:Nn *<dim>* { *<text>* } datatool-base v3.0+

§2.8.3;
147

Measures the combined height and depth of *<text>* but first implements `\l_datatool_measure_hook_tl` to locally disable problematic commands.

\datatool_measure_width:Nn *<dim>* { *<text>* } datatool-base v3.0+

§2.8.3;
147

A shortcut that uses `\settowidth` to measure the width of *<text>* but first implements `\l_datatool_measure_hook_tl` to locally disable problematic commands.

`\datatool_pad_trailing_zeros:Nn` $\langle tl-var \rangle$ $\{ \langle n \rangle \}$ datatool-base v3.0+

§2.8.3;
146

Pads $\langle tl-var \rangle$ with 0s to ensure that there are a minimum of $\langle n \rangle$ digits after the decimal point. The $\langle token list \rangle$ should contain a plain number (decimal or integer) before use. If the number in $\langle tl-var \rangle$ was originally an integer, it will become a decimal with $\langle n \rangle$ 0s after the decimal point. This command does nothing if $\langle n \rangle$ is not greater than zero.

`\datatoolparen` $\{ \langle text \rangle \}$

datatool-base v3.0+

§2.9.5.3;
167

Typesets parenthetical content, this command expands to `\space` ($\langle text \rangle$) normally but expands to nothing within `\dtlwordindexcompare`, `\dtlletterindexcompare` and `\DTLsortwordlist`.

`\datatoolparenstart`

datatool-base v2.13+

§2.9.5.3;
167

Designed to indicate the start of parenthetical content, this command expands to `\space` normally but expands to the character 0x1F inside `\dtlwordindexcompare`, `\dtlletterindexcompare` and `\DTLsortwordlist`.

`\datatoolpersoncomma`

datatool-base v2.13+

§2.9.5.3;
166

Designed to indicate word inversion for a person, this command expands to `,` `\space` normally but expands to the character 0x1C inside `\dtlwordindexcompare`, `\dtlletterindexcompare` and `\DTLsortwordlist`.

`\datatoolplacecomma`

datatool-base v2.13+

§2.9.5.3;
166

Designed to indicate a comma to clarify a place, this command expands to `,` `\space` normally but expands to the character 0x1D inside `\dtlwordindexcompare`, `\dtlletterindexcompare` and `\DTLsortwordlist`.

`\datatool_post_process_lettergroup:N` $\langle tl var \rangle$

datatool v3.0+

§3.14.1.1;
326

A hook that may be used to post-process the letter group token variable after sorting.

\datatool_prefix_adjust_sign:nnn {*<symbol>*} {*<sep>*}
 {*<value>*} datatool-base v3.0+

§2.6; 128

Designed for use in `\dtlcurrprefixfmt` this tests if *<value>* starts with a plus (+) or minus (-) and, if so, shifts the sign in front of the symbol and encapsulates the sign with `\datatool_adjust_sign_fmt:n`.

\datatool<Region>SetCurrency datatool-<Region>.ldf

§2.3.2; 45

Hook provided by region files to set the region's currency. This hook should check the boolean `\l_datatool_region_set_currency_bool` and only set the currency if true.

\datatool<Region>SetNumberChars datatool-<Region>.ldf

§2.3.2; 45

Hook provided by region files to set the region's number group and decimal characters. This hook should check the boolean `\l_datatool_region_set_numberchars_bool` and only set the number group character and decimal character if true.

\datatool<Region>symbolprefix{*<tag>*} datatool-<Region>.ldf

§2.3.2; 44

Provided by region files that support a prefix for the currency symbol (not all do). If supported, the region should provide an option called `currency-symbol-prefix` which can show or hide the prefix. The prefix, if enabled, is formatted with `\datatoolcurrencysymbolprefixfmt`.

\datatool_register_regional_currency_code:nn
 {*<region-code>*} {*<currency-code>*} datatool-base v3.0+

§2.3.2; 44

Region files should use this command to register the currency code defined by that region.

\datatool_set_apos_group_decimal_char:n{*<decimal char>*}
variant: V datatool-base v3.0+

§2.3.2; 43

Similar to `\datatool_set_numberchars:nn` but uses an apostrophe character for the number group character when formatting, and allows either straight apostrophe (U+27) or curly apostrophe (U+2019) as the number group character when parsing. The decimal character for both formatting and parsing is set to *<decimal char>*.

\datatoolSetCurrencySort

datatool–base v3.0+

§2.9.5.3;
168

Locally redefines common currency commands (for sorting).

\datatool_set_currency_symbol:nn {*⟨ISO⟩*} {*⟨symbol⟩*}

variants: nV ne datatool–base v3.0+

§2.6; 125

Set the symbol to *⟨symbol⟩* for the currency identified by *⟨ISO⟩*, which should already have been defined.

\datatool_set_fp:Nn *⟨fp-var⟩* {*⟨value⟩*}

datatool–base v3.0+

§2.2.4.3;
26

Sets the floating point variable *⟨fp-var⟩* to the floating point number obtained from the given *⟨value⟩*, which may be a datum control sequence or a datum item or in a locale-sensitive format that requires parsing.

\datatool_set_numberchars:nn {*⟨number group char⟩*} {*⟨decimal char⟩*}

variants: nV Vn VV datatool–base v3.0+

§2.3.2; 42

Sets the current number group character and decimal character.

\datatool_set_numberchars:nnnn {*⟨format number group char⟩*} {*⟨format decimal char⟩*} {*⟨parse number group char⟩*} {*⟨parse decimal char⟩*}

variants: VVVV eeee datatool–base v3.0+

§2.3.2; 42

Sets the current number group character and decimal character for formatting and parsing.

\datatool_set_numberchars_regex:nnnn {*⟨format number group char⟩*} {*⟨format decimal char⟩*} {*⟨parse number group regex⟩*} {*⟨parse decimal regex⟩*}

variants: VVnn Vnnn nVnn datatool–base v3.0+

§2.3.2; 42

Sets the current number group character and decimal character for formatting to *⟨format number group char⟩* and *⟨format decimal char⟩* and sets number group character and decimal character regular expressions used by the parser to *⟨parse number group regex⟩* and *⟨parse decimal regex⟩*.

\datatool_set_numberchars_regex_tl:nnnn { *format number group char* } { *format decimal char* } { *parse number group regex* } { *parse decimal char* }
 variants: VVnn Vnnn nVnn nVnV nnnV datatool-base v3.0+

§2.3.2; 43

Sets the current number group character and decimal character for formatting to *format number group char* and *format decimal char* and sets number group character regular expressions used by the parser to *parse number group regex* and the decimal character to *parse decimal char*.

\datatool_set_numberchars_tl_regex:nnnn { *format number group char* } { *format decimal char* } { *parse number group char* } { *parse decimal regex* }
 variants: VVnn Vnnn nVnn VnVn nnVn datatool-base v3.0+

§2.3.2; 43

Sets the current number group character and decimal character for formatting to *format number group char* and *format decimal char*, and sets the number group character to *parse number group char* and the decimal character regular expressions used by the parser to *parse decimal regex*.

\datatool_set_thinspace_group_decimal_char:n { *decimal char* }
 variant: V datatool-base v3.0+

§2.3.2; 43

Similar to `\datatool_set_numberchars:nn` but uses `\,` (thin space) for the number group character when formatting, and allows `\,` or a normal space or the Unicode character U+2009 (thin space) as the number group character when parsing. The decimal character for both formatting and parsing is set to *decimal char*.

\datatool_set_underscore_group_decimal_char:n { *decimal char* }
 variant: V datatool-base v3.0+

§2.3.2; 43

Similar to `\datatool_set_numberchars:nn` but uses `\_` for the number group character when formatting, and allows `\_` or the underscore character as the number group character when parsing. The decimal character for both formatting and parsing is set to *decimal char*.

\datatool_sort_preprocess:Nn *tl-var* { *text* } datatool-base v3.0+

§2.9.5;
162

Expand and apply the current sort hooks to *text* and store the result in *tl-var*.

\datatool_sort_preprocess:NnN $\langle tl-var \rangle$ { $\langle text \rangle$ } $\langle bool-var \rangle$
 datatool-base v3.2+

§2.9.5;
162

Expand and apply the current sort hooks to $\langle text \rangle$ and store the result in $\langle tl-var \rangle$. If $\langle bool-var \rangle$ is true, the sort value will be converted to lowercase.

\datatoolsubjectcomma

datatool-base v2.13+

§2.9.5.3;
167

Designed to indicate heading inversion, this command expands to `, \space` normally but expands to the character 0x1E inside `\dtlwordindexcompare`, `\dtlletterindexcompare` and `\DTLsortwordlist`.

\datatool_suffix_adjust_sign:nnn { $\langle symbol \rangle$ } { $\langle sep \rangle$ }
 { $\langle value \rangle$ }

datatool-base v3.0+

§2.6; 128

Designed for use in `\dtlcurrsuffixfmt` this tests if $\langle value \rangle$ starts with a plus (+) or minus (-) and, if so, encapsulates it with `\datatool_adjust_sign_fmt:n`. The separator and symbol are placed after the value.

\DataToolTimeFmt { $\langle hour \rangle$ } { $\langle minute \rangle$ } { $\langle second \rangle$ }

datatool-base v3.0+

§2.7; 138

Inserted by `parse=auto-format` to format a time. The arguments should all be integers, but $\langle second \rangle$ may be empty. By default, simply uses `\DTLCurrentLocaleFormatTime`.

\DataToolTimeStampFmtSep

datatool-base v3.0+

§2.7; 137

Used by the default timestamp formats to separate the date and time if both provided. By default, simply expands to `\DTLCurrentLocaleTimeStampFmtSep`.

\DataToolTimeStampNoZoneFmt { $\langle year \rangle$ } { $\langle month \rangle$ } { $\langle day \rangle$ } { $\langle dow \rangle$ }
 { $\langle hour \rangle$ } { $\langle minute \rangle$ } { $\langle second \rangle$ }

datatool-base v3.0+

§2.7; 137

Used by `\DataToolDateTimeFmt` if { $\langle date-specs \rangle$ } and { $\langle time-specs \rangle$ } are both not empty, but $\langle offset-specs \rangle$ is empty. By default, simply uses `\DTLCurrentLocaleFormatTimeStampNoZone`.

\DataToolTimeStampWithZoneFmt { *<year>* } { *<month>* } { *<day>* } { *<dow>* } { *<hour>* } { *<minute>* } { *<second>* } { *<tzh>* } { *<tzm>* }

datatool-base v3.0+

§2.7; 137

Used by `\DataToolDateTimeFmt` if { *<date-specs>* } and { *<time-specs>* } and *<offset-specs>* are all not empty. By default, simply uses `\DTLCurrentLocaleFormatTimeStampWithZone`.

\DataToolTimeZoneFmt { *<tzh>* } { *<tzm>* }

datatool-base v3.0+

§2.7; 138

Inserted by `parse=auto-format` to format a time zone offset. The arguments should all be integers. By default, simply uses `\DTLCurrentLocaleFormatTimeZone`.

\DBIBCitekey

databib

§7.8; 550

Placeholder command set by commands such as `\DTLformatthisbibentry` and `\DTLforeachbibentry` that should expand to the cite key identifying the required row of the database.

\DBIBentrytype

databib

§7.8; 550

Placeholder command set by commands such as `\DTLformatthisbibentry` and `\DTLforeachbibentry` that should expand to the entry type obtained from the `EntryType` field of the current row.

\DBIBname

databib

Placeholder command set by commands such as `\DTLformatthisbibentry` and `\DTLforeachbibentry` that should expand to database name supplied in the *<db-name>* argument of those commands.

\Description

datagidx v2.15+

§8.9.1;
619

Placeholder in `\printterms` that expands to the current entry's `Description` value.

\DTLabs {*cs*} {*num*}

datatool-base

§2.5.2;
110

Calculates the absolute value of the formatted number *num* and stores the result as a formatted number in the control sequence *cs*.

\dtlabs {*cs*} {*num*}

datatool-base

§2.5.1;
100

Defines the control sequence *cs* to the absolute value of the number *num*, where the number is a plain number.

\DTLaction [*settings*] {*action*}

datatool v3.0+

§3.3

Performs a command corresponding to *action*.

\DTLadd {*cs*} {*num1*} {*num2*}

datatool-base

§2.5.2;
108

Calculates *num1* + *num2* and stores the result in the control sequence *cs*, where the numbers are formatted numbers. The result will be a formatted number.

\dtladd {*cs*} {*num1*} {*num2*}

datatool-base

§2.5.1; 98

Calculates *num1* + *num2* and stores the result in the control sequence *cs*, where the numbers are plain numbers.

\dtladdalign *align-token-tl* {*type*} {*col-num*} {*max-cols*} datatool v3.0+

§3.7.2;
256

Used by `\DTLdisplaydb` and `\DTLdisplaylongdb` to add the appropriate column alignment specifier to the *align-token-tl* token list variable. Not used if the `align-specs` option is set.

\DTLaddall {*cs*} {*num list*}

datatool-base

§2.5.2;
108

Adds all the formatted numbers in the comma-separated *num list* and stores the result as a formatted number in *cs*.

\dtladdall { *<cs>* } { *<num list>* }

datatool-base v3.0+

§2.5.1; 99

Adds all the numbers in the comma-separated list *<num list>* and stores the result in the control sequence *<cs>*, where the numbers are plain numbers.

\DTLaddcolumn { *<db-name>* } { *<col key>* }

modifier: * datatool v2.11+

§3.6; 240

Adds a column with the label *<col key>* to the column meta data for the database identified by *<db-name>*. The starred version doesn't check if the database exists.

\DTLaddcolumnwithheader { *<db-name>* } { *<col key>* } { *<header>* }

modifier: * datatool v3.0+

§3.6; 240

Adds a column with the label *<col key>* and the given *<header>* to the column meta data for the database identified by *<db-name>*. The starred version doesn't check if the database exists.

\dtladdheaderalign *<align-token-tl>* { *<type>* } { *<col-num>* } { *<max-cols>* }

datatool v3.0+

§3.7.1.3;
247

Used by `\DTLdisplaydb` and `\DTLdisplaylongdb` to add the appropriate column alignment specifier for the header row to the *<align-token-tl>* token list variable. Not used if the `header-row` option is set.

\DTLaddtoplotlegend { *<marker>* } { *<line style>* } { *<text>* }

dataplot

§6.3.2;
512

Adds an entry to the legend.

\dtlaftercols

initial: empty datatool v2.0+

§3.7.2;
255

Expands to content to be inserted by `\DTLdisplaydb` and `\DTLdisplaylongdb` at the end of the alignment specification. Ignored if the `align-specs` option is set.

\DTLafterinitialbeforehyphen

initial: . datatool-base

§2.8.2;
142

Used by `\DTLinitials` and `\DTLstoreinitials` after an initial before a hyphen.

\DTLafterinitials

initial: . datatool-base

§2.8.2;
141

Used by \DTLinitials and \DTLstoreinitials after the initials.

\dtlafterrow

datatool v2.0+

§3.16.1;
371

Token register used to store the content after the current row when \dtlcurrentrow is set.

\DTLandlast

databib

§7.7.1;
545

Used between the final pair of names where there are more than two names in the list.

\DTLandname

(language-sensitive)

initial: varies datatool-base v2.28+

§2.9.2;
152

Expands to \andname if that command was defined when datatool-base was loaded otherwise expands to \& or the applicable localisation text.

\DTLandnotlast

databib

§7.7.1;
546

Used between all except the final pair of names where there are more than two names in the list.

\DTLangLatnLocaleGetGroupString{*<actual>*}{*<sort value>*}{*<cs>*}

datatool-ang-Latn.ldf

§2.3.5; 57

Anglo-Saxon (Latin Script) group string handler.

\DTLaposinitialpunc{*<H>*}{*<T>*}{*<punc>*}

datatool-base v3.0+

§2.8.2;
141

Used by \DTLstoreinitials to markup initials from words with an apostrophe.

\dtlappendentrytocurrentrow{*<col key>*}{*<value>*}

datatool v2.10+

§3.16.1;
374

Appends an entry with the given *<value>* to \dtlcurrentrow for the column identified by *<col key>*. The row must not already contain an element in the given column. The column data is updated according to *<value>* (the `global` option is checked).

\DTLappendtorow{ $\langle col\text{-}key \rangle$ }{ $\langle value \rangle$ }

datatool

§3.8.2.1;
297

May only be used within the unstarred `\DTLforeach`, this command will append an entry to the current row with the given value for the column identified by $\langle col\text{-}key \rangle$.

\DTLassign{ $\langle db\text{-}name \rangle$ }{ $\langle row\ idx \rangle$ }{ $\langle assign\text{-}list \rangle$ }

datatool v2.10+

§3.16.1;
373

Selects the current row according to its row index $\langle row\ idx \rangle$ (using `\dtlgetrow`) and globally defines the commands in the assignment list $\langle assign\text{-}list \rangle$ for the row identified by the row index $\langle row\ idx \rangle$ in the database identified by $\langle db\text{-}name \rangle$. This command is not affected by the `global` option. If you prefer a local assignment, use `\dtlgetrow` and `\DTLassignfromcurrentrow` instead.

\DTLassignfirstmatch{ $\langle db\text{-}name \rangle$ }{ $\langle col\ key \rangle$ }{ $\langle value \rangle$ }{ $\langle assign\text{-}list \rangle$ }

datatool v2.10+

§3.16; 370

Globally defines the commands in the assignment list $\langle assign\text{-}list \rangle$ for the first row in the database identified by $\langle db\text{-}name \rangle$ where the column identified by $\langle col\ key \rangle$ matches $\langle value \rangle$. This command is not affected by the `global` option.

\DTLassignfromcurrentrow{ $\langle assign\text{-}list \rangle$ }

datatool v3.0+

§3.16.1;
372

Assigns elements in the current row (`\dtlcurrentrow`) to the provided placeholder commands in the $\langle assign\text{-}list \rangle$.

\DTLassignlettergroup{ $\langle actual \rangle$ }{ $\langle sort\ value \rangle$ }{ $\langle cs \rangle$ }

datatool-base v3.0+

§2.9.5;
157

Used by `\DTLsortwordlist` and `\DTLsortdata` to assign the letter group based on the actual or sort value.

\DTLbaratbegintikz

databar

§5.4.4;
449

Hook used by `\DTLbarchart` and `\DTLmultibarchart` at the start of the `tikzpicture` environment.

\DTLbaratendtikz

databar

§5.4.4;
449

Hook used by `\DTLbarchart` and `\DTLmultibarchart` at the end of the `tikzpicture` environment.

\DTLbarchart [*⟨condition⟩*] { *⟨settings-list⟩* } { *⟨db-name⟩* } { *⟨assign-list⟩* } databar

§5; 400

Draws a bar chart using data from the database identified by *⟨db-name⟩*. The *⟨settings-list⟩* must include the `variable` setting to identify the placeholder command *⟨assign-list⟩* that will be assigned to the numeric values for the chart.

\DTLbarchartlength

initial: 3in databar

§5.4.1;
442

Length register that governs the bar chart length.

\DTLbarchartwidth

databar

§5.4.4;
449

For use in the bar chart hooks, this will expand to the length of the current bar chart's *x*-axis (in terms of bar width) as a dimensionless number.

\DTLbardisplayYticklabel { *⟨text⟩* }

databar v3.0+

§5.4.2;
445

Used to format *y* tick labels.

\DTLbargroupindex

databar v3.0+

§5.4.2;
443

Placeholder command that expands to the current group bar index, for use in lower or upper labels or hooks (expands to 0 in `\DTLbarchart`).

\DTLbargrouplabelalign

initial: \DTLbarXlabelalign

databar v3.0+

§5.4.2;
445

The expansion text of this command should be the `\pgftext` alignment options for the *x*-axis bar group labels.

\DTLbargroupwidth

databar

§5.4.4;
449

For use in the bar chart hooks, this will expand to the width of the bar groups in the current bar chart (in terms of bar width) as a dimensionless number.

\DTLbarindex

databar v3.0+

§5.4.2;
443

Placeholder command that expands to the current bar index, for use in lower or upper labels or hooks. For `\DTLmultibarchart`, this is reset at the start of each group.

\DTLbarlabeloffset

initial: 10pt databar

§5.4.2;
443

Length register that governs the distance from the bar to the bar label.

\DTLbarmax

initial: empty databar

§5.4.1;
442

The maximum value of the y axis or empty if the maximum value in the data should be used.

\DTLbaroutlinecolor

initial: black databar

§5.4.3;
446

The expansion text of this command should be the colour of the bar outline or empty if no outline should be drawn.

\DTLbaroutlinewidth

initial: 0pt databar

§5.4.3;
446

Length register that governs the bar outline width.

\DTLbarsetupperlabelalign [*<-ve align>*] {*<+ve align>*}

databar v3.0+

§5.4.2;
445

Used by the `upper-label-align=to` to redefine `\DTLbarXupperlabelalign` and optionally redefine `\DTLbarXnegupperlabelalign`.

\DTLBarStyle

initial: empty databar v3.0+

§5.4.1;
442

This command expands within the optional argument of `\path` when drawing or filling a bar. It may be redefined to apply additional styling.

\DTLbartotalvariables

databar v3.0+

§5.4.4;
450

For use in the bar chart hooks with `\DTLmultibarchart`, this will expand to the total number of variables (that is, the number of bars per group) for the current bar chart.

\DTLbarvalue

databar v3.0+

§5.4.2;
443

Placeholder command that may be used in lower or upper labels to show the formatted number (which will be rounded according to the `round` setting) for the bar value.

\DTLbarvariable

databar

§5.4.2;
443

Placeholder command that may be used in lower or upper labels to show the actual value.

\DTLbarwidth*initial:* 1cm databar§5.4.2;
443

Length register that governs the bar width.

\DTLBarXAxisStyle*initial:* — databar§5.4.1;
442

The expansion text of this command should be the tikz line style specifier for the *x*-axis.

\DTLbarXlabelalign

databar

§5.4.2;
443

The expansion text of this command should be the `\pgftext` alignment options for the *x*-axis bar labels.

\DTLbarXneglabelalign

databar v3.0+

§5.4.2;
444

The expansion text of this command should be the `\pgftext` alignment options for the lower labels of bars with negative values.

\DTLbarXnegupperlabelalign

databar v3.0+

§5.4.2;
444

The expansion text of this command should be the `\pgftext` alignment options for the upper labels of bars with negative values.

\DTLbarXupperlabelalign

databar v3.0+

§5.4.2;
444

The expansion text of this command should be the `\pgftext` alignment options for the upper labels of bars with positive values.

\DTLBarYAxisStyle

initial: – databar

§5.4.1;
442

The expansion text of this command should be the `tikz` line style specifier for the *y*-axis.

\DTLbarYticklabelalign

initial: varies databar

§5.4.2;
445

The expansion text of this command should be the `\pgftext` alignment options for the *y*-axis tick labels.

\dtlbeforecols

initial: empty datatool v2.0+

§3.7.2;
255

Expands to content to be inserted by `\DTLdisplaydb` and `\DTLdisplaylongdb` at the start of the alignment specification. Ignored if the `align-specs` option is set.

\dtlbeforerow

datatool v2.0+

§3.16.1;
371

Token register used to store the content before the current row when `\dtlcurrentrow` is set.

\dtlbetweencols

initial: empty datatool v2.0+

§3.7.2;
255

Expands to content to be inserted by `\DTLdisplaydb` and `\DTLdisplaylongdb` between the column alignment specifiers. Ignored if the `align-specs` option is set.

\DTLbetweeninitials

initial: . datatool-base

§2.8.2;
141

Used by `\DTLinitials` and `\DTLstoreinitials` between initials.

\DTLbibaccessedname

initial: accessed databib v3.0+

§7.7.1;
549

Locale-sensitive command used by `\DTLbiburldate`.

\DTLbibdatefield{*<field name>*}

databib

§7.8; 551

As `\DTLbibfield` but is specifically used for `Date` and `UrlDate`.

\DTLBIBdbname

databib

§7.4; 537

Placeholder command set by `\DTLloadbibl` that should expand to the name of the databib database containing the bibliography data.

\DTLbibdoi{*<doi>*}

databib v3.0+

§7.7.1;
548

Used to format the `DOI` field (which should already be detokenized).

\DTLbibdoihome

initial: `https://doi.org/` databib v3.0+

§7.7.1;
548

Used by `\DTLbibdoi` in the formation of the hyperlink, if applicable.

\DTLbibdoitag

databib v3.0+

§7.7.1;
548

Expands to the textual tag used at the start of `\DTLbibdoi`.

\DTLbibeprints{*<url>*}{*<eprint-type>*}

databib v3.0+

§7.7.1;
548

Used to format the `Eprints` field (which should already be detokenized).

\DTLbibfield{*<field name>*}

databib

§7.8; 550

For use in `\DTLforeachbibentry`, this expands to the value of the column identified by *<field name>* in the current iteration. Expands to nothing if the field isn't set or doesn't exist.

\DTLbibfieldcontains{*<field label>*}{*<value>*}

databib

§7.6; 541

For use in `\ifthenelse`, this conditional tests if the value of the named field for the current row contains *<value>*.

\DTLbibfieldexists { *⟨field label⟩* }

databib

§7.6; 540

For use in `\ifthenelse`, this conditional tests if the named field exists (the column is defined and the value is not null) for the current row.

\DTLbibfieldiseq { *⟨field label⟩* } { *⟨value⟩* }

databib

§7.6; 541

For use in `\ifthenelse`, this conditional tests if the value of the named field for the current row is equal to *⟨value⟩*.

\DTLbibfieldisge { *⟨field label⟩* } { *⟨value⟩* }

databib

§7.6; 542

For use in `\ifthenelse`, this conditional tests if the value of the named field for the current row is lexicographically greater than or equal to *⟨value⟩*.

\DTLbibfieldisgt { *⟨field label⟩* } { *⟨value⟩* }

databib

§7.6; 542

For use in `\ifthenelse`, this conditional tests if the value of the named field for the current row is lexicographically greater than *⟨value⟩*.

\DTLbibfieldisle { *⟨field label⟩* } { *⟨value⟩* }

databib

§7.6; 541

For use in `\ifthenelse`, this conditional tests if the value of the named field for the current row is lexicographically less than or equal to *⟨value⟩*.

\DTLbibfieldislt { *⟨field label⟩* } { *⟨value⟩* }

databib

§7.6; 541

For use in `\ifthenelse`, this conditional tests if the value of the named field for the current row is lexicographically less than *⟨value⟩*.

\DTLbibfieldlet { *⟨cs⟩* } { *⟨field name⟩* }

databib

§7.8; 551

For use in `\DTLforeachbibentry`, this sets the command (token list variable) *⟨cs⟩* to the value of the column identified by *⟨field name⟩* in the current iteration.

\DTLbibformatdigital

databib v3.0+

§7.7.1;
547

Formats the information provided in the [PubMed](#), [DOI](#), [Url](#) and [Eprints](#) fields.

\DTLbibitem

databib

§7.7.1;
546

Used within `\DTLbibliography` at the start of each iteration.

\DTLbibliography [*⟨condition⟩*] {*⟨db-name⟩*}

databib

§7.6; 540

Shortcut command that starts and ends `DTLthebibliography` and, within the body of that environment, iterates over the given database with `\DTLforeachbibentry*`, formatting each item according to the current style (as set by `\DTLbibliographystyle`). The supplied arguments are provided to both the `DTLthebibliography` environment and `\DTLforeachbibentry`.

\DTLbibliographystyle{*⟨name⟩*}

databib

§7.7; 543

Sets the bibliography style to *⟨name⟩*, which may be one of: `plain`, `abbrv` or `alpha`.

\DTLbibpubmed{*⟨pmid⟩*}

databib v3.0+

§7.7.1;
547

Used to format the [PubMed](#) field.

\DTLbibpubmedhome

initial: <https://pubmed.ncbi.nlm.nih.gov/> databib v3.0+

§7.7.1;
548

Used by `\DTLbibpubmed` in the formation of the hyperlink, if applicable.

\DTLbibpubmedtag

databib v3.0+

§7.7.1;
547

Expands to the textual tag used at the start of `\DTLbibpubmed`.

\DTLbibsorthencap { *<value>* } { *<col-idx>* } { *<db-name>* }

databib v3.0+

§7.5; 539

Provided for use with the [encap](#) sort option to convert the comma-separated name lists in the `Author` and `Editor` fields to a format better suited for a sort value. Each element in the list will be encapsulated with `\DTLbibsorthname` and separated by `\DTLbibsorthname-sep`.

\DTLbibsorthname { *<von>* } { *<surname>* } { *<jr>* } { *<forename>* }

databib v3.0+

§7.5; 539

Used by `\DTLbibsorthencap`, this command should expand to the format of the name best suited for sorting.

\DTLbibsorthnamesep

initial: `\datatoolasciend`

databib v3.0+

§7.5; 539

Separator used by `\DTLbibsorthencap`.

\DTLbiburl { *<url>* }

databib v3.0+

§7.7.1;
548

Used to format the `Url` field (which should already be detokenized).

\DTLbiburldate { *<date>* }

databib v3.0+

§7.7.1;
549

Used to format the `UrlDate` field.

\dtlbreak

datatool-base

§3.8.2;
296

When used in the loop body of `\DTLforeach`, this command will cause the loop to terminate at the end of the current iteration.

\DTLcite [*<text>*] { *<mbib>* } { *<label-list>* }

databib

§7.9; 553

Similar to `\cite` [*<text>*] { *<label-list>* } except that the cite information is written to the auxiliary file associated with the multi-bib *<mbib>* (as identified in `\DTLmultibibs`). The cross-referencing label is constructed from both *<mbib>* and the label to allow the same citation to appear in multiple bibliographies.

\DTLclearbarcolors

databar v3.0+

§5.4.3;
447

Clears the bar colour list.

\DTLcleardb { *db-name* }

datatool v2.03+

§3.5; 236

Clears the database with the label *db-name*. That is, the database becomes empty but is still defined.

\DTLclearnegbarcolors

databar v3.0+

§5.4.3;
447

Clears the negative bar colour list.

\DTLclip { *cs* } { *num* }

datatool-base

§2.5.2;
112

Converts the formatted number to a plain number, clips it using `\dtlclip` and stores the result as a formatted number in the control sequence *cs*.

\dtlclip { *cs* } { *num* }

datatool-base

§2.5.1;
100

Removes redundant trailing zeros from *num* and stores the result in the control sequence *cs*, where the number is a plain number.

\dtlcol

datatool

§3.8.2;
297

Defined by `\DTLforeachkeyinrow` to the current column index.

\DTLcolumncount { *db-name* }

datatool

§3.6; 237

Expands to the column count of the database with the label *db-name*.

\dtlcolumnheader { *align* } { *text* }

datatool v3.0+

§3.7.1.3;
247

Used by `\DTLdisplaydb` and `\DTLdisplaylongdb` to align the column headers.

\dtlcolumnindex { $\langle db-name \rangle$ } { $\langle col key \rangle$ }

datatool v2.0+

§3.6; 238

Expands to the column index for the column with the label $\langle col key \rangle$ in the database identified by $\langle db-name \rangle$. Expands to 0 if the database or column don't exist.

\dtlcolumnnum

datatool v2.0+

A count register used by various commands to keep track of a column number, which may be used by hooks.

\dtlcompare { $\langle count-reg \rangle$ } { $\langle string1 \rangle$ } { $\langle string2 \rangle$ }

datatool-base

§2.9.5.1;
163

Compares $\langle string1 \rangle$ and $\langle string2 \rangle$ and stores the result in the count register $\langle count-reg \rangle$, where 0 indicates the strings are the same, -1 means that $\langle string1 \rangle$ comes before (less than) $\langle string2 \rangle$ and 1 means that $\langle string1 \rangle$ comes after (greater than) $\langle string2 \rangle$ (case-sensitive).

\DTLcomputebounds [$\langle condition \rangle$] { $\langle db-list \rangle$ } { $\langle x-key \rangle$ } { $\langle y-key \rangle$ } { $\langle minX cmd \rangle$ } { $\langle minY cmd \rangle$ } { $\langle maxX cmd \rangle$ } { $\langle maxY cmd \rangle$ }

datatool

§3.13; 324

Computes the maximum and minimum x and y values over all databases listed in $\langle db-list \rangle$ for the columns identified by $\langle x-key \rangle$ and $\langle y-key \rangle$.

\DTLcomputewidestbibentry { $\langle condition \rangle$ } { $\langle db-name \rangle$ } { $\langle bib-label \rangle$ } { $\langle cs \rangle$ }

datatool

§7.8; 552

Computes the widest bibliography entry over all entries satisfying the given condition for the given database, where the bibliography label is formatted according to $\langle bib label \rangle$ and stores the result in the command (token list variable) $\langle cs \rangle$.

\DTLconverttodecimal { $\langle num \rangle$ } { $\langle cs \rangle$ }

datatool-base

§2.2.2; 15

Converts the formatted number $\langle num \rangle$ to a plain number and stores the result in the control sequence $\langle cs \rangle$.

\DTLcurr{*ISO*}

datatool–base v3.0+

§2.6; 126

Expands to `\DTLcurrISO` if defined or to `ISO` otherwise.

\DTLcurrChar{*ISO*}

datatool–base v3.0+

§2.6; 126

Expands to the currency character associated with the currency identified by *ISO*, or nothing if not defined.

\DTLcurrCodeOrSymOrChar{*ISO*}{*symbol*}{*character*}

datatool–base v3.0+

§2.6; 130

Expands to one of its arguments (*symbol* by default).

\dtlcurrdefaultfmt{*symbol*}{*value*}

datatool–base v3.0+

§2.6; 128

Default currency format (initialised to use `\dtlcurrprefixfmt`).

\DTLcurrency{*value*}

datatool–base v3.0+

§2.6; 129

Formats *value* as a currency using `\DTLfmtcurrency` with the default currency symbol.

\dtlcurrencyalign

initial: `r` datatool v2.0+

§3.7.2;
254

Expands to the column alignment specifier used by `\DTLdisplaydb` and `\DTLdisplaylongdb` for columns with the currency data type. Ignored if the `align-specs` option is set.

\DTLCurrencyCode

datatool–base v3.0+

§2.6; 127

Expands to the ISO code associated with the default currency.

\dtlcurrencyformat{*text*}

datatool v2.0+

§3.7.2;
253

Used by `\DTLdisplaydb` and `\DTLdisplaylongdb` to format entries in currency columns. Defined to use `\dtlnumericformat` by default.

\dtlcurrencygroup { *<sym>* } { *<num>* }

datatool–base v3.0+

§2.3.3; 53

Used by \DTLassignlettergroup to identify currency groups.

\DTLCurrencySymbol

datatool–base v3.0+

§2.6; 127

Expands to the internal command used to store the default currency symbol.

\DTLcurrencytype

datatool v2.0+

§3.6; 239

Expands to 3.

\DTLcurrentindex

datatool

§3.8.2;
296

Defined by \DTLforeach to expand to the current iteration index.

\DTLCurrentLocaleCurrencyDP

initial: 2 datatool–base v3.0+

§2.3.2; 46

The expansion text should be either empty (no rounding) or the number of decimal places that \DTLdecimaltocurrency should round to. This command should be redefined by the appropriate localisation hooks.

\DTLCurrentLocaleFormatDate { *<year>* } { *<month>* } { *<day>* } { *<dow>* }

datatool–base v3.0+

This command should be redefined by the applicable localisation hook. The arguments should all be integers, but *<dow>* may be empty.

\DTLCurrentLocaleFormatTime { *<hour>* } { *<minute>* } { *<second>* }

datatool–base v3.0+

This command should be redefined by the applicable localisation hook. The arguments should all be integers, but *<second>* may be empty.

\DTLCurrentLocaleFormatTimeStampNoZone { *year* } { *month* }
 { *day* } { *dow* } { *hour* } { *minute* } { *second* } datatool-base v3.0+

This command should be redefined by the applicable localisation hook. The arguments should all be integers, but *dow* and *second* may be empty.

\DTLCurrentLocaleFormatTimeStampWithZone { *year* } { *month* }
 { *day* } { *dow* } { *hour* } { *minute* } { *second* } { *tzh* } { *tzm* } datatool-base v3.0+

This command should be redefined by the applicable localisation hook. The arguments should all be integers, but *dow* and *second* may be empty.

\DTLCurrentLocaleFormatTimeZone { *tzh* } { *tzm* } datatool-base v3.0+

This command should be redefined by the applicable localisation hook. The arguments should all be integers.

\DTLCurrentLocaleGetGroupString { *actual* } { *sort value* } { *cs* }
 datatool-base v3.0+

§2.3.3; 49

Used by \DTLassignlettergroup to determine whether to obtain the letter group from the actual or sort value. Localisation support should redefine this as appropriate and may further process the value to ensure that the first character matches the appropriate group.

\DTLCurrentLocaleGetInitialLetter { *text* } { *cs* }
 datatool-base v3.0+

§2.3.3; 49

Used by \DTLGetInitialLetter and \DTLassignlettergroup.

\DTLCurrentLocaleTimeStampFmtSep *initial:* ☐ datatool-base v3.0+

This command should be redefined by the applicable localisation hook.

\DTLCurrentLocaleWordHandler { *cs* } datatool-base v3.0+

§2.3.3; 46

Current locale word handler used by string sorting.

\dtlcurrentrow

datatool v2.0+

§3.16.1;
371

Token register used to store the current row. Once set by commands such as `\DTLforeach` or `\dtlgetrow`, the row content can then be referenced or modified by commands like `\dtlreplaceentryincurrentrow`.

\DTLcurrEUR

datatool-base v3.0+

§2.6; 126

Euro currency EUR with symbol €.

\dtlcurrfmtsep

datatool-base v3.0+

§2.6; 128

Separator used by currency formats.

\dtlcurrfmtsymsep
(region-sensitive)

initial: empty datatool-base v3.0+

§2.6; 129

Used by `\dtlcurrfmtsep` when `\DTLcurrCodeOrSymOrChar` expands to either its second or third argument.

\DTLcurr*<ISO>*

datatool-base v3.0+

§2.6; 124

Defined by `\DTLdefcurrency`.

\dtlcurrprefixfmt { *<symbol>* } { *<value>* }

datatool-base v3.0+

§2.6; 128

Formats the currency with the symbol as a prefix using `\datatool_prefix_adjust_sign:nnn`.

\DTLcurrStr { *<ISO>* }

datatool-base v3.0+

§2.6; 126

Expands to the detokenised string associated with the currency identified by *<ISO>*, or nothing if not defined.

\dtlcurrsuffixfmt { *⟨symbol⟩* } { *⟨value⟩* }

datatool-base v3.0+

§2.6; 128

Formats the currency with the symbol as a suffix using `\datatool_suffix_adjust_sign:nnn`.

\DTLcurrSym{ *⟨ISO⟩* }

datatool-base v3.0+

§2.6; 126

Expands to the symbol associated with the currency identified by *⟨ISO⟩*, or nothing if not defined.

\DTLcurrXBT

datatool-base v3.0+

§2.6; 126

Bitcoin currency XBT with symbol ₿.

\DTLcurrXXX

datatool-base v3.0+

§2.6; 126

Generic currency XXX with symbol Ɑ.

\DTLcustombibitem { *⟨item code⟩* } { *⟨ref text⟩* } { *⟨cite key⟩* }

datbib v2.22+

§7.8; 552

As *⟨DTLbibitem⟩* but *⟨item code⟩* will be used instead of `\item`.

\DTLcustomlegend{ *⟨legend code⟩* }

dataplot v3.0+

§6.3.3;
514

Used to position the legend with `legend=custom`. This command should be redefined as applicable.

\DTLdatatypecurrencyname
(language-sensitive)

initial: currency datatool-base v3.0+

§2.2.3; 18

Expands to the name of the currency data type.

\DTLdatatypedatename
(language-sensitive)

initial: date datatool-base v3.0+

§2.2.3; 18

Expands to the name of the date data type.

\DTLdatatypedatetime *initial:* `date-time` datatool-base v3.0+
(language-sensitive)

§2.2.3; 18

Expands to the name of the datetime data type.

\DTLdatatypedecimal *initial:* `decimal` datatool-base v3.0+
(language-sensitive)

§2.2.3; 18

Expands to the name of the decimal data type.

\DTLdatatypeinteger *initial:* `integer` datatool-base v3.0+
(language-sensitive)

§2.2.3; 18

Expands to the name of the integer data type.

\DTLdatatypeinvalid *initial:* `invalid` datatool-base v3.0+
(language-sensitive)

§2.2.3; 18

Expands to the name of an invalid data type identifier.

\DTLdatatypestring *initial:* `string` datatool-base v3.0+
(language-sensitive)

§2.2.3; 18

Expands to the name of the string data type.

\DTLdatatypetime *initial:* `time` datatool-base v3.0+
(language-sensitive)

§2.2.3; 18

Expands to the name of the time data type.

\DTLdatatypeunset *initial:* `unset` datatool-base v3.0+
(language-sensitive)

§2.2.3; 18

Expands to the name of the unset data type.

\dtldategroup{*<value>*}

datatool-base v3.0+

Used by \DTLassignlettergroup to identify date groups.

\dtldatetimegroup{*<value>*}

datatool-base v3.0+

Used by \DTLassignlettergroup to identify datetime groups.

\DTLdatumcurrency{*<cs>*}

datatool-base v3.0+

§2.2.3; 17

Expands to the currency symbol that was parsed by \DTLparse or \DTLxparse.

\DTLdatumtype{*<cs>*}

datatool-base v3.0+

§2.2.3; 17

Expands to an integer representing the data type that was parsed by \DTLparse or \DTLxparse.

\DTLdatumvalue{*<cs>*}

datatool-base v3.0+

§2.2.3; 17

Expands to the numeric value that was parsed by \DTLparse or \DTLxparse (as a plain number).

\dtldbcolreconstruct{*<column-idx>*}{*<content>*}

datatool v3.0+

§3.15.1.3;
345

Only for use within the *<row code>* argument of \dtldbrowreconstruct.

\dtldbdatumreconstruct{*<string>*}{*<numeric>*}{*<currency>*}{*<type>*}

datatool v3.0+

§3.15.1.3;
345

Only for use within the *<content>* argument of \dtldbcolreconstruct.

\dtldbheaderreconstruct{*<column-idx>*}{*<key>*}{*<type>*}{*<header>*}

datatool v3.0+

§3.15.1.3;
344

Only for use within the *<header code>* argument of \DTLreconstructdatabase.

\dtldbname

datatool

§3.16.1;
372

Placeholder that expands to the current database name in certain contexts.

\DTLdbNewEntry { *col key* } { *value* }

datatool v3.0+

§3.15.1.2;
339

Does `\DTLnewdbentry { db-name } { col key } { value }` where *db-name* is the `default-name`.

\DTLdbNewRow

datatool v3.0+

§3.15.1.2;
339

Does `\DTLnewrow { db-name }` where *db-name* is the `default-name`.

\DTLdbProvideData

datatool v3.0+

§3.15.1.2;
339

Used at the start of a DTLTEX v3.0 and DBTEX v3.0 file to declare the database.

\dtldbconstructkeyindex { *key* } { *column-idx* }

datatool v3.0+

§3.15.1.3;
344

Only for use within the *key-index code* argument of `\DTLreconstructdatabase`.

\dtldbrowreconstruct { *row-idx* } { *row code* }

datatool v3.0+

§3.15.1.3;
345

Only for use within the *body code* argument of `\DTLreconstructdatabase`.

\DTLdbSetHeader { *col key* } { *header* }

datatool v3.0+

§3.15.1.2;
340

Does `\DTLsetheader { db-name } { col key } { header }` where *db-name* is the `default-name`.

\dtldbvaluereconstruct { *string* }

datatool v3.0+

§3.15.1.3;
345

Only for use within the *content* argument of `\dtldbcolreconstruct`.

\DTLdecimaltocurrency [*<currency symbol>*] { *<num>* } { *<cs>* } datatool–base

§2.3.2; 46

Converts the plain number *<num>* to a formatted currency using the supplied *<currency symbol>* and stores the result in the control sequence *<cs>*. If the optional argument is omitted, the default currency symbol is used.

\DTLdecimaltolocale { *<num>* } { *<cs>* } datatool–base

§2.3.2; 45

Converts the plain number *<num>* to a formatted number and stores the result in the control sequence *<cs>*.

\DTLdefaultEURcurrencyfmt datatool–base v3.0+

§2.6; 127

EUR currency formatting command.

\dtldefaultkey datatool

§3.15.2;
346

Expands to the default key prefix when column keys need to be automatically generated in `\DTLread`.

\DTLDefaultLocaleWordHandler { *<cs>* } datatool–base v3.0+

§2.9.5.2;
165

Word handler used by string sorting.

\DTLdefcurrency [*<fmt>*] { *<ISO>* } { *<symbol>* } { *<string>* } datatool–base v3.0+

§2.6; 123

Defines a new currency with associated ISO code, symbol and string equivalent. The optional argument indicates how the currency is formatted and defaults to `\dtlcurrdefaultfmt` if omitted.

\DTLdeletedb { *<db-name>* } datatool v2.03+

§3.5; 235

Deletes the database with the label *<db-name>*.

\dtldisplayafterhead

initial: *empty* datatool v2.0+

§3.7.2;
253

Used by \DTLdisplaydb and \DTLdisplaylongdb to insert content after the header.

\DTLdisplaybargrouplabel { *<text>* }

databar v3.0+

§5.4.2;
446

Used to format group bar labels.

\dtldisplaycr

initial: \tabularnewline datatool v2.19+

§3.7.2;
255

Used by \DTLdisplaydb and \DTLdisplaylongdb to insert a new line.

\DTLdisplaydb [*<omit list>*] { *<db-name>* }

datatool v2.0+

§3.7

Displays the database identified by *<db-name>* in a tabular environment, omitting the columns listed by their label in *<omit list>*.

\DTLdisplaydb* [*<options>*] { *<db-name>* }

datatool v3.0+

§3.7

Displays the database identified by *<db-name>* in a tabular environment with *<key>=<value>* list of options.

\DTLdisplaydbAddBegin { *<content-tl-var>* } { *<align-spec>* } { *<header>* }

datatool v3.0+

§3.7.2;
257

Used by \DTLdisplaydb to add the start of the tabular environment to *<content-tl-var>*.

\DTLdisplaydbAddEnd { *<content-tl-var>* }

datatool v3.0+

§3.7.2;
257

Used by \DTLdisplaydb to add the end of the tabular environment to *<content-tl-var>*.

\DTLdisplaydbAddItem { *<content-tl-var>* } { *<item>* } { *<fmt-cs>* } { *<type>* }

{ *<row-num>* } { *<row-idx>* } { *<col-num>* } { *<col-idx>* }

datatool v3.0+

§3.7.2;
255

Appends the item to the *<content-tl-var>* token list variable during construction within \DTL-

`displaydb` and `\DTLdisplaylongdb`.

`\dtldisplaydbenv`

initial: `tabular` datatool v3.0+

§3.7.2;
253

Expands to the name of the environment used by `\DTLdisplaydb`.

`\dtldisplayendtab`

initial: `empty` datatool v2.0+

§3.7.2;
253

Used by `\DTLdisplaydb` and `\DTLdisplaylongdb` to insert content before the end of the environment.

`\DTLdisplayinnerlabel` { *⟨text⟩* }

datapie

§4.5; 395

Used to format the inner label.

`\DTLdisplaylongdb` [*⟨options⟩*] { *⟨db-name⟩* }

datatool v2.0+

§3.7; 242

Displays the database identified by *⟨db-name⟩* in a longtable environment.

`\DTLdisplaylongdbAddBegin` { *⟨content-tl-var⟩* } { *⟨align-spec⟩* } { *⟨header⟩* }

datatool v3.0+

§3.7.2;
258

Used by `\DTLdisplaylongdb` to add the start of the tabular environment to *⟨content-tl-var⟩*.

`\DTLdisplaylongdbAddEnd` { *⟨content-tl-var⟩* }

datatool v3.0+

§3.7.2;
258

Used by `\DTLdisplaylongdb` to add the end of the tabular environment to *⟨content-tl-var⟩*.

`\dtldisplaylongdbenv`

initial: `longtable` datatool v3.0+

§3.7.2;
254

Expands to the name of the environment used by `\DTLdisplaylongdb`.

\DTLdisplaylowerbarlabel { *text* }

databar

§5.4.2;
445

Used to format lower bar labels.

\DTLdisplaylowermultibarlabel { *text* }

databar

§5.4.2;
446

Used to format lower bar labels for \DTLmultibarchart.

\DTLdisplayouterlabel { *text* }

datapie

§4.5; 395

Used to format the outer label.

\dtldisplaystartrow

initial: empty datatool v2.0+

§3.7.2;
253

Used by \DTLdisplaydb and \DTLdisplaylongdb to insert content at the start of each row.

\dtldisplaystarttab

initial: empty datatool v2.0+

§3.7.2;
253

Used by \DTLdisplaydb and \DTLdisplaylongdb to insert content before the header.

\DTLdisplayTBrowidxmap { *idx* }

datatool v3.0+

§3.7.2;
258

Provided for use with `row-idx-map-function`, this command expands to a row index that will arrange data rows from top to bottom instead of left to right when `per-row` is greater than 1. Note that this is only designed to work when no rows are omitted.

\DTLdisplayupperbarlabel { *text* }

databar

§5.4.2;
446

Used to format upper bar labels.

\DTLdisplayuppermultibarlabel { *text* }

databar

§5.4.2;
446

Used to format upper bar labels for \DTLmultibarchart.

\dtldisplayvalign

initial: C datatool v2.11+

§3.7.2;
253

Expands to the tabular vertical alignment specifier used by `\DTLdisplaydb`.

\DTLdiv{ $\langle cs \rangle$ }{ $\langle num1 \rangle$ }{ $\langle num2 \rangle$ }

datatool-base

§2.5.2;
110

Calculates $\langle num1 \rangle \div \langle num2 \rangle$ and stores the result in the control sequence $\langle cs \rangle$, where the numbers are formatted numbers. The result will be a formatted number.

\dtldiv{ $\langle cs \rangle$ }{ $\langle num1 \rangle$ }{ $\langle num2 \rangle$ }

datatool-base

§2.5.1; 99

Calculates $\langle num1 \rangle \div \langle num2 \rangle$ and stores the result in the control sequence $\langle cs \rangle$, where the numbers are plain numbers.

\DTLdobarcolor[$\langle value \rangle$]{ $\langle index \rangle$ }

databar

§5.4.3;
447

Shortcut that does `\color{ $\langle col \rangle$ }` where $\langle col \rangle$ is obtained by expanding `\DTLgetbarcolor{ $\langle index \rangle$ }` if the value is non-negative or omitted and to `\DTLgetnegbarcolor{ $\langle index \rangle$ }`, otherwise. Does nothing if an empty colour specification has been applied to the given bar index.

\DTLdocurrentbarcolor

databar

§5.4.3;
447

For use in bar chart hooks, this will use `\DTLdobarcolor` for the current bar index.

\DTLdocurrentpiesegmentcolor

datapie

§4.6; 397

Locally sets the current text colour to that of the current pie segment.

\DTLdopiesegmentcolor{ $\langle n \rangle$ }

datapie

§4.6; 396

Locally sets the current colour (with `\color`) to that associated with the $\langle n \rangle$ th segment.

\DTLencapbibfield{*<cs>*}{*<field name>*}

datbib v3.0+

§7.8; 550

Similar to `\DTLbibfield{<field name>}` but encapsulates the value with *<cs>*. Expands to nothing if the field isn't set or doesn't exist.

\DTLendbibitem

datbib

§7.7.1;
546

Hook used within `\DTLbibliography` and `\DTLmbibliography` at the end of each iteration. Does nothing by default.

\DTLendpt

databar

§5.4.4;
449

For use in the definition of `\DTLeverybarhook`, this expands to the end point of the current bar along its mid-axis as a `pgf` point.

\DTLenLocaleGetGroupString{*<actual>*}{*<sort value>*}{*<cs>*}

datatool–english

English group string handler.

\DTLenLocaleHandler{*<cs>*}

datatool–english

§2.3.5; 61

English locale word handler.

\DTLeverybargrouphook

databar v3.0+

§5.4.4;
449

Hook used by `\DTLmultibarchart` at the end of every bar group.

\DTLeverybarhook

databar

§5.4.4;
448

Hook used by `\DTLbarchart` and `\DTLmultibarchart` after the current bar is drawn.

\DTLeveryprebarhook

databar v3.0+

§5.4.4;
448

Hook used by `\DTLbarchart` and `\DTLmultibarchart` before the current bar is drawn.

\dtlexpandnewvalue

datatool

§3.1; 179

Ensures that new values are expanded before being added to a database (equivalent to the `new-value-expand=true` option). Has no effect when loading `dbtex` files.

\dtlfallbackaction { $\langle val1 \rangle$ } { $\langle val2 \rangle$ } { $\langle swap\ code \rangle$ } { $\langle no\ swap\ code \rangle$ }

datatool-base v3.0+

§2.9.5;
159

Used during sorting when two sort values are identical. The $\langle val1 \rangle$ and $\langle val2 \rangle$ arguments are the *original* values, which may not be identical. This command should expand to $\langle swap\ code \rangle$, if the values should be swapped and $\langle no\ swap\ code \rangle$ otherwise. The default behaviour uses `\DTLifstringgt` to compare the original $\langle val1 \rangle$ and $\langle val2 \rangle$.

\DTLfetchlistelement { $\langle list \rangle$ } { $\langle idx \rangle$ } { $\langle cs \rangle$ }

datatool-base

§2.9.3;
153

Fetches the $\langle idx \rangle$ th element in the list (starting with 1 for the first element) and defines the control sequence $\langle cs \rangle$ to that element value.

\DTLfmtcurr { $\langle currency-code \rangle$ } { $\langle value \rangle$ }

datatool-base v3.0+

§2.6; 129

Formats $\langle value \rangle$ as a currency according to the given currency code (which should already have been defined).

\DTLfmtcurrency { $\langle symbol \rangle$ } { $\langle value \rangle$ }

datatool-base v3.0+

§2.6; 127

Formats $\langle value \rangle$ as a currency with the given symbol.

\dtlforcolumn { $\langle cs \rangle$ } { $\langle db-name \rangle$ } { $\langle key \rangle$ } { $\langle body \rangle$ } *modifier:* * datatool

§3.16.2;
375

Loops through the column identified by $\langle key \rangle$ in the database identified by $\langle db-name \rangle$ and, for each iteration, defines $\langle cs \rangle$ to the value of the entry in the row.

\dtlforcolumidx { *<cs>* } { *<db-name>* } { *<col-idx>* } { *<body>* } *modifier:* *
datatool v2.0+

§3.16.2;
375

Loops through the column with the index *<col-idx>* in the database identified by *<db-name>* and, for each iteration, defines *<cs>* to the value of the entry in the row.

\DTLforeach [*<condition>*] { *<db-name>* } { *<assign-list>* } { *<body>* } *modifier:* *
datatool

§3.8.2

Iterates over the database identified by *<db-name>*, performs the assignments given in *<assign-list>* and does *<body>*.

\DTLforeachbibentry [*<condition>*] { *<db-name>* } { *<body>* } *modifier:* *
databib

§7.8; 549

Iterates over all rows in the given database using `\DTLforeach` (or `\DTLforeach*` for the starred form), setting up the placeholder commands `\DBIBname`, `\DBIBcitekey` and `\DBIBentrytype` and, for each row that satisfies the given (ifthen) condition, increments `DTLbibrow` and does *<body>*. Locally assigns placeholder commands.

\dtlforeachkey (*<key-cs>*, *<col-cs>*, *<type-cs>*, *<header-cs>*) \in { *<db-name>* }
\do { *<body>* } datatool

§3.16.2;
374

Iterates over each column in the database identified by *<db-name>*, and at each iteration defines *<key-cs>* to expand to the column key, *<col-cs>* to expand to the column index, *<type-cs>* to expand to the data type, and *<header-cs>* to expand to the column header, and then does *<body>*.

\DTLforeachkeyinrow { *<cs>* } { *<code>* } datatool

Iterates over each column in the current row of `\DTLforeach`, defines *<cs>* to the value in the current column, and does *<code>*.

\dtlforeachlevel datatool

§3.8.2;
294

A count register that keeps track of the current `\DTLforeach` nested level.

\dtlforint $\langle count-reg \rangle = \langle start \rangle \backslash to \langle end \rangle \backslash step \langle inc \rangle \backslash do \{ \langle body \rangle \}$ datatool-base

§3.16.2;
375

Integer iteration from $\langle start \rangle$ to $\langle end \rangle$, incrementing by $\langle inc \rangle$ using $\langle count-reg \rangle$ as the loop variable. This command is retained for backward-compatibility but L^AT_EX3 now provides integer step functions.

\DTLformatabbrvforenames $\{ \langle forenames \rangle \}$ databib

§7.7.1;
544

As $\backslash \text{DTLformatforenames}$ but converts the forenames to initials with $\backslash \text{DTLstoreinitials}$.

\DTLformatauthor $\{ \langle von part \rangle \} \{ \langle surname \rangle \} \{ \langle jr part \rangle \} \{ \langle forenames \rangle \}$ databib

§7.7.1;
543

Formats the author's name.

\DTLformatauthorlist databib

§7.7.1;
545

Used by styles to format the list of author names (which will be obtained from the `Author` field). Each name is formatted according to $\backslash \text{DTLformatauthor}$.

\DTLformatbibentry databib

§7.8; 551

Designed for use within $\backslash \text{DTLbibliography}$ to format the database row in the current iteration of $\backslash \text{DTLforeachbibentry}$.

\DTLformatbibnamelist $\{ \langle list \rangle \} \{ \langle max \rangle \} \{ \langle fmt-cs \rangle \}$ databib v3.0+

§7.7.1;
545

Used by $\backslash \text{DTLformatauthorlist}$ and $\backslash \text{DTLformateditorlist}$ to format the list of names. Each element in the list must be in the form $\{ \langle von \rangle \} \{ \langle surname \rangle \} \{ \langle jr \rangle \} \{ \langle forenames \rangle \}$ and $\langle fmt-cs \rangle$ must have those four arguments for its syntax. If the list has more than $\langle max \rangle$ items, it will be truncated with $\backslash \text{etalname}$.

\DTLformatdate databib

§7.7.1;
546

Formats the date obtained from the `Date` field if set, or from the `Year` and `Month` fields (if set).

\DTLformatteditor { *von part* } { *surname* } { *jr part* } { *forenames* } databib

§7.7.1;
543

Formats the editor's name.

\DTLformatteditorlist

databib

§7.7.1;
545

Used by styles to format the list of editor names (which will be obtained from the `Editor` field). Each name is formatted according to `\DTLformatteditor`.

\DTLformatforenames { *forenames* }

databib

§7.7.1;
544

Formats the forenames (with a leading space and updating conditionals).

\DTLformatjr { *jr* }

databib

§7.7.1;
544

Formats *jr* (with a leading space and updating conditionals) if not empty.

\DTLformatlegend { *legend code* }

dataplot

§6.3.3;
514

Outer formatting of the legend.

\DTLformatlist { *list* }

modifier: * datatool-base v2.28+

§2.9.2;
151

Formats the supplied CSV list. The unstarred version adds grouping.

\DTLformatpages

databib

§7.7.1;
546

Formats page(s).

\DTLformatsurname { *surname* }

databib

§7.7.1;
544

Formats *surname* (with a leading space and updating conditionals).

\DTLformatsurnameonly { *<von part>* } { *<surname>* } { *<jr part>* } { *<forenames>* }

databib

§7.7.1;
544

Formats the name, omitting the forenames.

\DTLformatthisbibentry { *<db-name>* } { *<cite key>* }

databib

§7.8; 552

As `\DTLformatbibentry` but for the row identified by *<cite key>* in the given database *<db-name>*.

\DTLformatvon { *<von>* }

databib

§7.7.1;
544

Formats *<von>* (with a leading space and updating conditionals) if not empty.

\DTLgabs { *<cs>* } { *<num>* }

datatool-base

§2.5.2;
111

As `\DTLabs` but globally defines *<cs>*.

\DTLgadd { *<cs>* } { *<num1>* } { *<num2>* }

datatool-base

§2.5.2;
108

As `\DTLadd` but globally defines *<cs>*.

\DTLgaddall { *<cs>* } { *<num list>* }

datatool-base

§2.5.2;
110

As `\DTLaddall` but globally defines *<cs>*.

\DTLgcleardb { *<db-name>* }

datatool v2.13+

§3.5; 236

Clears the database with the label *<db-name>*. That is, the database becomes empty but is still defined.

\DTLgclip { *<cs>* } { *<num>* }

datatool-base

§2.5.2;
112

As `\DTLclip` but globally defines *<cs>*.

\DTLgdeletedb { *<db-name>* }

datatool v2.13+

§3.5; 235

Deletes the database with the label *<db-name>*.

\DTLgdiv { *<cs>* } { *<num1>* } { *<num2>* }

datatool-base

§2.5.2;
110

As `\DTLdiv` but globally defines *<cs>*.

\DTLget [*<property>*] { *<cs>* }

datatool v3.0+

§3.3; 206

Gets the returned value from the last `\DTLaction` in the current scope, and stores the value in the token list control sequence *<cs>*. Leave the optional argument *<property>* empty (or omit) for the main return value, otherwise *<property>* should identify the particular value when there are multiple return values.

\DTLgetbarcolor { *<index>* }

databar

§5.4.3;
447

Expands to the colour for the given index in the bar colour list or `white` if not set. This takes the `color-style` setting into account.

\DTLgetcolumnindex { *<cs>* } { *<db-name>* } { *<col key>* }

datatool v2.0+

modifier: *

§3.6; 237

Gets the index for the column labelled *<col key>* from the database identified by *<db-name>* and defines the control sequence *<cs>* to expand to that value. The starred version doesn't check if the database or column exists.

\DTLgetdatatype { *<cs>* } { *<db-name>* } { *<col key>* } modifier: * datatool v2.0+

§3.6; 238

Sets the control sequence *<cs>* to expand to the data type integer identifier for the column labelled *<col key>* in the database identified by *<db-name>*.

\DTLgetDataTypeName { *<number>* }

datatool-base v3.0+

§2.2.3; 18

Expands to textual label associated with the data type *<number>*.

\dtlgetentryfromcurrentrow { *cs* } { *col num* }

datatool v2.0+

§3.16.1;
372

Gets the value for the column identified by its index *col num* in `\dtlcurrentrow` and locally defines the control sequence *cs* to that value. This is just a shortcut that uses `\dtlgetentryfromrow`.

\dtlgetentryfromrow { *cs* } { *col num* } { *row toks* }

datatool v2.0+

Gets the value for the column identified by its index *col num* in the token *row toks* (that should be in the correct row spec format) and locally defines the control sequence *cs* to that value.

\DTLGetInitialLetter { *text* } { *cs* }

datatool-base v3.0+

§2.8.2;
142

Gets the initial letter of *text* and stores it in *cs*.

\DTLgetkeyforcolumn { *cs* } { *db-name* } { *col idx* }

datatool v2.0+

modifier: *

§3.6; 238

Gets the key for the column with the index *col idx* from the database identified by *db-name* and defines the control sequence *cs* to expand to that value. The starred version doesn't check if the database or column exists.

\DTLgetlocation { *row-cs* } { *col-cs* } { *db-name* } { *value* }

datatool

§3.16; 370

Defines the control sequences *row-cs* and *col-cs* to expand to the row and column indexes, respectively, of the first entry in the database identified by *db-name* that matches the given *value*.

\DTLgetnegbarcolor { *index* }

databar

§5.4.3;
447

Expands to the colour for the given index in the negative bar colour list or the default colour list if not set. This takes the `negative-color-style` setting into account.

\DTLgetpiesegmentcolor { *n* }

datapie

§4.6; 396

Expands to the current colour to that associated with the *n*th segment or `white` if not set.

\dtlgetrow{*<db-name>*}{*<row idx>*}

datatool

§3.16.1;
372

Gets the row identified by the row index *<row idx>* from the database identified by *<db-name>* and stores the row in `\dtlcurrentrow` with preceding rows in `\dtlbeforerow` and following rows in `\dtlafterrow`. The row index is stored in `\dtlrownum`, and the database label *<db-name>* is stored in `\dtldbname`. This command assumes that the given row exists. Assignments are local. This command does not check the `global` option.

\dtlgetrowforvalue{*<db-name>*}{*<col idx>*}{*<value>*}

datatool v2.11+

§3.16.1;
372

As `\dtlgetrow` but gets the row where the entry in the column identified by its index *<col idx>* matches *<value>*.

\DTLgetrowindex{*<row-cs>*}{*<db-name>*}{*<col idx>*}{*<value>*} *modifier: **
datatool v2.11+

§3.16; 370

Defines the control sequence *<row cs>* to expand to the row index of the first entry in the column identified by its index *<col idx>* that matches the given *<value>* in the database identified by *<db-name>*. The unstarred version triggers an error if not found.

\dtlgetrowindex{*<row-cs>*}{*<db-name>*}{*<col idx>*}{*<value>*} datatool v2.11+

§3.16; 369

As `\DTLgetrowindex` but doesn't produce an error if not found.

\DTLgetvalue{*<cs>*}{*<db-name>*}{*<row idx>*}{*<col idx>*}

datatool

§3.16; 370

Gets the element in the row identified by *<row idx>* for the column identified by its index *<col idx>* in the database identified by *<db-name>* and defines the control sequence *<cs>* to expand to the element's value.

\dtlforint*<count-reg>*=*<start>*\to*<end>*\step*<inc>*\do{*<body>*}
datatool-base

§3.16.2;
375

As `\dtlforint` but globally sets the count register.

\DTLgidxAcrStyle { *⟨long⟩* } { *⟨short⟩* }

datagidx

§8.7; 606

Used to encapsulate the long and short form in the `Text` field.

\DTLgidxAssignList

datagidx

§8.9.1;
619

Expands to the assignment list used by `\DTLgidxForeachEntry`, which is internally used by `\printterms`.

\DTLgidxCategoryNameFont { *⟨text⟩* }

datagidx

§8.8.2.5;
616

Specific to `style=dict`, this encapsulates the “category” child name.

\DTLgidxCategorySep

initial: `\space` datagidx

§8.8.2.5;
616

Separator used with `style=dict` between “category” child entries.

\DTLgidxChildCountLabel

datagidx

§8.8.1;
612

Used by `child=noname` to show the child index instead of the name.

\DTLgidxChildSep

initial: `□` datagidx

§8.8.2.4;
615

Separator between child entries for `style=gloss`.

\DTLgidxChildStyle { *⟨name⟩* }

datagidx

§8.8.1;
612

Encapsulates the child entry’s name (including font and case-changing commands and post name hook).

\DTLgidxCounter

initial: `page` datagidx

§8.5.2;
603

Expands to the counter to use for locations.

\DTLgidxCurrentdb

datagidx

§8.8; 610

Defined by `\printterms` to the name of the current database.

\DTLgidxDictHead

datagidx

§8.8.2.5;
616

Specific to `style=dict`, this inserts the group header.

\DTLgidxDictPostItem

initial: \DTLgidxEndItem datagidx

§8.8.2.5;
616

Specific to the `dict` style, this is done at the end of each item.

\DTLgidxDisableHyper

datagidx

§8.9; 618

Disable hyperlinks.

\DTLgidxEnableHyper

datagidx

§8.9; 618

Enable hyperlinks, if supported.

\DTLgidxEndItem

datagidx

§8.8.1;
613

Inserted at the end of each item (after the location list, child list and cross references, if applicable).

\DTLgidxFetchEntry { *<cs>* } { *<label>* } { *<col-key>* }

datagidx

§8.5; 601

Fetches the value of the given column in the row identified by *<label>* from the index/glossary database associated with *<label>* and defines the command *<cs>* to expand to that value.

\DTLgidxForeachEntry { *<body>* }

datagidx

§8.9.1;
619

Used by `\printterms` to iterate over the database.

\DTLgidxFormatAcr { *⟨label⟩* } { *⟨long-field⟩* } { *⟨short-field⟩* } datagidx

§8.7.1;
608

Used by `\acr` and `\acrpl` to format the full form.

\DTLgidxFormatAcrUC { *⟨label⟩* } { *⟨long-field⟩* } { *⟨short-field⟩* } datagidx

§8.7.1;
609

Used by `\Acr` and `\Acrpl` to format the full form.

\DTLgidxFormatDesc { *⟨text⟩* } datagidx

§8.8.1;
612

Encapsulates the description, if set.

\DTLgidxFormatSee { *⟨tag⟩* } { *⟨label list⟩* } datagidx

§8.8.1;
613

Used to format the “see” cross-reference list.

\DTLgidxFormatSeeAlso { *⟨tag⟩* } { *⟨label list⟩* } datagidx

§8.8.1;
613

Used to format the “see also” cross-reference list.

\DTLgidxGobble { *⟨text⟩* } datagidx

§8.9.2;
621

Used by `\newterm` when creating a label or sort value, expands to nothing.

\DTLgidxGroupHeaderTitle { *⟨group⟩* } datagidx

§8.8; 610

Expands to the title corresponding to the given letter group.

\DTLgidxIgnore { *⟨text⟩* } datagidx

§8.4.1;
596

Normally simply expands to *⟨text⟩* but may be used to markup content to be stripped by the automated label and sort values.

\DTLgidxLocation

datagidx

Used to display the location list, if applicable.

\DTLgidxMac { *text* }

datagidx

§8.4.1;
595

Normally simply expands to *text* but may be used to markup content to be converted to “Mac” in the sort value.

\DTLgidxName { *forename* } { *surname* }

datagidx

§8.4.1;
591

Used to markup a person’s name.

\DTLgidxNameCase { *text* }

datagidx

§8.8.1;
611

Used to apply the appropriate case change to the name.

\DTLgidxNameFont { *text* }

datagidx

§8.8.1;
611

Used to apply the appropriate font change to the name.

\DTLgidxNameNum { *num* }

datagidx

§8.4.1;
592

Used to markup a person’s ordinal (for a monarch etc).

\DTLgidxNoFormat { *text* }

datagidx

§8.9.2;
621

Used by `\newterm` when creating a label or sort value, simply expands to *text*.

\DTLgidxOffice { *office* } { *name* }

datagidx

§8.4.1;
593

Used to markup a person’s office.

\DTLgidxParen { *⟨text⟩* }

datagidx

§8.4.1;
595

Used to markup parenthetical material.

\DTLgidxParticle { *⟨particle⟩* } { *⟨surname⟩* }

datagidx

§8.4.1;
594

Used to markup a surname with a particle.

\DTLgidxPlace { *⟨country⟩* } { *⟨town/city⟩* }

datagidx

§8.4.1;
592

Used to markup a place.

\DTLgidxPostChild

initial: empty datagidx

§8.8.2.4;
615

Hook inserted at the end of the list of child entries for `style=gloss`.

\DTLgidxPostChildName

initial: \DTLgidxPostName datagidx

§8.8.1;
612

Inserted after the child name.

\DTLgidxPostDescription

initial: empty datagidx

§8.8.1;
612

Inserted after the description.

\DTLgidxPostLocation

initial: □ datagidx

§8.8.1;
613

Inserted after the location list.

\DTLgidxPostName

initial: □ datagidx

§8.8.1;
612

Inserted after the name.

\DTLgidxPreLocation

initial: \enspace datagidx

§8.8.1;
613

Inserted before the location list.

\DTLgidxRank { *<title>* } { *<forename(s)/initial(s)>* }

datagidx

§8.4.1;
594

Used to markup a person's rank.

\DTLgidxSaint { *<text>* }

datagidx

§8.4.1;
595

Normally simply expands to *<text>* but may be used to markup content to be converted to “Saint” in the sort value.

\DTLgidxSeeTagFont { *<text>* }

datagidx

§8.8.1;
613

Used to format the *<tag>* argument of \DTLgidxFormatSee.

\DTLgidxSetColumns { *<n>* }

datagidx

§8.8.1;
611

Sets the number of columns in the index.

\DTLgidxSetCompositor { *<character>* }

datagidx

§8.5.2;
604

Sets the location compositor.

\DTLgidxSetDefaultDB { *<db-name>* }

datagidx

Sets the default datagidx database name.

\DTLgidxStripBackslash { *<cs>* }

datagidx

§8.4.1;
596

Expands to the command name without the leading backslash.

\DTLgidxSubCategorySep

initial: \space datagidx

§8.8.2.5;
616

Separator used with `style=dict` between sub-category child entries.

\DTLgidxSubject { $\langle main \rangle$ } { $\langle category \rangle$ }

datagidx

§8.4.1;
592

Used to markup a subject.

\DTLgidxSymDescSep

initial: \space datagidx

§8.8.1;
612

Separator between symbol and description if both are present.

\DTLgmax { $\langle cs \rangle$ } { $\langle num1 \rangle$ } { $\langle num2 \rangle$ }

datatool-base

§2.5.2;
113

As `\DTLmax` but globally defines $\langle cs \rangle$.

\DTLgmaxall { $\langle cs \rangle$ } { $\langle num list \rangle$ }

datatool-base

§2.5.2;
113

As `\DTLmaxall` but globally defines $\langle cs \rangle$.

\DTLgmeanforall { $\langle cs \rangle$ } { $\langle num list \rangle$ }

datatool-base

§2.5.2;
113

As `\DTLmeanforall` but globally defines $\langle cs \rangle$.

\DTLgmin { $\langle cs \rangle$ } { $\langle num1 \rangle$ } { $\langle num2 \rangle$ }

datatool-base

§2.5.2;
112

As `\DTLmin` but globally defines $\langle cs \rangle$.

\DTLgminall { $\langle cs \rangle$ } { $\langle num list \rangle$ }

datatool-base

§2.5.2;
113

As `\DTLminall` but globally defines $\langle cs \rangle$.

\DTLgmul { *cs* } { *num1* } { *num2* }

datatool-base

§2.5.2;
110

As \DTLmul but globally defines *cs*.

\DTLgneg { *cs* } { *num* }

datatool-base

§2.5.2;
111

As \DTLneg but globally defines *cs*.

\DTLgnewdb { *db-name* }

datatool v2.13+

§3.4; 232

Globally creates a new database with the label *db-name*.

\DTLground { *cs* } { *num* } { *num digits* }

datatool-base

§2.5.2;
112

As \DTLround but globally defines *cs*.

\DTLgsdforall { *cs* } { *num list* }

datatool-base

§2.5.2;
114

As \DTLsdforall but globally defines *cs*.

\DTLgsqrt { *cs* } { *num* }

datatool-base

§2.5.2;
111

As \DTLsqrt but globally defines *cs*.

\DTLgsub { *cs* } { *num1* } { *num2* }

datatool-base

§2.5.2;
110

As \DTLsub but globally defines *cs*.

\DTLgt trunc { *cs* } { *num* } { *num digits* }

datatool-base

§2.5.2;
112

As \DTLtrunc but globally defines *cs*.

\DTLgvarianceforall { *<cs>* } { *<num list>* }

datatool-base

§2.5.2;
114

As `\DTLvarianceforall` but globally defines *<cs>*.

\dtlheader

datatool

§3.8.2;
297

Defined by `\DTLforeachkeyinrow` to the current column header.

\dtlheaderformat { *<text>* }

datatool v2.0+

§3.7.2;
252

Used by `\dtlcolumnheader` to apply any font change to the header text.

\dtlcompare { *<count-reg>* } { *<string1>* } { *<string2>* }

datatool-base

§2.9.5.1;
163

Compares *<string1>* and *<string2>* and stores the result in the count register *<count-reg>*, where 0 indicates the strings are the same, -1 means that *<string1>* comes before (less than) *<string2>* and 1 means that *<string1>* comes after (greater than) *<string2>* (case-insensitive).

\DTLidxFormatSeeItem { *<label>* }

datagidx

§8.8.1;
613

Used to fetch the name of the cross-reference label and display it in a hyperlink, if supported.

\DTLidxSeeLastSep

initial: `\&` datagidx

§8.8.1;
614

Separator between final pair of items in cross-reference list.

\DTLidxSeeSep

initial: `,` datagidx

§8.8.1;
614

Separator used in all but final pair of items in cross-reference list.

\DTLifaction { *<property>* } { *<true>* } { *<false>* }

datatool v3.0+

§3.3; 206

Expands to *<true>* if the last action set the given property or *<false>* otherwise.

\DTLifAllLowerCase { *<string>* } { *<true>* } { *<false>* }

datatool-base

§2.4.1.2;
71

Does *<true>* if *<string>* is all in lowercase (disregarding punctuation and spaces) otherwise does *<false>*.

\DTLifAllUpperCase { *<string>* } { *<true>* } { *<false>* }

datatool-base

§2.4.1.2;
71

Does *<true>* if *<string>* is all in uppercase (disregarding punctuation and spaces) otherwise does *<false>*.

\DTLifanybibfieldexists { *<field list>* } { *<true>* } { *<false>* }

datbib

§7.8; 551

For use in `\DTLforeachbibentry`, this tests if any of the given fields exists (that is, if both the column is defined and the value is not null) for the current iteration.

\DTLifbibfieldexists { *<field name>* } { *<true>* } { *<false>* }

datbib

§7.8; 551

For use in `\DTLforeachbibentry`, this tests if the given field exists (that is, if both the column is defined and the value is not null) for the current iteration.

\DTLifcasedatatype { *<arg>* } { *<string case>* } { *<int case>* } { *<real case>* } { *<currency case>* }

datatool-base

§2.4.1.1;
64

Parses the first argument *<arg>* and does *<string case>* if *<arg>* is a string, *<int case>* if *<arg>* is an integer, *<real case>* if *<arg>* is a real (decimal), or *<currency case>* if *<arg>* is currency.

\DTLifclosedbetween { *<value>* } { *<min>* } { *<min>* } { *<true>* } { *<false>* }

modifier: * datatool-base

§2.4.1.5;
88

Uses either `\DTLifnumclosedbetween` or `\DTLifstringclosedbetween` depending on the data type of *<value>*, *<min>* and *<max>*. The starred version ignores case if `\DTLifstringclosedbetween` is required.

\DTLifcurrency { *<arg>* } { *<true>* } { *<false>* }

datatool-base

§2.4.1.1;
63

Does *<true>* if the first argument *<arg>* is a currency formatted number (not an integer or decimal)

otherwise does *<false>*.

\DTLifcurrencyunit { *<arg>* } { *<symbol>* } { *<true>* } { *<false>* } datatool-base

§2.4.1.1;
63

Does *<true>* if the first argument *<arg>* is a currency formatted number (not an integer or decimal) and has the given currency *<symbol>* otherwise does *<false>*.

\DTLifdbempty { *<db-name>* } { *<true>* } { *<false>* } datatool

§3.6; 236

Does *<true>* if a database with the label *<db-name>* is empty, otherwise does *<false>*.

\DTLifdbexists { *<db-name>* } { *<true>* } { *<false>* } datatool

§3.6; 236

Does *<true>* if a database with the label *<db-name>* exists, otherwise does *<false>*.

\DTLifEndsWith { *<string>* } { *<fragment>* } { *<true>* } { *<false>* } *modifier:* *
datatool-base v3.0+

§2.4.1.2;
71

Does *<true>* if *<string>* ends with *<fragment>* otherwise does *<false>*. The starred version is case-insensitive.

\DTLifeq { *<arg1>* } { *<arg2>* } { *<true>* } { *<false>* } *modifier:* * datatool-base

§2.4.1.5;
87

Uses either `\DTLifnumeq` or `\DTLifstringeq` depending on the data type of both *<arg1>* and *<arg2>*. The starred version ignores case if `\DTLifstringeq` is required.

\DTLiffirstrow { *<true>* } { *<false>* } datatool

§3.8.2;
296

May only be used within `\DTLforeach`, this command does *<true>* if the current row is the first iteration of the loop (that is, where `DTLrow<n>` is 1), otherwise it does false. Note that if `\DTLforeach` has a condition, this references the loop index which may not match the actual row index.

\DTLifFPclosedbetween { *<num>* } { *<min>* } { *<min>* } { *<true>* } { *<false>* }
datatool-base

§2.4.1.4;
84

Synonym of `\dtlifnumclosedbetween`.

\DTLifFPopenbetween { *<num>* } { *<min>* } { *<min>* } { *<true>* } { *<false>* }
datatool-base

§2.4.1.4;
83

Synonym of \dtlifnumopenbetween.

\DTLifgt { *<arg1>* } { *<arg2>* } { *<true>* } { *<false>* } *modifier:* * datatool-base

§2.4.1.5;
88

Uses either \DTLifnumgt or \DTLifstringgt depending on the data type of both *<arg1>* and *<arg2>*. The starred version ignores case if \DTLifstringgt is required.

\DTLifhaskey { *<db-name>* } { *<key>* } { *<true>* } { *<false>* } *modifier:* *
datatool v2.0+

§3.6; 237

Tests if the database with the label *<db-name>* has a column with the given *<key>* (label). The starred version doesn't check if the database exists.

\DTLifinlist { *<element>* } { *<list>* } { *<true>* } { *<false>* } datatool-base

§2.4.1.2;
69

Does *<true>* if *<element>* is in the CSV *<list>* otherwise does *<false>*. One level expansion performed on *<list>* but not *<element>*.

\DTLifint { *<arg>* } { *<true>* } { *<false>* } datatool-base

§2.4.1.1;
62

Does *<true>* if the first argument *<arg>* is an integer formatted number (not a decimal or currency) otherwise does *<false>*.

\dtlifintclosedbetween { *<num>* } { *<min>* } { *<min>* } { *<true>* } { *<false>* }
datatool-base

§2.4.1.4;
84

Does *<true>* if $\langle min \rangle \leq \langle num \rangle \leq \langle max \rangle$ otherwise does *<false>*, where the numbers are plain number integers.

\dtlifintopenbetween { *<num>* } { *<min>* } { *<min>* } { *<true>* } { *<false>* }
datatool-base

§2.4.1.4;
83

Does *<true>* if $\langle min \rangle < \langle num \rangle < \langle max \rangle$ otherwise does *<false>*, where the numbers are plain number integers.

\DTLiflastrow{ $\langle true \rangle$ }{ $\langle false \rangle$ }

datatool

§3.8.2;
296

May only be used within `\DTLforeach`, this command does $\langle true \rangle$ if the current loop is on the last row.

\DTLiflt{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }{ $\langle true \rangle$ }{ $\langle false \rangle$ }

modifier: * datatool-base

§2.4.1.5;
87

Uses either `\DTLifnumlt` or `\DTLifstringlt` depending on the data type of both $\langle arg1 \rangle$ and $\langle arg2 \rangle$. The starred version ignores case if `\DTLifstringlt` is required.

\DTLifnull{ $\langle arg \rangle$ }{ $\langle true \rangle$ }{ $\langle false \rangle$ }

datatool

§3.10; 311

Expands to $\langle true \rangle$ if the first argument represents null (see §3.10), or to $\langle false \rangle$ otherwise.

\DTLifnulloreempty{ $\langle arg \rangle$ }{ $\langle true \rangle$ }{ $\langle false \rangle$ }

datatool v2.20+

§3.10; 311

Expands to $\langle true \rangle$ if the first argument is empty or represents null (see §3.10), or to $\langle false \rangle$ otherwise.

\DTLifnumclosedbetween{ $\langle num \rangle$ }{ $\langle min \rangle$ }{ $\langle min \rangle$ }{ $\langle true \rangle$ }{ $\langle false \rangle$ }

datatool-base

§2.4.1.3;
82

Does $\langle true \rangle$ if $\langle min \rangle \leq \langle num \rangle \leq \langle max \rangle$ otherwise does $\langle false \rangle$, where the numbers are formatted numbers.

\dtlifnumclosedbetween{ $\langle num \rangle$ }{ $\langle min \rangle$ }{ $\langle min \rangle$ }{ $\langle true \rangle$ }{ $\langle false \rangle$ }

datatool-base

§2.4.1.4;
83

Does $\langle true \rangle$ if $\langle min \rangle \leq \langle num \rangle \leq \langle max \rangle$ otherwise does $\langle false \rangle$, where the numbers are plain numbers.

\DTLifnumeq{ $\langle num1 \rangle$ }{ $\langle num2 \rangle$ }{ $\langle true \rangle$ }{ $\langle false \rangle$ }

datatool-base

§2.4.1.3;
81

Does $\langle true \rangle$ if $\langle num1 \rangle$ equals $\langle num2 \rangle$ otherwise does $\langle false \rangle$, where the numbers are formatted numbers.

\dtlifnumeq { *num1* } { *num2* } { *true* } { *false* }

datatool-base

§2.4.1.4;
83

Does *true* if *num1* equals *num2* otherwise does *false*, where the numbers are plain numbers.

\DTLifnumerical { *arg* } { *true* } { *false* }

datatool-base

§2.4.1.1;
63

Does *true* if the first argument *arg* is numerical (a formatted number that's an integer, decimal or currency) otherwise does *false*.

\DTLifnumgt { *num1* } { *num2* } { *true* } { *false* }

datatool-base

§2.4.1.3;
81

Does *true* if *num1* is greater than *num2* otherwise does *false*, where the numbers are formatted numbers.

\dtlifnumgt { *num1* } { *num2* } { *true* } { *false* }

datatool-base

§2.4.1.4;
83

Does *true* if *num1* is greater than *num2* otherwise does *false*, where the numbers are plain numbers.

\DTLifnumlt { *num1* } { *num2* } { *true* } { *false* }

datatool-base

§2.4.1.3;
81

Does *true* if *num1* is less than *num2* otherwise does *false*, where the numbers are formatted numbers.

\dtlifnumlt { *num1* } { *num2* } { *true* } { *false* }

datatool-base

§2.4.1.4;
83

Does *true* if *num1* is less than *num2* otherwise does *false*, where the numbers are plain numbers.

\DTLifnumopenbetween { *num* } { *min* } { *max* } { *true* } { *false* }

datatool-base

§2.4.1.3;
81

Does *true* if *min* < *num* < *max* otherwise does *false*, where the numbers are formatted numbers.

\dtlifnumopenbetween { $\langle num \rangle$ } { $\langle min \rangle$ } { $\langle min \rangle$ } { $\langle true \rangle$ } { $\langle false \rangle$ }
 datatool-base

§2.4.1.4;
83

Does $\langle true \rangle$ if $\langle min \rangle < \langle num \rangle < \langle max \rangle$ otherwise does $\langle false \rangle$, where the numbers are plain numbers.

\DTLifoddrow { $\langle true \rangle$ } { $\langle false \rangle$ } datatool

§3.8.2;
296

May only be used within `\DTLforeach`, this command does $\langle true \rangle$ if the current loop index is odd.

\DTLifopenbetween { $\langle value \rangle$ } { $\langle min \rangle$ } { $\langle min \rangle$ } { $\langle true \rangle$ } { $\langle false \rangle$ }
modifier: * datatool-base

§2.4.1.5;
88

Uses either `\DTLifnumopenbetween` or `\DTLifstringopenbetween` depending on the data type of $\langle value \rangle$, $\langle min \rangle$ and $\langle max \rangle$. The starred version ignores case if `\DTLifstringopenbetween` is required.

\DTLifreal { $\langle arg \rangle$ } { $\langle true \rangle$ } { $\langle false \rangle$ } datatool-base

§2.4.1.1;
63

Does $\langle true \rangle$ if the first argument $\langle arg \rangle$ is a real (decimal) formatted number (not an integer or currency) otherwise does $\langle false \rangle$.

\DTLifStartsWith { $\langle string \rangle$ } { $\langle fragment \rangle$ } { $\langle true \rangle$ } { $\langle false \rangle$ } *modifier:* *
 datatool-base

§2.4.1.2;
70

Does $\langle true \rangle$ if $\langle string \rangle$ starts with $\langle fragment \rangle$ otherwise does $\langle false \rangle$. The starred version is case-insensitive.

\DTLifstring { $\langle arg \rangle$ } { $\langle true \rangle$ } { $\langle false \rangle$ } datatool-base

§2.4.1.1;
63

Does $\langle true \rangle$ if the first argument $\langle arg \rangle$ is a string (not numerical) otherwise does $\langle false \rangle$.

\DTLifstringclosedbetween { $\langle str \rangle$ } { $\langle min \rangle$ } { $\langle min \rangle$ } { $\langle true \rangle$ } { $\langle false \rangle$ }
modifier: * datatool-base

§2.4.1.2;
70

Does $\langle true \rangle$ if $\langle str \rangle$ lies between $\langle min \rangle$ and $\langle max \rangle$, inclusive, otherwise does $\langle false \rangle$.

\DTLifstringeq { *⟨str1⟩* } { *⟨str2⟩* } { *⟨true⟩* } { *⟨false⟩* }
datatool–base

modifier: *

§2.4.1.2;
70

Does *⟨true⟩* if *⟨str1⟩* is lexicographically equal to *⟨str2⟩* otherwise does *⟨false⟩*. The starred version ignores case.

\DTLifstringgt { *⟨str1⟩* } { *⟨str2⟩* } { *⟨true⟩* } { *⟨false⟩* }
datatool–base

modifier: *

§2.4.1.2;
70

Does *⟨true⟩* if *⟨str1⟩* is lexicographically greater than *⟨str2⟩* otherwise does *⟨false⟩*. The starred version ignores case.

\DTLifstringlt { *⟨str1⟩* } { *⟨str2⟩* } { *⟨true⟩* } { *⟨false⟩* }
datatool–base

modifier: *

§2.4.1.2;
70

Does *⟨true⟩* if *⟨str1⟩* is lexicographically less than *⟨str2⟩* otherwise does *⟨false⟩*. The starred version ignores case.

\DTLifstringopenbetween { *⟨str⟩* } { *⟨min⟩* } { *⟨max⟩* } { *⟨true⟩* } { *⟨false⟩* }
modifier: * datatool–base

§2.4.1.2;
70

Does *⟨true⟩* if *⟨str⟩* lies between *⟨min⟩* and *⟨max⟩*, but is not equal to *⟨min⟩* or *⟨max⟩*, otherwise does *⟨false⟩*.

\DTLifSubString { *⟨string⟩* } { *⟨fragment⟩* } { *⟨true⟩* } { *⟨false⟩* }
datatool–base v1.01+ modifier: *

§2.4.1.2;
70

Does *⟨true⟩* if *⟨fragment⟩* is a substring of *⟨string⟩* otherwise does *⟨false⟩*. The starred version is case-insensitive.

\DTLiftemporal { *⟨arg⟩* } { *⟨true⟩* } { *⟨false⟩* }
datatool–base v3.0+

§2.4.1.1;
63

Does *⟨true⟩* if the first argument *⟨arg⟩* is a temporal value otherwise does *⟨false⟩*.

\DTLinitialhyphen
initial: – datatool–base

§2.8.2;
142

Used by `\DTLinitials` and `\DTLstoreinitials` where a hyphen occurs.

\DTLinitialpunc{ $\langle letter \rangle$ }{ $\langle punc \rangle$ }

datatool-base v3.0+

§2.8.2;
141

Used by `\DTLstoreinitials` to markup initials.

\DTLinitials{ $\langle text \rangle$ }

datatool-base

§2.8.2;
140

Splits $\langle text \rangle$ into words and displays the initial of each word. Internally uses `\DTLstoreinitials`.

\dtlinsertinto{ $\langle element \rangle$ }{ $\langle sorted-list cs \rangle$ }{ $\langle criteria cs \rangle$ }

datatool-base v2.27+

§2.9.4;
155

Globally inserts $\langle element \rangle$ into the given sorted list according to the $\langle criteria cs \rangle$ handler macro.

\dtlintalign

initial: `r` datatool v2.0+

§3.7.2;
254

Expands to the column alignment specifier used by `\DTLdisplaydb` and `\DTLdisplaylongdb` for columns with the integer data type. Ignored if the `align-specs` option is set.

\dtlintformat{ $\langle text \rangle$ }

datatool v2.0+

§3.7.2;
252

Used by `\DTLdisplaydb` and `\DTLdisplaylongdb` to format entries in integer columns. Defined to use `\dtlnumericformat` by default.

\DTLinttype

datatool v2.0+

§3.6; 239

Expands to 1.

\DTLisclosedbetween{ $\langle num \rangle$ }{ $\langle min \rangle$ }{ $\langle min \rangle$ }

datatool-base

§2.4.2; 92

As the unstarred `\DTLifclosedbetween` but for use in the conditional part of commands like `\ifthenelse`.

\DTLiscurrency{ $\langle arg \rangle$ }

datatool-base

§2.4.2; 90

As `\DTLifcurrency` but for use in the conditional part of commands like `\ifthenelse`.

\DTLiscurrencyunit{ $\langle arg \rangle$ }

datatool-base

§2.4.2; 90

As `\DTLifcurrencyunit` but for use in the conditional part of commands like `\ifthenelse`.

\DTLiseq{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }

datatool-base

§2.4.2; 90

As the unstarred `\DTLifeq` but for use in the conditional part of commands like `\ifthenelse`.

\DTLisFPclosedbetween{ $\langle num \rangle$ }{ $\langle min \rangle$ }{ $\langle min \rangle$ }

datatool-base

§2.4.2; 93

Synonym of `\DTLisnumclosedbetween`.

\DTLisFPeq{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }

datatool-base

§2.4.2; 90

Synonym of `\DTLisnumeq`.

\DTLisFPgt{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }

datatool-base

§2.4.2; 92

Synonym of `\DTLisnumgt`.

\DTLisFPgteq{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }

datatool-base

§2.4.2; 92

Synonym of `\DTLisnumgteq`.

\DTLisFPlt{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }

datatool-base

§2.4.2; 91

Synonym of `\DTLisnumlt`.

\DTLisFPlteq{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }

datatool-base

§2.4.2; 91

Synonym of \DTLisnumlteq.

\DTLisFPopenbetween{ $\langle num \rangle$ }{ $\langle min \rangle$ }{ $\langle min \rangle$ }

datatool-base

§2.4.2; 92

Synonym of \DTLisnumopenbetween.

\DTLisgt{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }

datatool-base

§2.4.2; 91

As the unstarred \DTLifgt but for use in the conditional part of commands like \ifthenelse.

\DTLisiclosedbetween{ $\langle num \rangle$ }{ $\langle min \rangle$ }{ $\langle min \rangle$ }

datatool-base

§2.4.2; 92

As the starred \DTLifclosedbetween* but for use in the conditional part of commands like \ifthenelse.

\DTLisieq{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }

datatool-base v1.01+

§2.4.2; 90

As the starred \DTLifeq* but for use in the conditional part of commands like \ifthenelse.

\DTLisigt{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }

datatool-base v1.01+

§2.4.2; 91

As the starred \DTLifgt* but for use in the conditional part of commands like \ifthenelse.

\DTLisilt{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }

datatool-base v1.01+

§2.4.2; 91

As the starred \DTLiflt* but for use in the conditional part of commands like \ifthenelse.

\DTLisinlist{ $\langle element \rangle$ }{ $\langle list \rangle$ }

datatool-base

§2.4.2; 93

As \DTLifinlist but for use in the conditional part of commands like \ifthenelse.

\DTLisint{*<arg>*}

datatool-base

§2.4.2; 89

As \DTLifint but for use in the conditional part of commands like \ifthenelse.

\DTLisiopenbetween{*<num>*}{*<min>*}{*<min>*}

datatool-base

§2.4.2; 92

As the starred \DTLifopenbetween* but for use in the conditional part of commands like \ifthenelse.

\DTLisiPrefix{*<string>*}{*<fragment>*}

datatool-base v3.0+

§2.4.2; 93

As \DTLifStartsWith* but for use in the conditional part of commands like \ifthenelse.

\DTLisiSubString{*<string>*}{*<fragment>*}

datatool-base

§2.4.2; 93

As \DTLifSubString* but for use in the conditional part of commands like \ifthenelse.

\DTLisiSuffix{*<string>*}{*<fragment>*}

datatool-base v3.0+

§2.4.2; 93

As \DTLifEndsWith* but for use in the conditional part of commands like \ifthenelse.

\DTLislt{*<arg1>*}{*<arg2>*}

datatool-base

§2.4.2; 91

As the unstarred \DTLiflt but for use in the conditional part of commands like \ifthenelse.

\DTLisnumclosedbetween{*<num>*}{*<min>*}{*<min>*}

datatool-base

§2.4.2; 93

As \DTLifnumclosedbetween but for use in the conditional part of commands like \ifthenelse.

\DTLisnumeq{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }

datatool-base v3.0+

§2.4.2; 90

As `\DTLifnumeq` but for use in the conditional part of commands like `\ifthenelse`.

\DTLisnumerical{ $\langle arg \rangle$ }

datatool-base

§2.4.2; 90

As `\DTLifnumerical` but for use in the conditional part of commands like `\ifthenelse`.

\DTLisnumgt{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }

datatool-base v3.0+

§2.4.2; 91

As `\DTLifnumgt` but for use in the conditional part of commands like `\ifthenelse`.

\DTLisnumgteq{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }

datatool-base v3.0+

§2.4.2; 92

Numerical “less than or equal to” for use in the conditional part of commands like `\ifthenelse`.

\DTLisnumlt{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }

datatool-base v3.0+

§2.4.2; 91

As `\DTLifnumlt` but for use in the conditional part of commands like `\ifthenelse`.

\DTLisnumlteq{ $\langle arg1 \rangle$ }{ $\langle arg2 \rangle$ }

datatool-base v3.0+

§2.4.2; 91

Numerical “less than or equal to” for use in the conditional part of commands like `\ifthenelse`.

\DTLisnumopenbetween{ $\langle num \rangle$ }{ $\langle min \rangle$ }{ $\langle min \rangle$ }

datatool-base

§2.4.2; 92

As `\DTLifnumopenbetween` but for use in the conditional part of commands like `\ifthenelse`.

\DTLisopenbetween{ $\langle num \rangle$ }{ $\langle min \rangle$ }{ $\langle min \rangle$ }

datatool-base

§2.4.2; 92

As the unstarred `\DTLifopenbetween` but for use in the conditional part of commands like `\ifthenelse`.

\DTLisPrefix{*<string>*}{*<fragment>*}

datatool-base

§2.4.2; 93

As `\DTLifStartsWith` but for use in the conditional part of commands like `\ifthenelse`.

\DTLisreal{*<arg>*}

datatool-base

§2.4.2; 90

As `\DTLifreal` but for use in the conditional part of commands like `\ifthenelse`.

\DTLisstring{*<arg>*}

datatool-base

§2.4.2; 90

As `\DTLifstring` but for use in the conditional part of commands like `\ifthenelse`.

\DTLisSubString{*<string>*}{*<fragment>*}

datatool-base

§2.4.2; 93

As `\DTLifSubString` but for use in the conditional part of commands like `\ifthenelse`.

\DTLisSuffix{*<string>*}{*<fragment>*}

datatool-base v3.0+

§2.4.2; 93

As `\DTLifEndsWith` but for use in the conditional part of commands like `\ifthenelse`.

\dtlkey

datatool

§3.8.2;
297

Defined by `\DTLforeachkeyinrow` to the current column key.

\dtllastloadeddb

datatool

§3.15.3;
360

Expands to the label of the last database to be loaded by `\DTLread`.

\DTLlegendxoffset

initial: 10pt dataplot

§6.4.1;
520

Dimension that specifies the gap between the border of the plot and the legend for the *x*-axis.

\DTLlegendyoffset*initial:* 10pt dataplot§6.4.1;
520

Dimension that specifies the gap between the border of the plot and the legend for the *y*-axis.

\dtllettergroup{*<character>*}

datatool-base v3.0+

§2.3.3; 52

Used by \DTLassignlettergroup to identify alphabetical letter groups.

\dtlletterindexcompare{*<count-reg>*}{*<string1>*}{*<string2>*}

datatool-base

§2.9.5.1;
163

Letter order comparison that may be used in \dtlsortlist.

\DTLlistand

datatool-base v3.0+

§2.9.2;
152

Used in the default definition of \DTLlistformatlastsep. Expands either to \DTL-andname or to \&, depending on the *and* option in the *lists* setting.

\DTLlistelement{*<list>*}{*<idx>*}

datatool-base

§2.9.3;
153

Does the *<idx>*th element in the list (starting with 1 for the first element).

\DTLlistformatitem{*<item>*}

datatool-base v2.28+

§2.9.2;
151

Formats an item within \DTLformatlist.

\DTLlistformatlastsep

datatool-base v2.28+

§2.9.2;
151

Separator used between the final two elements in \DTLformatlist.

\DTLlistformatoxford*initial:* empty datatool-base v2.28+§2.9.2;
152

Inserted before \DTLlistformatlastsep if the list contains more than two items.

\DTLlistformatsep {*item*}*initial:* , □ datatool–base v2.28+§2.9.2;
151

Separator used between each item except for the final two within `\DTLformatlist`.

\DTLloadbbl [*bbl file*] {*db-name*} {*bib-list*}

databib

§7.3; 536

Does `\bibliographystyle{databib}`, creates a new database called *db-name* (and defines the placeholder command `\DTLBIBdbname` to *db-name*), and inputs the given *bbl file*, which is expected to be in the format created by the `databib.bst` bibtex style file. The *bib-list* identifies the list of bib files that bibtex needs to read in order to create the *bbl file*. If the optional argument is omitted, `\jobname.bbl` is assumed. If the *db-name* argument is empty, the default database name will be used.

 **\DTLloaddb** [*options*] {*db-name*} {*filename*}

datatool

§3.15.3;
364

Loads a CSV / TSV file according to the current delimiter and separator and globally defines a database labelled *db-name*.

 **\DTLloaddbtex** {*cs*} {*filename*}

datatool v2.20+

§3.15.3;
363

Inputs a dbtex file.

\DTLloadmbbl {mbib} {*db-name*} {*bib-list*}

databib

§7.9; 553

Similar to `\DTLloadbbl` but for multiple bibliographies.

 **\DTLloadrawdb** [*options*] {*db-name*} {*filename*}

datatool

§3.15.3;
364

Loads a CSV / TSV file where T_EX special characters should be interpreted literally according to the current delimiter and separator and globally defines a database labelled *db-name*.

\DTLmajorgridstyle*initial:* color=gray, – dataplot§6.4.3;
524

Expands to the tikz option to use for the major grid line style.

\dtlmapcurrentrow { *<cs>* } { *<body>* }

datatool v3.0+

§3.16.1;
374

Iterates over the columns of the current row (as stored in `\dtlcurrentrow`) and, at each iteration, defines the control sequence *<cs>* to the current column and does *<body>*.

\DTLmapdata [*<key=value list>*] { *<loop-body>* }

datatool v3.0+

§3.8.1

Iterates over each row in the given database and does *<loop-body>*. Within the loop body, the row index is stored in the register `\dtlrownum`. Values can be accessed with `\DTLmapget` or iterated over with `\DTLmaprow`.

\DTLmapatabreak

modifier: * datatool v3.0+

§3.8.1;
286

Breaks out of `\DTLmapdata` and `DTLenvmapdata`. The starred version will discard any pending edits.

\DTLmapget { *<key=value list>* }

datatool v3.0+

§3.8.1.1;
287

For use within `\DTLmapdata` (or `DTLenvmapdata`) to access a value in the current iteration row.

\DTLmapgetvalues { *<assign-list>* }

modifier: * datatool v3.0+

§3.8.1.1;
289

For use within `\DTLmapdata` (or `DTLenvmapdata`) to access values in the current iteration row. The starred form won't trigger an error if a column key in *<assign-list>* isn't defined.

\DTLmaprow { *<cs>* } { *<body>* }

datatool v3.0+

§3.8.1.1;
288

Only for use within the *<loop-body>* argument of `\DTLmapdata` or the environment body of `DTLenvmapdata`, this command iterates over each column in the current `\DTLmapdata` row, sets `\dtlcolumnnum` to the column index and *<cs>* to the column value and does *<body>*.

\DTLmaprowbreak

datatool v3.0+

§3.8.1.1;
289

Breaks out of `\DTLmaprow`.

\DTLmax {*cs*} {*num1*} {*num2*}

datatool-base

§2.5.2;
113

Sets *cs* to the maximum of *num1* and *num2*, where the numbers are formatted numbers. The result will be a formatted number.

\dtlmax {*cs*} {*num1*} {*num2*}

datatool-base

§2.5.1;
100

Defines the control sequence *cs* to the larger of the two numbers, where the numbers are plain numbers.

\DTLmaxall {*cs*} {*num list*}

datatool-base

§2.5.2;
113

Sets *cs* to the maximum of all the formatted numbers in the comma-separated *num list*. The result will be a formatted number.

\dtlmaxall {*cs*} {*num list*}

datatool-base v3.0+

§2.5.1;
100

Sets *cs* to the maximum of all the plain numbers in the comma-separated *num list*.

\DTLmaxforcolumn {*db*} {*key*} {*cmd*}

datatool

§3.13; 324

Determines the maximum value over all numeric values in the column with the label *key* of the database identified by *db* and stores the result as a formatted number in the control sequence *cmd*.

\DTLmaxforkeys [*condition*] [*assign-list*] {*db list*} {*key list*} {*cmd*}

datatool

§3.13; 323

Determines the maximum value over all numeric values in the listed columns in the given databases and stores the result as a formatted number in the control sequence *cmd*. The optional arguments may be used to skip rows where the condition evaluates to false.

\DTLmaxX

dataplot

§6.3.1;
511

Placeholder for use in `\DTLplot` hooks, this expands to the maximum *x* value of the plot bounds.

\DTLmaxY

dataplot

§6.3.1;
512

Placeholder for use in `\DTLplot` hooks, this expands to the maximum y value of the plot bounds.

\DTLmbibitem

datbib

§7.7.1;
546

Used within `\DTLmbibliography` at the start of each iteration.

\DTLmbibliography [$\langle condition \rangle$] { $\langle mbib name \rangle$ } { $\langle db-name \rangle$ }

datbib

§7.9; 553

Similar to `\DTLbibliography` but for the given multi-bib database, which should have been loaded with `\DTLloadmbbl`.

\DTLmeanforall { $\langle cs \rangle$ } { $\langle num list \rangle$ }

datatool-base

§2.5.2;
113

Sets $\langle cs \rangle$ to the mean (average) of all the formatted numbers in the comma-separated $\langle num list \rangle$. The result will be a formatted number.

\dtlmeanforall { $\langle cs \rangle$ } { $\langle num list \rangle$ }

datatool-base v3.0+

§2.5.1;
101

Calculates the mean (average) of all the numbers in the comma-separated list $\langle num list \rangle$ and stores the result in the control sequence $\langle cs \rangle$, where the numbers are plain numbers.

\DTLmeanforcolumn { $\langle db \rangle$ } { $\langle key \rangle$ } { $\langle cmd \rangle$ }

datatool

§3.13; 322

Computes the mean (average) over all numeric values in the column with the label $\langle key \rangle$ of the database identified by $\langle db \rangle$ and stores the result as a formatted number in the control sequence $\langle cmd \rangle$.

\DTLmeanforkeys [$\langle condition \rangle$] [$\langle assign-list \rangle$] { $\langle db list \rangle$ } { $\langle key list \rangle$ } { $\langle cmd \rangle$ }

datatool

§3.13; 322

Computes the mean (average) over all numeric values in the listed columns in the given databases and stores the result as a formatted number in the control sequence $\langle cmd \rangle$. The optional arguments may be used to skip rows where the condition evaluates to false.

\DTLmidpt

databar

§5.4.4;
449

For use in the definition of `\DTLeverybarhook`, this expands to the mid point of the current bar as a pgf point.

\DTLmin {*cs*} {*num1*} {*num2*}

datatool-base

§2.5.2;
112

Sets *cs* to the minimum of *num1* and *num2*, where the numbers are formatted numbers. The result will be a formatted number.

\dtlmin {*cs*} {*num1*} {*num2*}

datatool-base

§2.5.1;
100

Defines the control sequence *cs* to the smaller of the two numbers, where the numbers are plain numbers.

\DTLminall {*cs*} {*num list*}

datatool-base

§2.5.2;
113

Sets *cs* to the minimum of all the formatted numbers in the comma-separated *num list*. The result will be a formatted number.

\dtlminall {*cs*} {*num list*}

datatool-base v3.0+

§2.5.1;
100

Sets *cs* to the minimum of all the plain numbers in the comma-separated *num list*.

\DTLminforcolumn {*db*} {*key*} {*cmd*}

datatool

§3.13; 323

Determines the minimum value over all numeric values in the column with the label *key* of the database identified by *db* and stores the result as a formatted number in the control sequence *cmd*.

\DTLminforkeys [*condition*] [*assign-list*] {*db list*} {*key list*} {*cmd*}

datatool

§3.13; 323

Determines the minimum value over all numeric values in the listed columns in the given databases and stores the result as a formatted number in the control sequence *cmd*. The optional arguments may be used to skip rows where the condition evaluates to false.

\DTLminminortickgap

initial: 5pt

§6.4.1;
520

Dimension used to obtain the suggested minimum minor tick gap when not otherwise specified.

\DTLminorgridstyle

initial: color=lightgray,very thin

dataplot

§6.4.3;
524

Expands to the tikz option to use for the minor grid line style. If this is redefined to expand to nothing, the minor grid lines won't be drawn.

\DTLminorticklength

initial: 2pt dataplot

§6.4.1;
520

Dimension that specifies length of the minor tick marks.

\DTLmintickgap

initial: 20pt

§6.4.1;
520

Dimension used to obtain the suggested minimum tick gap if not overridden by `x-tick-gap` or `y-tick-gap` or `x-tick-points` or `y-tick-points`.

\DTLminX

dataplot

§6.3.1;
511

Placeholder for use in `\DTLplot` hooks, this expands to the minimum x value of the plot bounds.

\DTLminY

dataplot

§6.3.1;
511

Placeholder for use in `\DTLplot` hooks, this expands to the minimum y value of the plot bounds.

\DTLmonthname { *month num* }

datbib

§7.3; 537

Formats the month name according to the current style.

\DTLmul {*cs*} {*num1*} {*num2*}

datatool-base

§2.5.2;
110

Calculates $\langle num1 \rangle \times \langle num2 \rangle$ and stores the result in the control sequence *cs*, where the numbers are formatted numbers. The result will be a formatted number.

\dtlmul {*cs*} {*num1*} {*num2*}

datatool-base

§2.5.1; 99

Calculates $\langle num1 \rangle \times \langle num2 \rangle$ and stores the result in the control sequence *cs*, where the numbers are plain numbers.

\DTLmultibarchart [*condition*] {*settings-list*} {*db-name*} {*assign-list*}

databar

§5; 400

Draws a multi-bar bar chart using data from the database identified by *db-name*. The *settings-list* must include the `variables` setting to identify the placeholder commands *assign-list* that will be assigned to the numeric values for the chart.

\DTLmultibibs {*list*}

(preamble only)

databib

§7.9; 552

Creates an auxiliary file for each name in the given *list*. Each element in the list should be the base name without the extension.

\DTLneg {*cs*} {*num*}

datatool-base

§2.5.2;
111

Calculates the negative of the formatted number *num* and stores the result as a formatted number in the control sequence *cs*.

\dtlneg {*cs*} {*num*}

datatool-base

§2.5.1;
101

Defines the control sequence *cs* to the negative of the number *num*, where the number is a plain number.

\DTLnegextent

initial: empty databar

§5.4.1;
442

The maximum depth (negative) of the *y* axis or empty if the maximum depth in the data should

be used. The expansion text must be either empty (for the default) or a decimal number less than or equal to 0.

\DTLnewbibitem{*<col-key>*}{*<item>*}

datbib

§7.4; 537

Intended for use in the bbl loaded by \DTLloadbbl, this adds a new value to the last row in the database identified by \DTLBIBdbname (without checking for existence). Internally uses \DTLnewdbentry*, so it obeys the general database options.

\DTLnewbibliteralitem{*<col-key>*}{*<item>*}

datbib v3.0+

§7.4; 538

As \DTLnewbibitem but the item has literal content.

\DTLnewbibrow

datbib

§7.4; 537

Intended for use in the bbl loaded by \DTLloadbbl, this creates a new row in the database identified by \DTLBIBdbname (without checking for existence). Internally uses \DTLnewrow*, so it obeys the general database options.

\DTLnewcurrencysymbol{*<symbol>*}

datatool-base

§2.6; 123

Adds *<symbol>* to the list of known currencies.

\DTLnewdb{*<db-name>*}

datatool

§3.4; 232

Creates a new database with the label *<db-name>*. Note that new databases are always global, regardless of the `global` option, so this command is equivalent to \DTLgnewdb.

\DTLnewdbentry{*<db-name>*}{*<col key>*}{*<value>*}

modifier: * datatool

§3.4; 232

Adds an entry to the last row of the database identified by the label *<db-name>* for the column identified by its label *<col key>*. The unstarred version checks the database exists. The starred version doesn't. Both versions will trigger an error if the row already contains an entry for the given column.

\DTLnewrow{*<db-name>*}

modifier: * datatool

§3.4; 232

Adds a new row to the database identified by the label *<db-name>*. The unstarred version checks the database exists. The starred version doesn't.

\DTLnocite{*<mbib>*}{*<label-list>*}

datatool

§7.9; 553

As \DTLcite but analogous to \nocite.

\dtlnoexpandnewvalue

datatool

§3.1; 179

Prevents new values from being expanded when added to a database (equivalent to the `new-value-expand=false` option). Has no effect when loading `dbtex` files. Note that values can still be expanded with the `expand-value` and `expand-once-value` action settings.

\dtlnonlettergroup{*<character>*}

datatool-base v3.0+

§2.3.3; 53

Used by \DTLassignlettergroup to identify non-letter groups.

\dtlnovalue

datatool-base

§3.10.2;
315

Used to represent a null value (see §3.10). Prior to version 3.0, this was defined by `datatool` rather than `datatool-base`.

\dtlnumbergroup{*<num>*}

datatool-base v3.0+

§2.3.3; 53

Used by \DTLassignlettergroup to identify number groups.

\DTLnumbernull

datatool-base

§3.10.2;
316

Used to represent a null value for a numeric column (see §3.10). Prior to version 3.0, this was defined by `datatool` rather than `datatool-base`.

\DTLnumcompare { *<count-reg>* } { *<num1>* } { *<num2>* }

datatool-base v3.0+

§2.9.5.1;
164

Compares *<num1>* and *<num2>*, where the numbers are formatted numbers or datum control sequences, and sets the count register (integer variable) *<count-reg>* to -1 if *<num1>* is less than *<num2>*, to 0 if *<num1>* and *<num2>* are equal or to $+1$ if *<num1>* is greater than *<num2>*.

\dtlnumericformat { *<text>* }

datatool v3.0+

§3.7.2;
252

Used by `\DTLdisplaydb` and `\DTLdisplaylongdb` to format entries in numeric columns.

\DTLnumitemsinlist { *<list>* } { *<cs>* }

datatool-base

§2.9.3;
153

Counts the number of elements in *<list>* and defines the command *<cs>* to that number.

\dtlpadleadingzeros { *<num-digits>* } { *<value>* }

datatool-base v3.0+

§2.5.1; 98

Displays the plain number *<value>* zero-padded to *<num-digits>*. This command is designed for sorting numbers by character code and so is expandable. Unlike `\two@digits`, the *<value>* may be a decimal. The expansion is also a plain number. The *<num-digits>* argument should be in the range 1–7.

\dtlpadleadingzerosminus*initial:* $-$ datatool-base v3.0+

§2.5.1; 98

Inserted at the start of the expansion text of `\dtlpadleadingzeros` if the value is negative.

\dtlpadleadingzerosplus*initial:* *empty* datatool-base v3.0+

§2.5.1; 98

Inserted at the start of the expansion text of `\dtlpadleadingzeros` if the value is positive.

\DTLpar

datatool

§3.4; 235

Robust command defined to do `\par`. For use in “short” arguments where `\par` isn’t permitted.

\DTLparse { *<cs>* } { *<content>* }

datatool–base v3.0+

§2.2.3; 16

Parses *<content>* and stores the data in the control sequence *<cs>*, which can then be passed to `\DTLusedatum`, `\DTLdatumvalue`, `\DTLdatumcurrency` or `\DTLdatum-type`.

\DTLpieatbegintikz

datapie

§4.7; 398

Hook used by `\DTLpiechart` at the start of the `tikzpicture` environment.

\DTLpieatendtikz

datapie

§4.7; 398

Hook used by `\DTLpiechart` at the end of the `tikzpicture` environment.

\DTLpieatsegment { *<index>* } { *<total>* } { *<start angle>* } { *<mid angle>* } { *<end angle>* } { *<shift angle>* } { *<shift offset>* } { *<inner offset>* } { *<outer offset>* } datapie v3.0+

§4.7; 397

Hook used at each segment of the pie chart.

\DTLpiechart [*<condition>*] { *<settings-list>* } { *<db-name>* } { *<assign-list>* } datapie

§4; 377

Draws a pie chart using data from the database identified by *<db-name>*. The *<settings-list>* must include the `variable` setting to identify the placeholder command included in *<assign-list>* that will be assigned to the numeric values for the chart.

\DTLpieoutlinecolor

initial: black datapie

§4.6; 397

Expands to the colour name for the segment outline.

\DTLpieoutlinewidth

initial: 0pt datapie

§4.6; 397

Length register that stores the width of the segment outline.

\DTLpiepercent

datapie

§4.4; 394

Placeholder command that may be used in inner or outer labels to show the percent value.

\DTLpievariable

datapie

§4.4; 394

Placeholder command that may be used in inner or outer labels to show the actual value.

\DTLplot [*⟨condition⟩*] { *⟨db-list⟩* } { *⟨key=value list⟩* }

dataplot

§6; 451

Plots the data from the listed databases as line or scatter diagrams.

\DTLplotatbegintikz

dataplot

§6.3; 510

Hook used at the start of the tikzpicture environment within \DTLplot.

\DTLplotatendtikz

dataplot

§6.3; 511

Hook used at the end of the tikzpicture environment within \DTLplot.

\DTLplotdisplayticklabel { *⟨text⟩* }

dataplot v3.0+

§6.4.3;
523

Used by both \DTLplotdisplayXticklabel and \DTLplotdisplayYticklabel to format the *x* and *y* tick labels.

\DTLplotdisplayXticklabel { *⟨text⟩* }

dataplot v3.0+

§6.4.3;
523

Used to format *x* tick labels.

\DTLplotdisplayYticklabel { *⟨text⟩* }

dataplot v3.0+

§6.4.3;
523

Used to format *y* tick labels.

\dtlplotohandlermark { *mark code* }

dataplot v2.15+

§6.5; 524

Only for use within the `\DTLplot` hooks, this resets the pgf transformation matrix and uses `\pgfplotohandlermark` { *mark code* }.

\DTLplotheight

initial: 4in dataplot

§6.4.1;
521

Dimension that controls the plot height.

\DTLplotlegendname [*db-index*] { *db-name* }

dataplot v3.0+

§6.3.3;
516

If there more than one database is supplied for the plot or if there is only a single *x* and *y* key, the legend will include this command to typeset the database name or a mapping that was previously provided with `\DTLplotlegendsetname`.

\DTLplotlegendnamesep

initial: □ dataplot v3.0+

§6.3.3;
517

If more than one database was specified, this command will be placed after `\DTLplotlegendname`.

\DTLplotlegendsetname { *db-name* } { *text* }

dataplot v3.0+

§6.3.3;
516

Provides a mapping from a database name to *text*, which `\DTLplotlegendname` should use instead of *db-name*.

\DTLplotlegendsetxlabel { *x-key* } { *text* }

dataplot v3.0+

§6.3.3;
517

Provides a mapping from a column key to *text*, which `\DTLplotlegendx` should use instead of the column header.

\DTLplotlegendsetylabel { *y-key* } { *text* }

dataplot v3.0+

§6.3.3;
518

Provides a mapping from a column key to *text*, which `\DTLplotlegendy` should use instead of the column header.

\DTLplotlegendx [*<db-index>*] {*<db-name>*} [*<x-index>*] {*<x-key>*} dataplot v3.0+

§6.3.3;
517

Used by `\DTLplotlegendxy` show the *x* label in the legend. The default definition is to show the header for the column identified by *<x-key>* unless a mapping has been provided with `\DTLplotlegendsetxlabel`. The optional arguments are ignored.

\DTLplotlegendxy [*<db-index>*] {*<db-name>*} [*<x-index>*] {*<x-key>*} [*<y-index>*] {*<y-key>*} dataplot v3.0+

§6.3.3;
517

If there are multiple keys listed in *x*, this command will be used to show both the *x* and *y* headers in the legend.

\DTLplotlegendxysep dataplot v3.0+

§6.3.3;
517

If there are more than one column keys in *x*, this command will be used to separate the *x* label from the *y* label. The default definition is a slash with a space on either side.

\DTLplotlegendy [*<db-index>*] {*<db-name>*} [*<y-index>*] {*<y-key>*} dataplot v3.0+

§6.3.3;
518

The legend will show the *y* text using this command if the *y* option had more than one key. The default definition is to show the header for the column identified by *<y-key>* unless a mapping has been provided with `\DTLplotlegendsetylabel`. The optional arguments are ignored.

\DTLplotlinecolors dataplot

§6.4.3;
521

Expands to a comma-separated list of colour specifications for plot lines.

\DTLplotlines dataplot

§6.4.3;
522

Expands to a comma-separated list of plot line styles.

\DTLplotmarkcolors dataplot

§6.4.3;
522

Expands to a comma-separated list of colour specifications for plot marks.

\DTLplotmarks

dataplot

§6.4.3;
522

Expands to a comma-separated list of plot marks.

\DTLplotstream [*⟨condition⟩*] { *⟨db-name⟩* } { *⟨x-key⟩* } { *⟨y-key⟩* }

dataplot

§6.5; 524

Adds data from columns identified by *⟨x-key⟩* and *⟨y-key⟩* from the given database to the current pgf plot stream.

\DTLplotwidth

initial: 4in dataplot

§6.4.1;
521

Dimension that controls the plot width.

\DTLpostpagename

initial: ~ databib v3.0+

§7.7.1;
546

Space inserted after *\pagename* or *\pagesname* (before the page number).

\DTLpostvon

initial: ~ databib v3.0+

§7.7.1;
544

Space to insert after the *⟨von part⟩* of a name.

\DTLPreProcessCurrencyGroup { *⟨sym-cs⟩* } { *⟨num-cs⟩* } { *⟨actual⟩* }

datatool-base v3.0+

§2.9.5;
158

Used by *\DTLassignlettergroup* to pre-process the currency group, where *⟨num-cs⟩* contains the numeric value and *⟨sym-cs⟩* contains the currency symbol. This is performed before encapsulation with *\dtlcurrencygroup*.

\DTLPreProcessDecimalGroup { *⟨cs⟩* } { *⟨actual⟩* }

datatool-base v3.0+

§2.9.5;
158

Used by *\DTLassignlettergroup* to pre-process the (decimal) numeric group stored in the given token list variable *⟨cs⟩*. This is performed before encapsulation with *\dtlnumbergroup*.

\DTLPreProcessIntegerGroup { *<cs>* } { *<actual>* }

datatool-base v3.0+

§2.9.5;
157

Used by \DTLassignlettergroup to pre-process the (integer) numeric group stored in the given token list variable *<cs>*. This is performed before encapsulation with \dtlnumbergroup.

\DTLPreProcessLetterGroup { *<cs>* }

datatool-base v3.0+

§2.9.5;
158

Used by \DTLassignlettergroup to pre-process the (alphabetical) letter group stored in the given token list variable *<cs>*. This is performed after \DTLCurrentLocaleGetInitialLetter and before encapsulation with \dtllettergroup.

\DTLPreProcessNonLetterGroup { *<cs>* }

datatool-base v3.0+

§2.9.5;
158

Used by \DTLassignlettergroup to pre-process the (symbol/punctuation) non-letter group stored in the given token list variable *<cs>*. This is performed before encapsulation with \dtlnonlettergroup.

 **\DTLprotectedsaverawdb** { *<db-name>* } { *<filename>* }

datatool v2.15+

§3.15.4;
365

Similar to \DTLsaverawdb but designed for database with fragile commands.

\DTLrawmap { *<original>* } { *<replacement>* }

datatool

§3.15.2;
350

Appends a mapping used by the `csv-content=literal` setting.

\DTLread [*<options>*] { *<filename>* }

datatool v3.0+

§3.15.3;
360

Loads the data in *<filename>* to create a new database or, if applicable, append to an existing database.

\dtlrealalign

initial: `r` datatool v2.0+

§3.7.2;
254

Expands to the column alignment specifier used by \DTLdisplaydb and \DTLdisplaylongdb for columns with the decimal (real) data type. Ignored if the `align-specs` option is set.

\dtlrealformat { *<text>* }

datatool v2.0+

§3.7.2;
252

Used by `\DTLdisplaydb` and `\DTLdisplaylongdb` to format entries in decimal columns. Defined to use `\dtlnumericformat` by default.

\DTLrealtype

datatool v2.0+

§3.6; 239

Expands to 2.

\dtlrecombine

datatool

§3.16.1;
373

Recombines the database (identified by `\dtldbname`) from `\dtlbeforerow`, `\dtl-currentrow` and `\dtlafterrow`. This command checks the `global` option.

\dtlrecombineomitcurrent

datatool

§3.16.1;
373

As `\dtlrecombine` but omits the current row (that is, the current row is deleted from the database).

\DTLreconstructdatabase { *<num-rows>* } { *<num-columns>* } { *<header code>* } { *<body code>* } { *<key-index code>* }

datatool v3.0+

§3.15.1.3;
344

Used in a DBTEX v3.0 file to construct the database.

 **\DTLreconstructdbdata** { *<header code>* } { *<body code>* } { *<num-rows>* } { *<num-columns>* } { *<key-index code>* }

datatool v3.0+

Used in experimental version of DBTEX v3.0 file to construct the database. This command has been replaced with `\DTLreconstructdatabase`.

\DTLremovecurrentrow

datatool

§3.8.2.1;
298

May only be used within the unstarred `\DTLforeach`, this command will remove the current row.

\DTLremoveentryfromrow{*<col-key>*}

datatool

§3.8.2.1;
297

May only be used within the unstarred `\DTLforeach`, this command will remove the entry in the current row for the column identified by *<col-key>*.

\dtlremoveentryincurrentrow{*<col idx>*}

datatool v2.10+

§3.16.1;
374

Locally removes the entry for the column identified by the index *<col idx>* from `\dtlcurrentrow`.

\DTLreplaceentryforrow{*<col-key>*}{*<value>*}

datatool

§3.8.2.1;
297

May only be used within the unstarred `\DTLforeach`, this command will replace the entry in the current row with the given value for the column identified by *<col-key>*.

\dtlreplaceentryincurrentrow{*<new value>*}{*<col idx>*} datatool v2.10+

§3.16.1;
374

Replaces the entry for the column identified by the index *<col idx>* in `\dtlcurrentrow` with the given value *<new value>*. The column data is updated according to the new value honouring the `global` option (but `\dtlcurrentrow` will only be locally changed).

\DTLresetLanguage

datatool-base v3.0+

§2.3; 28

Resets all language (not region) sensitive commands back to their default definitions (but only those defined by datatool-base). Commands provided by supplementary packages are not affected.

\DTLresetRegion

datatool-base v3.0+

§2.3; 28

Resets all region sensitive commands back to their default definitions.

\DTLrmentry{*<key=value list>*}

datatool v3.0+

§3.8.1.2;
290

Only available with `\DTLmapdata` and `read-only=false`, this command will remove a column from the current iteration row.

\DTLrmrow

datatool v3.0+

§3.8.1.2;
290

Only available with `\DTLmapdata` and `read-only=false`, this command will remove the current iteration row.

\dtlroot { *cs* } { *num* } { *n* }

datatool-base

§2.5.1; 99

Calculates the *n*th root of *num* and stores the result in the control sequence *cs*, where the number is a plain number.

\DTLround { *cs* } { *num* } { *num digits* }

datatool-base

§2.5.2;
111

Rounds the formatted number *num* to *num digits* and stores the result as a formatted number in the control sequence *cs*.

\dtlround { *cs* } { *num* } { *dp* }

datatool-base

§2.5.1; 99

Rounds *num* to *dp* decimal places and stores the result in the control sequence *cs*, where the number is a plain number.

\DTLrowcount { *db-name* }

datatool

§3.6; 237

Expands to the row count of the database with the label *db-name*.

\DTLrowincr

datatool v3.0+

§3.8.2;
295

Increments the counter `DTLrow`*n* where *n* is one more than `\dtlforeachlevel`.

\dtlrownum

datatool v2.0+

§3.16.1;
371

A count register used by various commands to keep track of a row number, which may be used by hooks.

\DTLrowreset

datatool v3.0+

§3.8.2;
295

Increments DTLrow and resets the counter DTLrow $\langle n \rangle$ where $\langle n \rangle$ is one more than \dtlforeachlevel.

 **\DTLsavedb** { $\langle db-name \rangle$ } { $\langle filename \rangle$ }

datatool v2.13+

§3.15.4;
364

Saves the given database as a CSV file.

\DTLsavelastrowcount { $\langle cs \rangle$ }

datatool

§3.8.2;
294

Saves the final loop index of the most recent \DTLforeach in $\langle cs \rangle$.

 **\DTLsaverawdb** { $\langle db-name \rangle$ } { $\langle filename \rangle$ }

datatool v2.13+

§3.15.4;
364

Saves the given database in its internal form.

 **\DTLsavetexdb** { $\langle db-name \rangle$ } { $\langle filename \rangle$ }

datatool

§3.15.4;
365

Saves the given database in DTLTEX v2.0 format.

\DTLscinum { $\langle value \rangle$ }

datatool-base v3.0+

§2.2.1; 14

If siunitx is loaded, this will expand to \num{value} otherwise it will just expand to $\langle value \rangle$.

\DTLsdforall { $\langle cs \rangle$ } { $\langle num list \rangle$ }

datatool-base

§2.5.2;
114

Sets $\langle cs \rangle$ to the standard deviation of all the formatted numbers in the comma-separated $\langle num list \rangle$. The result will be a formatted number.

\dtlsdforall [$\langle mean \rangle$] { $\langle cs \rangle$ } { $\langle num list \rangle$ }

datatool-base v3.0+

§2.5.1;
101

Calculates the standard deviation of all the numbers in the comma-separated list $\langle num list \rangle$ and stores the result in the control sequence $\langle cs \rangle$, where the numbers are plain numbers. If the mean

value has already been computed, it can be supplied in the optional argument *⟨mean⟩*. If omitted, the mean will be calculated before calculating the standard deviation.

\DTLsdforcolumn { *⟨db⟩* } { *⟨key⟩* } { *⟨cmd⟩* }

datatool

§3.13; 323

Computes the standard deviation over all numeric values in the column with the label *⟨key⟩* of the database identified by *⟨db⟩* and stores the result as a formatted number in the control sequence *⟨cmd⟩*.

\DTLsdforkeys [*⟨condition⟩*] [*⟨assign-list⟩*] { *⟨db list⟩* } { *⟨key list⟩* } { *⟨cmd⟩* }

datatool

§3.13; 323

Computes the standard deviation over all numeric values in the listed columns in the given databases and stores the result as a formatted number in the control sequence *⟨cmd⟩*. The optional arguments may be used to skip rows where the condition evaluates to false.

\DTLsetbarcolor { *⟨index⟩* } { *⟨colour⟩* }

databar

§5.4.3;
446

Sets the colour for the given index in the bar colour list.

\DTLsetcurrencydatum { *⟨cs⟩* } { *⟨formatted value⟩* } { *⟨value⟩* } { *⟨currency symbol⟩* }

datatool–base v3.0+

§2.2.3; 19

Defines *⟨cs⟩* to a currency datum.

\DTLsetdecimaldatum { *⟨cs⟩* } { *⟨formatted value⟩* } { *⟨value⟩* } datatool–base v3.0+

§2.2.3; 18

Defines *⟨cs⟩* to a decimal datum.

\DTLsetdefaultcurrency { *⟨ISO or symbol⟩* }

datatool–base v3.0+

§2.3.2; 44

Sets the default currency.

\DTLsetdelimiter { *⟨char⟩* }

datatool

§3.15.2;
353

Sets the delimiter for CSV files to *⟨char⟩*.

\DTLsetentry { *<key=value list>* }

datatool v3.0+

§3.8.1.2;
291

Only available with \DTLmapdata and `read-only=false`, this command will set a column in the current iteration row.

\DTLsetfpdatum { *<cs>* } { *<formatted value>* } { *<value>* }

datatool-base v3.0+

§2.2.3; 19

Defines *<cs>* to a decimal datum where the value is converted into `\datatool_datum_fp:nnn` { *<value>* } { *<fp>* } { *<decimal>* } for easy conversion to an l3fp variable.

\DTLsetheader { *<db-name>* } { *<col key>* } { *<header>* }

modifier: * datatool

§3.6; 240

Sets the header for the column given by *<col key>* in the database identified by *<db-name>*. The starred version doesn't check if the database exists.

\DTLsetintegerdatum { *<cs>* } { *<formatted value>* } { *<value>* }

datatool-base v3.0+

§2.2.3; 18

Defines *<cs>* to an integer datum.

\DTLsetLocaleOptions [*<parent module(s)>*] { *<module(s)>* } { *<key=value list>* }

modifier: * datatool-base v3.0+

§2.3; 30

Set localisation options for all parent/module combinations.

\DTLsetnegbarcolor { *<index>* } { *<colour>* }

databar

§5.4.3;
447

Sets the colour for the given index in the negative bar colour list.

\DTLsetnumberchars { *<number group char>* } { *<decimal char>* }

datatool-base

§2.3.2; 42

Sets the current number group character and decimal character.

\DTLsetpiesegmentcolor { *<n>* } { *<colour>* }

datapie

§4.6; 396

Assigns the colour for the *<n>*th segment.

\DTLsetseparator { *char* }

datatool

§3.15.2;
359

Sets the separator for CSV files to *char*.

\DTLsetstringdatum { *cs* } { *string* }

datatool-base v3.0+

§2.2.3; 19

Defines *cs* to a string datum.

\DTLsettabseparator

datatool

§3.15.2;
359

Sets the separator to the tab character in order to load TSV files and changes the category code for the tab character to 12 “other”.

\DTLsetup { *key=value list* }

datatool-base v3.0+

§2.1; 8

Set available options for all loaded packages in the datatool bundle.

\DTLsort [*replacements*] { *criteria* } { *db-name* }

modifier: * datatool

§3.14.2;
335

Uses `\dtlsort` to sort the given database. The starred version uses `\dtlcompare` as the handler function, and the unstarred version uses `\dtlcompare`.

\dtlsort [*replacements*] { *criteria* } { *db-name* } { *handler-cs* }

datatool v2.13+

§3.14.2;
334

Sorts the rows of the database identified by *db-name*.

\DTLsortdata [*options*] { *db-name* } { *criteria* }

datatool v3.0+

§3.14.1;
325

Sorts a database according to the given criteria, which should be a *key=value* list.

\DTLsortedactual { *sorted element* }

datatool-base v3.0+

§2.9.5;
159

Expands the actual (original) value from a sorted element obtained by `\DTLsortwordlist`.

\DTLsortedletter { *sorted element* }

datatool-base v3.0+

§2.9.5;
159

Expands the letter group from a sorted element obtained by `\DTLsortwordlist`.

\DTLsortedvalue { *sorted element* }

datatool-base v3.0+

§2.9.5;
159

Expands the sort value from a sorted element obtained by `\DTLsortwordlist`.

\DTLsortlettercasehandler { *original* } { *cs* }

datatool-base v3.0+

§2.9.5.2;
165

Case-sensitive letter handler for use in `\DTLsortwordlist`. Expands *original*, strips hyphens and spaces, then applies `\DTLDefaultLocaleWordHandler` and purifies the result.

\DTLsortletterhandler { *original* } { *cs* }

datatool-base v3.0+

§2.9.5.2;
165

Case-insensitive letter handler for use in `\DTLsortwordlist`. Expands and converts *original* to lowercase, strips hyphens and spaces, then applies `\DTLDefaultLocaleWordHandler` and purifies the result.

\dtlsortlist { *list-cs* } { *criteria cs* }

datatool-base v2.27+

§2.9.5;
156

Sorts the given CSV list according to the *criteria cs* command, which must take three arguments: { *reg* } { *A* } { *B* } where *reg* is a count register and *A* and *B* are two elements to compare.

\DTLsortwordcasehandler { *original* } { *cs* }

datatool-base v3.0+

§2.9.5.2;
165

Case-sensitive word handler for use in `\DTLsortwordlist`. Expands *original*, then applies `\DTLDefaultLocaleWordHandler` and purifies the result.

\dtlSortWordCommands { *code* }

datatool-base v3.0+

§2.9.5.3;
168

Globally adds *code* to the hook used by `\DTLsortwordlist`, `\dtlwordindexcompare` and `\dtlletterindexcompare`.

\DTLsortwordhandler { *⟨original⟩* } { *⟨cs⟩* }

datatool–base v3.0+

§2.9.5.2;
164

Case-insensitive word handler for use in `\DTLsortwordlist` and `\dtlwordindexcompare` and `\dtlletterindexcompare`. Expands and converts *⟨original⟩* to lowercase, then applies `\DTLDefaultLocaleWordHandler` and purifies the result.

\DTLsortwordlist { *⟨list-cs⟩* } { *⟨handler-cs⟩* }

datatool–base v3.0+

§2.9.5;
156

Sorts the CSV list stored in the command *⟨list-cs⟩*. The *⟨handler-cs⟩* argument is a handler macro used to convert the original value into a byte sequence. The handler macro should be defined with the syntax *⟨handler-cs⟩* { *⟨original⟩* } { *⟨tl⟩* } where *⟨original⟩* is the original value and *⟨tl⟩* is a token list control sequence in which to store the byte sequence.

\dtlspecialvalue { *⟨text⟩* }

datatool v3.0+

§3.11; 317

Ordinarily simply expands to *⟨text⟩*. However, if present at the start of a database element, `\DTLwrite` will allow it to expand for DBTEX formats during the write operation, even if `expand=none` is on.

\DTLsplitstring { *⟨string⟩* } { *⟨split text⟩* } { *⟨before cmd⟩* } { *⟨after cmd⟩* }

datatool–base v1.01+

§2.8.1;
139

Splits *⟨string⟩* at *⟨split text⟩* and defines *⟨before cmd⟩* to the pre-split text and *⟨after cmd⟩* to the post-split text (neither *⟨string⟩* nor *⟨split text⟩* are expanded).

\DTLsqrt { *⟨cs⟩* } { *⟨num⟩* }

datatool–base

§2.5.2;
111

Calculates the square root of the formatted number *⟨num⟩* and stores the result as a formatted number in the control sequence *⟨cs⟩*.

\dtlsqrt { *⟨cs⟩* } { *⟨num⟩* }

datatool–base

§2.5.1; 99

Calculates the square root of *⟨num⟩* and stores the result in the control sequence *⟨cs⟩*, where the number is a plain number.

\DTLstartpt

databar

§5.4.4;
448

For use in the definition of `\DTLeverybarhook`, this expands to the starting point of the current bar along its mid-axis as a pgf point.

\DTLStoreInitialGetLetter { *<word>* } { *<cs>* }

datatool-base v3.0+

§2.8.2;
141

Used by `\DTLstoreinitials` to get the initial letter of *<word>*. Default definition just uses `\DTLGetInitialLetter`.

\DTLstoreinitials { *<text>* } { *<cs>* }

datatool-base

§2.8.2;
140

Splits *<text>* into words and defines the control sequence *<cs>* to expand to the initials of each word.

\dtlstringalign

initial: 1 datatool v2.0+

§3.7.2;
254

Expands to the column alignment specifier used by `\DTLdisplaydb` and `\DTLdisplaylongdb` for columns with the string data type. Ignored if the `align-specs` option is set.

\dtlstringformat { *<text>* }

datatool v2.0+

§3.7.2;
252

Used by `\DTLdisplaydb` and `\DTLdisplaylongdb` to format entries in string columns.

\DTLstringnull

datatool-base

§3.10.2;
316

Used to represent a null value for a string column (see §3.10). Prior to version 3.0, this was defined by `datatool` rather than `datatool-base`.

\DTLstringtype

datatool v2.0+

§3.6; 239

Expands to 0.

\DTLsub {*<cs>*} {*<num1>*} {*<num2>*}

datatool-base

§2.5.2;
110

Calculates $\langle num1 \rangle - \langle num2 \rangle$ and stores the result in the control sequence *<cs>*, where the numbers are formatted numbers. The result will be a formatted number.

\dtlsub {*<cs>*} {*<num1>*} {*<num2>*}

datatool-base

§2.5.1; 99

Calculates $\langle num1 \rangle - \langle num2 \rangle$ and stores the result in the control sequence *<cs>*, where the numbers are plain numbers.

\DTLsubstitute {*<cs>*} {*<original>*} {*<replacement>*}

datatool-base

§2.8.1;
138

Substitutes the first occurrence of *<original>* with *<replacement>* within the expansion text of the command *<cs>*.

\DTLsubstituteall {*<cs>*} {*<original>*} {*<replacement>*}

datatool-base

§2.8.1;
139

Substitutes all occurrences of *<original>* with *<replacement>* within the expansion text of the command *<cs>*.

\DTLsumcolumn {*<db>*} {*<key>*} {*<cmd>*}

datatool

§3.13; 322

Sums all numeric values in the column with the label *<key>* of the database identified by *<db>* and stores the result as a formatted number in the control sequence *<cmd>*.

\DTLsumforkeys [*<condition>*] [*<assign-list>*] {*<db list>*} {*<key list>*} {*<cmd>*}

datatool

§3.13; 321

Sums over all numeric values in the listed columns in the given databases and stores the result as a formatted number in the control sequence *<cmd>*. The optional arguments may be used to skip rows where the condition evaluates to false.

\dtlswapentriesincurrentrow {*<col1 num>*} {*<col2 num>*} datatool v2.10+

§3.16.1;
374

Locally swaps the values in the columns identified by their index, *<col1 num>* and *<col2 num>*, in `\dtlcurrentrow`.

\dtlswaprows { $\langle db-name \rangle$ } { $\langle row1-idx \rangle$ } { $\langle row2-idx \rangle$ }

datatool

§3.16; 371

Swaps the rows identified by their index. No check is performed to determine if the database exists or if the indexes are in range.

\DTL $\langle tag \rangle$ LocaleHook

datatool- $\langle tag \rangle$.ldf

Hook provided by localisation files, in particular, the region hook control sequence name needs to be in this form for datatool-base to add it to the captions hook.

\DTLtemporalvalue { $\langle number \rangle$ } { $\langle ISO \rangle$ }

datatool-base v3.0+

§2.2.4; 21

Used within temporal datum values to store both the numeric value and ISO format within the value part.

\dtltexorsort { $\langle T_{EX} \rangle$ } { $\langle sort \rangle$ }

datatool-base v3.0+

§2.9.5.3;
166

By default this expands to $\langle T_{EX} \rangle$, ignoring the second argument, but is redefined to expand to the second argument, ignoring the first, by commands like `\DTLsortwordlist`.

\DTLtherow

datatool v3.0+

§3.8.2;
295

Uses `\theDTLrow $\langle n \rangle$` where $\langle n \rangle$ is one more than `\dtlforeachlevel`.

\DTLticklabeloffset

initial: 8pt dataplot

§6.4.1;
521

Dimension that specifies the offset from the axis to the tick label.

\DTLticklength

initial: 5pt dataplot

§6.4.1;
521

Dimension that specifies length of the tick marks.

\dtltimegroup { $\langle value \rangle$ }

datatool-base v3.0+

Used by `\DTLassignlettergroup` to identify time groups.

\DTLtotalbargroups

databar v3.0+

§5.4.4;
450

For use in the bar chart hooks with `\DTLmultibarchart`, this will expand to the total number of bar groups in the current bar chart.

\DTLtotalbars

databar v3.0+

§5.4.4;
449

For use in the bar chart hooks, this will expand to the total number of bars in the current bar chart.

\DTLtrunc { *cs* } { *num* } { *num digits* }

datatool-base

§2.5.2;
112

Truncates the formatted number *num* to *num digits* and stores the result as a formatted number in the control sequence *cs*.

\dtltrunc { *cs* } { *num* } { *dp* }

datatool-base

§2.5.1;
100

Truncates *num* to *dp* decimal places and stores the result in the control sequence *cs*, where the number is a plain number.

\DTLtwoand

databib

§7.7.1;
546

Used between the names where there are only two names in the list.

\dtltype

datatool

§3.8.2;
297

Defined by `\DTLforeachkeyinrow` to the current column type.

\DTLunsettype

datatool v2.0+

§3.6; 239

Expands to empty.

\dtlupdateentryincurrentrow{*<col key>*}{*<value>*}

datatool v2.11+

§3.16.1;
374

Appends the entry, as per `\dtlappendentrytocurrentrow`, if there is no entry for the given column in the current row, otherwise updates the entry.

\DTLuse{*<property>*}

datatool v3.0+

§3.3; 206

Uses the returned value from the last `\DTLaction` in the current scope. Leave the argument empty for the main return value, otherwise *<property>* should identify the particular value when there are multiple return values.

\DTLusedatum{*<cs>*}

datatool-base v3.0+

§2.2.3; 17

Expands to the original value that was parsed by `\DTLparse` (or the expanded value for `\DTLxparse`).

\DTLvarianceforall{*<cs>*}{*<num list>*}

datatool-base

§2.5.2;
114

Sets *<cs>* to the variance of all the formatted numbers in the comma-separated *<num list>*. The result will be a formatted number.

\dtlvarianceforall[*<mean>*]{*<cs>*}{*<num list>*}

datatool-base v3.0+

§2.5.1;
101

Calculates the variance of all the numbers in the comma-separated list *<num list>* and stores the result in the control sequence *<cs>*, where the numbers are plain numbers. If the mean value has already been computed, it can be supplied in the optional argument *<mean>*. If omitted, the mean will be calculated before calculating the variance.

\DTLvarianceforcolumn{*<db>*}{*<key>*}{*<cmd>*}

datatool

§3.13; 322

Computes the variance over all numeric values in the column with the label *<key>* of the database identified by *<db>* and stores the result as a formatted number in the control sequence *<cmd>*.

\DTLvarianceforkeys [*<condition>*] [*<assign-list>*] {*<db list>*} {*<key list>*} {*<cmd>*}
datatool

§3.13; 322

Computes the variance over all numeric values in the listed columns in the given databases and stores the result as a formatted number in the control sequence *<cmd>*. The optional arguments may be used to skip rows where the condition evaluates to false.

\dtlwordindexcompare {*<count-reg>*} {*<string1>*} {*<string2>*}
datatool-base

§2.9.5.1;
163

Word order comparison that may be used in *\dtlsortlist*.

\DTLwrite [*<options>*] {*<filename>*}
datatool v3.0+

§3.15.4;
364

Saves the data in the database identified by the *name* option in the file called *<filename>*. If the file extension is omitted, the default for the given format is used.

\DTLXAxisStyle
initial: — dataplot

§6.4.3;
524

Expands to the *tikz* option to use for the *x*-axis line style.

\DTLxparse {*<cs>*} {*<content>*}
datatool-base v3.0+

§2.2.3; 16

As *\DTLparse* but first fully expands *<content>*.

\DTLxsetcurrencydatum {*<cs>*} {*<formatted value>*} {*<value>*} {*<currency symbol>*}
datatool-base v3.0+

§2.2.3; 19

Defines *<cs>* to a currency datum, expanding *<formatted value>*, *<value>* and *<currency symbol>*.

\DTLxsetdecimaldatum {*<cs>*} {*<formatted value>*} {*<value>*}
datatool-base v3.0+

§2.2.3; 18

Defines *<cs>* to a decimal datum, expanding *<formatted value>* and *<value>*.

\DTLxsetintegerdatum { *cs* } { *formatted value* } { *value* }
 datatool-base v3.0+

§2.2.3; 18

Defines *cs* to an integer datum, expanding *formatted value* and *value*.

\DTLxsetstringdatum { *cs* } { *string* }
 datatool-base v3.0+

§2.2.3; 19

Defines *cs* to a string datum, expanding *string*.

\DTLxsplitstring { *string* } { *split text* } { *before cmd* } { *after cmd* }
 datatool-base v3.0+

§2.8.1;
139

As `\DTLsplitstring` but expands *string* and *split text* once.

\DTLYAxisStyle
initial: – dataplot

§6.4.3;
524

Expands to the tikz option to use for the *y*-axis line style.

E

\editionname
initial: edition databib

Locale-sensitive command provided if it hasn't already been defined.

\editorname
initial: editor databib

Locale-sensitive command provided if it hasn't already been defined.

\editorsname
initial: editors databib

Locale-sensitive command provided if it hasn't already been defined.

\edtlgetrowforvalue { *db-name* } { *col idx* } { *value* }
 datatool v2.17+

As `\edtlgetrowforvalue` but first expands *value*.

\etalname

initial: et al. databib

Locale-sensitive command provided if it hasn't already been defined.

F

 **\femalelabels**

person pre v3.0

§9.4; 634

Prior to v3.0, `\femalelabels` expanded to the comma-separated list of recognised female gender labels. This command was replaced with `\g_person_female_label_clist` variable in version 3.0 to reduce the possibility of a command name clash. Now only available with rollback. Use `\PersonSetFemaleLabels` to reset the list or `\PersonAddFemaleLabel` to append to the list.

\field{*<key>*}

datagidx

§8.6; 604

May be used in the *<default value>* argument of `\newtermaddfield` to reference the value of another field.

\FirstId

datagidx v2.15+

§8.9.1;
620

Placeholder in `\printterms` that expands to the current entry's `FirstId` value.

\forallpeople [*<people-label-list>*] {*<label-cs>*} {*<body>*}

person

§9.7.2;
657

Iterates over all labels in the given list and, at each iteration, sets *<label-cs>* to the current label and does *<body>*. If the optional argument is omitted, the list of all defined people is used.

\foreachperson (*<name-cs>*, *<full-name-cs>*, *<gender-cs>*, *<label-cs>*) \in
{*<label-list>*} \do {*<body>*}

person

§9.7.2;
657

Iterates through the list of people and, at each iteration, assigns *<name-cs>* to the person's name, *<full-name-cs>* to the person's full name, *<gender-cs>* to the language-sensitive gender text, and *<label-cs>* to the person's label, and then does *<body>*. The `\in{<label-list>}` is optional. If omitted the list of all defined people is assumed.

\foreachpersonbreak

person v3.0+

§9.7.2;
656

Breaks out of the people loops.

G

\gDTLforeachbibentry [*⟨condition⟩*] { *⟨db-name⟩* } { *⟨body⟩* } *modifier:* *
databib

§7.8; 549

As `\DTLforeachbibentry` but performs global assignments.

\gDTLformatbibentry

databib

§7.8; 551

As `\DTLformatbibentry` but performs global changes.

\getpersonforenames { *⟨cs⟩* } { *⟨person-label⟩* }

person v3.0+

§9.5.1;
645

Defines *⟨cs⟩* to the person's forenames.

\getpersonfullname { *⟨cs⟩* } { *⟨person-label⟩* }

person

§9.5.1;
645

Defines *⟨cs⟩* to the person's full name.

\getpersongender { *⟨cs⟩* } { *⟨person-label⟩* }

person

§9.5.1;
644

Defines *⟨cs⟩* to the language-sensitive text identifying the person's gender.

\getpersongenderlabel { *⟨cs⟩* } { *⟨person-label⟩* }

person

§9.5.1;
644

Defines *⟨cs⟩* to the internal gender label associated with the given person.

\getpersonname { *⟨cs⟩* } { *⟨person-label⟩* }

person

§9.5.1;
644

Defines *⟨cs⟩* to the person's name.

\getpersonsurname { *cs* } { *person-label* }

person v3.0+

§9.5.1;
645

Defines *cs* to the person's surname.

\getpersontitle { *cs* } { *person-label* }

person

§9.5.1;
645

Defines *cs* to the person's title.

\Gls { *label* }

datagidx

§8.5.1;
602

Shortcut for `\Useentry{label}{Text}`.

\gls { *label* }

datagidx

§8.5.1;
602

Shortcut for `\useentry{label}{Text}`.

\glsadd { *label* }

datagidx

§8.5; 601

Indexes the term identified by *label* without producing any text.

\glsaddall { *db-name* }

datagidx

§8.5; 602

Indexes all the terms in the database identified by *db-name* without producing any text.

\Glsdispentry { *label* } { *col-key* }

datagidx

§8.5; 601

As `\Glsdispentry` but converts the value to sentence case.

\glsdispentry { *label* } { *col-key* }

datagidx

§8.5; 601

Displays the value of the given column in the row identified by *label* from the index/glossary database associated with *label* (without indexing or creating a hyperlink).

\glslink { *label* } { *text* }

datagidx

§8.5; 600

Like `\useentry` but displays the provided text instead of the value of a field.

\Glsnl { *label* }

datagidx

§8.5.1;
603

Shortcut for `\Useentrynl` { *label* } { *Text* }.

\glsnl { *label* }

datagidx

§8.5.1;
602

Shortcut for `\useentrynl` { *label* } { *Text* }.

\Glspl { *label* }

datagidx

§8.5.1;
603

Shortcut for `\Useentry` { *label* } { *Plural* }.

\glspl { *label* }

datagidx

§8.5.1;
602

Shortcut for `\useentry` { *label* } { *Plural* }.

\Glsplnl { *label* }

datagidx

§8.5.1;
603

Shortcut for `\Useentrynl` { *label* } { *Plural* }.

\glsplnl { *label* }

datagidx

§8.5.1;
602

Shortcut for `\useentrynl` { *label* } { *Plural* }.

\glsreset { *label* }

datagidx

§8.7.2;
609

Resets a term so that it's marked as not used.

\glsresetall{*<db-name>*}

datagidx

§8.7.2;
609

Resets all terms in the given database.

\Glsym{*<label>*}

datagidx

§8.5.1;
603

Shortcut for `\Useentry{<label>}{Symbol}`.

\glsym{*<label>*}

datagidx

§8.5.1;
603

Shortcut for `\useentry{<label>}{Symbol}`.

\glsunset{*<label>*}

datagidx

§8.7.2;
609

Unsets a term so that it's marked as used.

\glsunsetall{*<db-name>*}

datagidx

§8.7.2;
609

Unsets all terms in the given database.

\g_person_female_label_clist

person v3.0+

§9.4; 634

Comma separated list variable used to store the list of labels that may be used to identify the female gender.

\g_person_male_label_clist

person v3.0+

§9.4; 633

Comma separated list variable used to store the list of labels that may be used to identify the male gender.

H

\HierSort

datagidx v3.0+

§8.9.1;
620

Placeholder in `\printterms` that expands to the current entry's `HierSort` value.

|

\ifDTLbarxaxis `<true>\else <false>\fi` *initial:* `\iffalse` databar

If true, the x axis should be drawn on bar charts.

\ifDTLbaryaxis `<true>\else <false>\fi` *initial:* `\iffalse` databar

If true, the y axis should be drawn on bar charts.

\ifDTLbarytics `<true>\else <false>\fi` *initial:* `\iffalse` databar

If true, y ticks should be drawn on bar charts.

\ifDTLbox `<true>\else <false>\fi` *initial:* `\iffalse` dataplot

§6.4; 518

If true, the plot will be enclosed in a box.

\ifDTLgrid `<true>\else <false>\fi` *initial:* `\iffalse` dataplot

§6.4; 518

If true, a grid will be drawn. The minor grid lines will only be drawn if the corresponding minor tick mark setting is on.

 **\ifDTLnewdbonload** `<true>\else <false>\fi` *initial:* `\iftrue`
datatool-base v2.13+

§3.15.2;
357

If true, create a new database when loading a CSV/TSV file. Use `load-action` option instead.

\ifDTLshowlines $\langle true \rangle \backslash else \langle false \rangle \backslash fi$ *initial:* \iffalse dataplot

§6.4; 518

If true, lines should be drawn.

\ifDTLshowmarkers $\langle true \rangle \backslash else \langle false \rangle \backslash fi$ *initial:* \iftrue dataplot

§6.4; 519

If true, markers should be drawn.

\ifDTLverticalbars $\langle true \rangle \backslash else \langle false \rangle \backslash fi$ *initial:* \iftrue
databar

§5.4.1;
441

If true, the bar charts will be drawn with vertical bars, otherwise they will be drawn with horizontal bars.

\ifDTLxaxis $\langle true \rangle \backslash else \langle false \rangle \backslash fi$ *initial:* \iftrue dataplot

§6.4; 519

If true, x -axis should be drawn.

\ifDTLxminortics $\langle true \rangle \backslash else \langle false \rangle \backslash fi$ *initial:* \iffalse dataplot

§6.4; 519

If true, the x minor ticks will be drawn.

\ifDTLxtics $\langle true \rangle \backslash else \langle false \rangle \backslash fi$ *initial:* \iftrue dataplot

§6.4; 519

If true, the x ticks will be drawn.

\ifDTLxticsin $\langle true \rangle \backslash else \langle false \rangle \backslash fi$ *initial:* \iftrue dataplot

§6.4; 519

If true, the x ticks will be drawn inwards (if they should be drawn) otherwise they will be draw outwards.

\ifDTLyaxis $\langle true \rangle \backslash else \langle false \rangle \backslash fi$ *initial:* \iftrue dataplot

§6.4; 519

If true, y -axis should be drawn.

\ifDTLymnortics $\langle true \rangle \backslash else \langle false \rangle \backslash fi$ *initial:* `\iffalse` `dataplot`

§6.4; 519

If true, the y minor ticks will be drawn.

\ifDTLytics $\langle true \rangle \backslash else \langle false \rangle \backslash fi$ *initial:* `\iftrue` `dataplot`

§6.4; 520

If true, the y ticks will be drawn.

\ifDTLyticsin $\langle true \rangle \backslash else \langle false \rangle \backslash fi$ *initial:* `\iftrue` `dataplot`

§6.4; 520

If true, the y ticks will be drawn inwards (if they should be drawn) otherwise they will be draw outwards.

\ifentryused [$\langle label \rangle$] { $\langle true \rangle$ } { $\langle false \rangle$ } `datagidx`

§8.9.1;
619

Robust command that does $\langle true \rangle$ if the entry identified by $\langle label \rangle$ has been marked as used, otherwise does false.

 **\iffemalelabel** { $\langle gender-label \rangle$ } { $\langle true \rangle$ } { $\langle false \rangle$ } `person`

§9.4; 635

Deprecated. Use `\PersonIfFemaleLabel` instead.

 **\ifmalelabel** { $\langle gender-label \rangle$ } { $\langle true \rangle$ } { $\langle false \rangle$ } `person`

§9.4; 635

Deprecated. Use `\PersonIfMaleLabel` instead.

\ifpersonexists { $\langle person-label \rangle$ } { $\langle true \rangle$ } { $\langle false \rangle$ } `person`

§9.7.1;
654

Does $\langle true \rangle$ if there is a person defined with the given label, otherwise does $\langle false \rangle$. The $\langle person-label \rangle$ argument is trimmed and expanded before testing.

\iftermexists [$\langle label \rangle$] { $\langle true \rangle$ } { $\langle false \rangle$ } `datagidx`

§8.9.1;
618

Expandable command that does $\langle true \rangle$ if an entry has been defined with the given label, otherwise does $\langle false \rangle$.

L

\Label

datagidx v2.15+

§8.9.1;
620

Placeholder in \printterms that expands to the current entry's Label value.

\l_dataplot_legend_names_prop

dataplot v3.0+

§6.3.3;
516

Property list used by \DTLplotlegendname and modified by \DTLplotlegendsetname.

\l_dataplot_legend_name_tl

dataplot v3.0+

§6.3.3;
516

Token list variable specifically for accessing values in the \l_dataplot_legend_names_prop property list.

\l_dataplot_legend_tl

dataplot v3.0+

§6.3.2;
513

Token list variable used to construct the legend.

\l_dataplot_legend_xlabels_prop

dataplot v3.0+

§6.3.3;
517

Property list used by \DTLplotlegendx and modified by \DTLplotlegendsetxlabel.

\l_dataplot_legend_xlabel_tl

dataplot v3.0+

§6.3.3;
517

Token list variable specifically for accessing values in the \l_dataplot_legend_xlabels_prop property list.

\l_dataplot_legend_ylabels_prop

dataplot v3.0+

§6.3.3;
518

Property list used by \DTLplotlegandy and modified by \DTLplotlegendsetylabel.

\l_dataplot_legend_ylabel_tl

dataplot v3.0+

§6.3.3;
518

Token variable list specifically for accessing values in the `\l_dataplot_legend_ylabels_prop` property list.

\l_dataplot_stream_index_int

dataplot v3.0+

§6.3.2;
513

Integer variable that keeps track of the stream index (starting from 1).

\l_dataplot_x_key_int

dataplot v3.0+

§6.3.2;
514

Integer variable used by `\DTLplot` to keep track of the index of the x list.

\l_dataplot_y_key_int

dataplot v3.0+

§6.3.2;
514

Integer variable used by `\DTLplot` to keep track of the index of the y list.

\l_datatool_austral_str

datatool-base v3.0+

§2.3.1; 38

Expands to the string representation of the austral sign “ ”, if supported by the current encoding, or “A” otherwise.

\l_datatool_austral_tl

datatool-base v3.0+

§2.3.1; 38

Expands to the symbol representation of the austral sign “ ”, if supported by the current encoding, or “A” otherwise.

\l_datatool_baht_str

datatool-base v3.0+

§2.3.1; 33

Expands to the string representation of the baht sign “ ”, if supported by the current encoding, or “B” otherwise.

\l_datatool_baht_tl

datatool-base v3.0+

§2.3.1; 33

Expands to the symbol representation of the baht sign “ ”, if supported by the current encoding, or “B” otherwise.

`\l_datatool_bitcoin_str`

datatool-base v3.0+

§2.3.1; 41

Expands to the string representation of the bitcoin sign “ ₿ ”, if supported by the current encoding, or “ L ” otherwise.

`\l_datatool_bitcoin_tl`

datatool-base v3.0+

§2.3.1; 41

Expands to the symbol representation of the bitcoin sign “ ₿ ”, if supported by the current encoding, or “ L ” otherwise.

`\l_datatool_caption_tl`

initial: `\c_novalue_tl` datatool v3.0+

§3.7.2;
259

A token list variable assigned by the `caption` key.

`\l_datatool_cedi_str`

datatool-base v3.0+

§2.3.1; 39

Expands to the string representation of the cedi sign “ ₵ ”, if supported by the current encoding, or “ S ” otherwise.

`\l_datatool_cedi_tl`

datatool-base v3.0+

§2.3.1; 39

Expands to the symbol representation of the cedi sign “ ₵ ”, if supported by the current encoding, or “ S ” otherwise.

`\l_datatool_cent_str`

datatool-base v3.0+

§2.3.1; 32

Expands to the string representation of the cent sign “ ¢ ”, if supported by the current encoding, or “ C ” otherwise.

`\l_datatool_cent_tl`

datatool-base v3.0+

§2.3.1; 32

Expands to the symbol representation of the cent sign “ ¢ ”, if supported by the current encoding, or “ C ” otherwise.

\l_datatool_colonsign_str

datatool-base v3.0+

§2.3.1; 34

Expands to the string representation of the colon sign “ ”, if supported by the current encoding, or “C” otherwise.

\l_datatool_colonsign_tl

datatool-base v3.0+

§2.3.1; 34

Expands to the symbol representation of the colon sign “ ”, if supported by the current encoding, or “C” otherwise.

\l_datatool_cont_caption_tl

initial: \c_novalue_tl

datatool v3.0+

§3.7.2;
259

A token list variable assigned by the `cont-caption` key.

\l_datatool_cruzerio_str

datatool-base v3.0+

§2.3.1; 34

Expands to the string representation of the cruzerio sign “ ”, if supported by the current encoding, or “Cr” otherwise.

\l_datatool_cruzerio_tl

datatool-base v3.0+

§2.3.1; 34

Expands to the symbol representation of the cruzerio sign “ ”, if supported by the current encoding, or “Cr” otherwise.

\l_datatool_currencysigns_regex

datatool-base v3.0+

§2.3.1; 42

A regular expression that matches supported currency symbols.

\l_datatool_currency_str

datatool-base v3.0+

§2.3.1; 32

Expands to the string representation of the currency sign “ ”, if supported by the current encoding, or “#” otherwise.

\l_datatool_currency_tl

datatool-base v3.0+

§2.3.1; 32

Expands to the symbol representation of the currency sign “ ”, if supported by the current encoding, or “ # ” otherwise.

\l_datatool_current_language_tl *initial: empty* datatool-base v3.0+

§2.3.5; 60

Each language file should ensure that the captions hook sets this token list variable to expand to the language’s ISO code.

\l_datatool_current_region_tl *initial: empty* datatool-base v3.0+

Each region file should ensure that the captions hook sets this token list variable to expand to the region’s ISO code.

\l_datatool_display_per_row_int

datatool v3.0+

§3.7.2;
259

Integer variable that corresponds to the `per-row` option.

\l_datatool_display_tab_rows_int

datatool v3.0+

§3.7.2;
259

Integer variable set to the database row count divided by the value of `per-row` rounded up.

\l_datatool_dong_str

datatool-base v3.0+

§2.3.1; 36

Expands to the string representation of the dong sign “ ”, if supported by the current encoding, or “ d ” otherwise.

\l_datatool_dong_tl

datatool-base v3.0+

§2.3.1; 36

Expands to the symbol representation of the dong sign “ ”, if supported by the current encoding, or “ d ” otherwise.

\l_datatool_drachma_str

datatool-base v3.0+

§2.3.1; 37

Expands to the string representation of the drachma sign “ ”, if supported by the current encoding, or “Dr” otherwise.

\l_datatool_drachma_tl

datatool-base v3.0+

§2.3.1; 37

Expands to the symbol representation of the drachma sign “ ”, if supported by the current encoding, or “Dr” otherwise.

\l_datatool_ecu_str

datatool-base v3.0+

§2.3.1; 33

Expands to the string representation of the ecu sign “ ”, if supported by the current encoding, or “CE” otherwise.

\l_datatool_ecu_tl

datatool-base v3.0+

§2.3.1; 33

Expands to the symbol representation of the ecu sign “ ”, if supported by the current encoding, or “CE” otherwise.

\l_datatool_euro_str

datatool-base v3.0+

§2.3.1; 36

Expands to the string representation of the euro sign “ ”, if supported by the current encoding, or “E” otherwise.

\l_datatool_euro_tl

datatool-base v3.0+

§2.3.1; 36

Expands to the symbol representation of the euro sign “ ”, if supported by the current encoding, or “E” otherwise.

\l_datatool_florin_str

datatool-base v3.0+

§2.3.1; 33

Expands to the string representation of the florin sign “ ”, if supported by the current encoding, or “f” otherwise.

\l_datatool_florin_tl

datatool-base v3.0+

§2.3.1; 33

Expands to the symbol representation of the florin sign “ ₣ ”, if supported by the current encoding, or “ f ” otherwise.

\l_datatool_foot_tl

initial: \c_novalue_tl datatool v3.0+

§3.7.2;
259

A token list variable assigned by the `foot` key.

\l_datatool_frenchfranc_str

datatool-base v3.0+

§2.3.1; 34

Expands to the string representation of the French franc sign “ ₣ ”, if supported by the current encoding, or “ F ” otherwise.

\l_datatool_frenchfranc_tl

datatool-base v3.0+

§2.3.1; 34

Expands to the symbol representation of the French franc sign “ ₣ ”, if supported by the current encoding, or “ F ” otherwise.

\l_datatool_germanpenny_str

datatool-base v3.0+

§2.3.1; 37

Expands to the string representation of the Germany penny sign “ ¢ ”, if supported by the current encoding, or “ p ” otherwise.

\l_datatool_germanpenny_tl

datatool-base v3.0+

§2.3.1; 37

Expands to the symbol representation of the Germany penny sign “ ¢ ”, if supported by the current encoding, or “ p ” otherwise.

\l_datatool_guarani_str

datatool-base v3.0+

§2.3.1; 38

Expands to the string representation of the guarani sign “ ₧ ”, if supported by the current encoding, or “ G . ” otherwise.

`\l_datatool_guarani_tl`

datatool-base v3.0+

§2.3.1; 38

Expands to the symbol representation of the guarani sign “ ”, if supported by the current encoding, or “ G . ” otherwise.

`\l_datatool_hryvnia_str`

datatool-base v3.0+

§2.3.1; 38

Expands to the string representation of the hryvnia sign “ ”, if supported by the current encoding, or “ S ” otherwise.

`\l_datatool_hryvnia_tl`

datatool-base v3.0+

§2.3.1; 38

Expands to the symbol representation of the hryvnia sign “ ”, if supported by the current encoding, or “ S ” otherwise.

`\l_datatool_include_header_bool`

initial: `true` datatool v3.0+

§3.7.2;
259

A boolean variable corresponding to the inverse of the `no-header` key.

`\l_datatool_indianrupee_str`

datatool-base v3.0+

§2.3.1; 40

Expands to the string representation of the Indian rupee sign “ ”, if supported by the current encoding, or “ R ” otherwise.

`\l_datatool_indianrupee_tl`

datatool-base v3.0+

§2.3.1; 40

Expands to the symbol representation of the Indian rupee sign “ ”, if supported by the current encoding, or “ R ” otherwise.

`\l_datatool_kip_str`

datatool-base v3.0+

§2.3.1; 37

Expands to the string representation of the kip sign “ ”, if supported by the current encoding, or “ K ” otherwise.

`\l_datatool_kip_tl`

datatool-base v3.0+

§2.3.1; 37

Expands to the symbol representation of the kip sign “ ”, if supported by the current encoding, or “K” otherwise.

`\l_datatool_label_tl`

initial: `\c_novalue_tl` datatool v3.0+

§3.7.2;
259

A token list variable assigned by the `label` key.

`\l_datatool_lari_str`

datatool-base v3.0+

§2.3.1; 41

Expands to the string representation of the lari sign “ ”, if supported by the current encoding, or “L” otherwise.

`\l_datatool_lari_tl`

datatool-base v3.0+

§2.3.1; 41

Expands to the symbol representation of the lari sign “ ”, if supported by the current encoding, or “L” otherwise.

`\l_datatool_last_foot_tl`

initial: `\c_novalue_tl` datatool v3.0+

§3.7.2;
259

A token list variable assigned by the `last-foot` key.

`\l_datatool_lira_str`

datatool-base v3.0+

§2.3.1; 34

Expands to the string representation of the lira sign “ ”, if supported by the current encoding, or “L” otherwise.

`\l_datatool_lira_tl`

datatool-base v3.0+

§2.3.1; 34

Expands to the symbol representation of the lira sign “ ”, if supported by the current encoding, or “L” otherwise.

\l_datatool_livretournois_str

datatool-base v3.0+

§2.3.1; 39

Expands to the string representation of the livre tournois sign “ ”, if supported by the current encoding, or “`lt`” otherwise.

\l_datatool_livretournois_tl

datatool-base v3.0+

§2.3.1; 39

Expands to the symbol representation of the livre tournois sign “ ”, if supported by the current encoding, or “`lt`” otherwise.

\l_datatool_manat_str

datatool-base v3.0+

§2.3.1; 40

Expands to the string representation of the manat sign “ ”, if supported by the current encoding, or “`M`” otherwise.

\l_datatool_manat_tl

datatool-base v3.0+

§2.3.1; 40

Expands to the symbol representation of the manat sign “ ”, if supported by the current encoding, or “`M`” otherwise.

\l_datatool_measure_hook_tl

datatool-base v3.0+

§2.8.3;
147

Token list variable used by the measuring commands `\datatool_measure_height:Nn`, `\datatool_measure_width:Nn` and `\datatool_measure_depth:Nn`.

\l_datatool_middot_str

datatool-base v3.0+

§2.3.1; 33

Expands to the string representation of the middle dot (raised decimal point) “ ”, if supported by the current encoding, or “`.`” otherwise.

\l_datatool_middot_tl

datatool-base v3.0+

§2.3.1; 33

Expands to the symbol representation of the middle dot (raised decimal point) “ ”, if supported by the current encoding, or “`.`” otherwise.

\l_datatool_mill_str

datatool-base v3.0+

§2.3.1; 35

Expands to the string representation of the mill sign “ ¢ ”, if supported by the current encoding, or “m” otherwise.

\l_datatool_mill_tl

datatool-base v3.0+

§2.3.1; 35

Expands to the symbol representation of the mill sign “ ¢ ”, if supported by the current encoding, or “m” otherwise.

\l_datatool_naira_str

datatool-base v3.0+

§2.3.1; 35

Expands to the string representation of the naira sign “ ₦ ”, if supported by the current encoding, or “N” otherwise.

\l_datatool_naira_tl

datatool-base v3.0+

§2.3.1; 35

Expands to the symbol representation of the naira sign “ ₦ ”, if supported by the current encoding, or “N” otherwise.

\l_datatool_nordicmark_str

datatool-base v3.0+

§2.3.1; 40

Expands to the string representation of the Nordic mark sign “ ¤ ”, if supported by the current encoding, or “M” otherwise.

\l_datatool_nordicmark_tl

datatool-base v3.0+

§2.3.1; 40

Expands to the symbol representation of the Nordic mark sign “ ¤ ”, if supported by the current encoding, or “M” otherwise.

\l_datatool_peseta_str

datatool-base v3.0+

§2.3.1; 35

Expands to the string representation of the peseta sign “ ¢ ”, if supported by the current encoding, or “Pts” otherwise.

`\l_datatool_peseta_tl`

datatool-base v3.0+

§2.3.1; 35

Expands to the symbol representation of the peseta sign “*₧*”, if supported by the current encoding, or “*Pts*” otherwise.

`\l_datatool_peso_str`

datatool-base v3.0+

§2.3.1; 38

Expands to the string representation of the peso sign “*\$*”, if supported by the current encoding, or “*P*” otherwise.

`\l_datatool_peso_tl`

datatool-base v3.0+

§2.3.1; 38

Expands to the symbol representation of the peso sign “*\$*”, if supported by the current encoding, or “*P*” otherwise.

`\l_datatool_post_head_tl` *initial:* `\c_novalue_tl` datatool v3.0+

§3.7.2;
259

A token list variable assigned by the `post-head` key.

`\l_datatool_pound_str`

datatool-base v3.0+

§2.3.1; 32

Expands to the string representation of the pound sign “*£*”, if supported by the current encoding, or “*L*” otherwise.

`\l_datatool_pound_tl`

datatool-base v3.0+

§2.3.1; 32

Expands to the symbol representation of the pound sign “*£*”, if supported by the current encoding, or “*L*” otherwise.

`\l_datatool_region_set_currency_bool`

datatool-base v3.0+

§2.3.2; 45

Boolean variable that corresponds to the `region-currency` option, which should be checked by region files.

\l_datatool_region_set_numberchars_bool

datatool-base v3.0+

§2.3.2; 45

Boolean variable that corresponds to the `region-number-chars` option, which should be checked by region files.

\l_datatool_ruble_str

datatool-base v3.0+

§2.3.1; 41

Expands to the string representation of the ruble sign “`₽`”, if supported by the current encoding, or “`R`” otherwise.

\l_datatool_ruble_tl

datatool-base v3.0+

§2.3.1; 41

Expands to the symbol representation of the ruble sign “`₽`”, if supported by the current encoding, or “`R`” otherwise.

\l_datatool_rupee_str

datatool-base v3.0+

§2.3.1; 35

Expands to the string representation of the rupee sign “`₹`”, if supported by the current encoding, or “`Rs`” otherwise.

\l_datatool_rupee_tl

datatool-base v3.0+

§2.3.1; 35

Expands to the symbol representation of the rupee sign “`₹`”, if supported by the current encoding, or “`Rs`” otherwise.

\l_datatool_shekel_str

datatool-base v3.0+

§2.3.1; 36

Expands to the string representation of the shekel sign “`₪`”, if supported by the current encoding, or “`S`” otherwise.

\l_datatool_shekel_tl

datatool-base v3.0+

§2.3.1; 36

Expands to the symbol representation of the shekel sign “`₪`”, if supported by the current encoding, or “`S`” otherwise.

\l_datatool_short_caption_tl

datatool v3.0+

initial: \c_novalue_tl

§3.7.2;
259

A token list variable assigned by the `short-caption` key.

\l_datatool_som_str

datatool-base v3.0+

§2.3.1; 41

Expands to the string representation of the som sign “ ”, if supported by the current encoding, or “c” otherwise.

\l_datatool_som_tl

datatool-base v3.0+

§2.3.1; 41

Expands to the symbol representation of the som sign “ ”, if supported by the current encoding, or “c” otherwise.

\l_datatool_spesmilo_str

datatool-base v3.0+

§2.3.1; 39

Expands to the string representation of the spesmilo sign “ ”, if supported by the current encoding, or “Sm” otherwise.

\l_datatool_spesmilo_tl

datatool-base v3.0+

§2.3.1; 39

Expands to the symbol representation of the spesmilo sign “ ”, if supported by the current encoding, or “Sm” otherwise.

\l_datatool_str_csv_regex_cases_tl

datatool v3.0+

§3.15.2;
350

A token list variable containing regular expression cases used by the `csv-content=literal` setting.

\l_datatool_tenge_str

datatool-base v3.0+

§2.3.1; 39

Expands to the string representation of the tenge sign “ ”, if supported by the current encoding, or “T” otherwise.

\l_datatool_tenge_tl

datatool-base v3.0+

§2.3.1; 39

Expands to the symbol representation of the tenge sign “ ₸ ”, if supported by the current encoding, or “ T ” otherwise.

\l_datatool_tugrik_str

datatool-base v3.0+

§2.3.1; 37

Expands to the string representation of the tugrik sign “ ₮ ”, if supported by the current encoding, or “ T ” otherwise.

\l_datatool_tugrik_tl

datatool-base v3.0+

§2.3.1; 37

Expands to the symbol representation of the tugrik sign “ ₮ ”, if supported by the current encoding, or “ T ” otherwise.

\l_datatool_turkishlira_str

datatool-base v3.0+

§2.3.1; 40

Expands to the string representation of the Turkish lira sign “ ₺ ”, if supported by the current encoding, or “ L ” otherwise.

\l_datatool_turkishlira_tl

datatool-base v3.0+

§2.3.1; 40

Expands to the symbol representation of the Turkish lira sign “ ₺ ”, if supported by the current encoding, or “ L ” otherwise.

\l_datatool_won_str

datatool-base v3.0+

§2.3.1; 36

Expands to the string representation of the won sign “ ₩ ”, if supported by the current encoding, or “ W ” otherwise.

\l_datatool_won_tl

datatool-base v3.0+

§2.3.1; 36

Expands to the symbol representation of the won sign “ ₩ ”, if supported by the current encoding, or “ W ” otherwise.

\l_datatool_yen_str

datatool-base v3.0+

§2.3.1; 32

Expands to the string representation of the yen sign “ ”, if supported by the current encoding, or “Y” otherwise.

\l_datatool_yen_tl

datatool-base v3.0+

§2.3.1; 32

Expands to the symbol representation of the yen sign “ ”, if supported by the current encoding, or “Y” otherwise.

\loadgidx [*<options>*] { *<filename>* } { *<title>* }

datagidx v2.15+

§8.3; 588

Loads a previously saved datagidx database.

\Location

datagidx v2.15+

§8.9.1;
620

Placeholder in `\printterms` that expands to the current entry’s `Location` value.

\Long

datagidx v2.15+

§8.9.1;
619

Placeholder in `\printterms` that expands to the current entry’s `Long` value.

\LongPlural

datagidx v2.15+

§8.9.1;
619

Placeholder in `\printterms` that expands to the current entry’s `LongPlural` value.

\l_person_label_tl

person v3.0+

§9.7.4;
660

Token list variable that expands to the current person label, which may be used in `\newperson` hooks.

M

❌ **\malelabels**

person pre v3.0

§9.4; 633

Prior to v3.0, `\malelabels` expanded to the comma-separated list of recognised male gender labels. This command was replaced with `\g_person_male_label_clist` variable in version 3.0 to reduce the possibility of a command name clash. Now only available with rollback. Use `\PersonSetMaleLabels` to reset the list or `\PersonAddMaleLabel` to append to the list.

\mscthesisname

initial: Master's thesis databib

Locale-sensitive command provided if it hasn't already been defined.

N

\Name

datagidx v2.15+

§8.9.1;
619

Placeholder in `\printterms` that expands to the current entry's Name value.

\newacro [*⟨options⟩*] {*⟨short⟩*} {*⟨long⟩*}

datagidx

§8.7; 606

Shortcut that defines an abbreviation with `\newterm`.

\newgidx [*⟨options⟩*] {*⟨db-name⟩*} {*⟨title⟩*}
(preamble only)

datagidx

§8.2; 587

Defines a new index/glossary database.

\newperson [*⟨person-label⟩*] {*⟨full name⟩*} {*⟨name⟩*} {*⟨gender-label⟩*}

person

§9.3; 630

Defines a person identified by the given label. If omitted, *⟨person-label⟩* will default to `anon`. The *⟨gender-label⟩* may be “unknown” or empty, if not known, or a valid gender label. The starred form `\newperson*` uses a *⟨key⟩=⟨value⟩* interface.

\newperson* [*⟨person-label⟩*] {*⟨key=value list⟩*}

person v3.0+

§9.3; 630

Defines a person identified by the given label. If omitted, *⟨person-label⟩* will default to `anon`.

\newterm [*⟨options⟩*] {*⟨name⟩*}
(preamble only)

datagidx

§8.4; 588

Defines a term with the given name.

\newtermaddfield [*⟨db list⟩*] {*⟨column key⟩*} [*⟨placeholder cs⟩*] {*⟨new term key⟩*} [*⟨data type⟩*] {*⟨default value⟩*}

datagidx

§8.6; 604

Defines a new column and associated option.

\newtermlabelhook

datagidx

§8.4.2;
597

Used by `\newterm` when creating a label.

\newtermsorthook

datagidx

§8.4.3;
599

Used by `\newterm` when creating a sort value.

\numbername

initial: number databib

Locale-sensitive command provided if it hasn't already been defined.

P

\pagename

initial: page databib

§7.7.1;
546

Locale-sensitive command provided if it hasn't already been defined. This command is defined by language packages, such as babel.

\pagesname

initial: pages databib

§7.7.1;
546

Locale-sensitive command provided if it hasn't already been defined.

\Parent

datagidx v2.15+

§8.9.1;
620

Placeholder in `\printterms` that expands to the current entry's `Parent` value.

\Parents

(requires `shortcuts` option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for `\Peopleparent`.

\parents

(requires `shortcuts` option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for `\peopleparent`.

\Peoplechild

person

§9.5.0.2;
642

As `\peoplechild` but starts with a capital.

\peoplechild

person

§9.5.0.2;
642

Displays the relationship of the defined people to their collective parents.

\peopleforenames

person v3.0+

§9.5.2;
646

Displays all the defined people, showing the forenames for each person.

\peoplefullname

person

§9.5.2;
646

Displays all the defined people, showing the full name for each person.

\peoplename

person

§9.5.2;
646

Displays all the defined people, showing the name for each person.

\Peopleobjpronoun

person

§9.5.0.1;
639

As \peopleobjpronoun, but sentence-case.

\peopleobjpronoun

person

§9.5.0.1;
639

Displays the third person plural objective pronoun according to the gender of all defined people.

\Peopleobjpronounii

person

§9.5.0.1;
639

As \peopleobjpronounii, but sentence-case.

\peopleobjpronounii

person

§9.5.0.1;
639

Displays the second person plural objective pronoun according to the gender of all defined people.

\Peopleparent

person

§9.5.0.2;
643

As \peopleparent but starts with a capital.

\peopleparent

person

§9.5.0.2;
643

Displays the relationship of the defined people to their collective children.

\Peoplepossadj

person

§9.5.0.1;
640

As \peoplepossadj, but sentence-case.

\peoplepossadj

person

§9.5.0.1;
640

Displays the third person plural possessive adjective according to the gender of all defined people.

\Peoplepossadjii

person

§9.5.0.1;
641

As \peoplepossadjii, but sentence-case.

\peoplepossadjii

person

§9.5.0.1;
640

Displays the second person plural possessive adjective according to the gender of all defined people.

\Peopleposspronoun

person

§9.5.0.1;
641

As \peopleposspronoun, but sentence-case.

\peopleposspronoun

person

§9.5.0.1;
641

Displays the third person plural possessive pronoun according to the gender of all defined people.

\Peopleposspronounii

person

§9.5.0.1;
642

As \peopleposspronounii, but sentence-case.

\peopleposspronounii

person

§9.5.0.1;
642

Displays the second person plural possessive pronoun according to the gender of all defined people.

\Peoplepronoun

person

§9.5.0.1;
638

As \peoplepronoun, but sentence-case.

\peoplepronoun

person

§9.5.0.1;
638

Displays the third person plural subjective pronoun according to the gender of all defined people.

\Peoplepronounii

person

§9.5.0.1;
638

As `\peoplepronounii`, but sentence-case.

\peoplepronounii

person

§9.5.0.1;
638

Displays the second person plural subjective pronoun according to the gender of all defined people.

\Peoplesibling

person

§9.5.0.2;
643

As `\peoplesibling` but starts with a capital.

\peoplesibling

person

§9.5.0.2;
643

Displays the relationship of the defined people to their collective siblings.

\peoplesurname

person v3.0+

§9.5.2;
646

Displays all the defined people, showing the surname for each person.

\peopletitlesurname

person

§9.5.2;
646

Displays all the defined people, showing the title and surname for each person.

\PersonAddFemaleLabel { *<label>* }

person v3.0+

Adds the given label (after expansion) to the list of labels that may be used to identify the female gender.

\PersonAddMaleLabel { *<label>* }

person v3.0+

Adds the given label (after expansion) to the list of labels that may be used to identify the male gender.

\PersonAddNonBinaryLabel { *<label>* }

person v3.0+

Adds the given label (after expansion) to the list of labels that may be used to identify the non-binary gender.

\person_all_gender_case:nnnn { *<all-male-case>* }
 { *<all-female-case>* } { *<all-non-binary-case>* } { *<other-case>* }

person v3.0+

§9.7.1;
656

Does *<all-male-case>* if all defined people have been identified as male, does *<all-female-case>* if all defined people have been identified as female, does *<all-non-binary-case>* if all defined people have been identified as non-binary, and does *<other-case>* otherwise (that is, a mixture of genders or all defined people have no gender specified).

\Personchild [*<person-label>*]

person

§9.5.0.2;
642

As `\personchild` but starts with a capital.

\personchild [*<person-label>*]

person

§9.5.0.2;
642

Displays the person's relationship to their parents.

\person_do_if_valid_gender:nt { *<token-list>* } { *<code>* } person v3.0+

§9.7.1;
656

Does *<code>* if the given *<token-list>* matches a recognised internal gender label, otherwise triggers an error and does nothing.

\PersonFemaleCount

person v3.0+

§9.4; 635

Expands to the number of female people who have been defined.

\personforenames [*<person-label>*]

person v3.0+

§9.5.1;
644

Displays the person's forenames. If the label is omitted, `anon` is assumed.

\personfullname [*<person-label>*]

person

§9.5.1;
643

Displays the person's full name. If the label is omitted, anon is assumed.

\Persongender { *<person-label>* }

person

As \persongender but starts with a capital.

\persongender { *<person-label>* }

person

§9.5.1;
644

Displays the person's gender using localisation.

\person_gender_case:Nnnnn *<tl-var>* { *<male-case>* } { *<female-case>* }
{ *<non-binary-case>* } { *<unknown-case>* } person v3.0+

§9.7.1;
656

As \person_gender_case:Nnnnnn but issues an error for the invalid case and treats it as unknown.

\person_gender_case:nnnnn *<token-list>* { *<male-case>* }
{ *<female-case>* } { *<non-binary-case>* } { *<unknown-case>* } person v3.0+

§9.7.1;
656

As \person_gender_case:Nnnnnn but the gender label is supplied as a token list.

\person_gender_case:Nnnnnn *<tl-var>* { *<invalid-case>* }
{ *<male-case>* } { *<female-case>* } { *<non-binary-case>* } { *<unknown-case>* }
person v3.0+

§9.7.1;
655

Compares the given token list variable (using \tl_if_eq:NNTF) against the constants \c_person_male_label_tl, \c_person_female_label_tl, \c_person_nonbinary_label_tl, and \c_person_unknown_label_tl and does the applicable case or *<invalid-case>* if no match.

```
\person_gender_case:nnnnnn <token-list> { <invalid-case> }
{ <male-case> } { <female-case> } { <non-binary-case> } { <unknown-case> }
person v3.0+
```

§9.7.1;
656

As \person_gender_case:Nnnnnn but the gender label is supplied as a token list.

```
\person_get_attribute:nn { <person-label> } { <attribute> } person v3.0+
```

§9.7; 653

Expands to the value of the attribute for the given person. No test is made to determine if the person exists or if the attribute is valid.

```
\person_get_attribute:Nnn <tl var> { <person-label> } { <attribute> }
person v3.0+
```

§9.7; 654

Sets the token list variable *<tl var>* to the value of the attribute for the given person. No test is made to determine if the person exists or if the attribute is valid.

```
\PersonIfAllFemale [ <person-label-list> ] { <true> } { <false> } person v3.0+
```

§9.7.1;
654

Does *<true>* if all the people identified in the list *<person-label-list>* were defined with the gender set to female, otherwise does *<false>*. If the optional argument is omitted, this simply compares if \PersonTotalCount is equal to \PersonFemaleCount.

```
\PersonIfAllMale [ <person-label-list> ] { <true> } { <false> } person v3.0+
```

§9.7.1;
654

Does *<true>* if all the people identified in the list *<person-label-list>* were defined with the gender set to male, otherwise does *<false>*. If the optional argument is omitted, this simply compares if \PersonTotalCount is equal to \PersonMaleCount.

```
\PersonIfAllNonBinary [ <person-label-list> ] { <true> } { <false> } person v3.0+
```

§9.7.1;
655

Does *<true>* if all the people identified in the list *<person-label-list>* were defined with the gender set to non-binary, otherwise does *<false>*. If the optional argument is omitted, this simply compares if \PersonTotalCount is equal to \PersonNonBinaryCount.

\PersonIfAllUnknownGender [*<person-label-list>*] { *<true>* } { *<false>* }
 person v3.0+

§9.7.1;
655

Does *<true>* if all the people identified in the list *<person-label-list>* were defined with the gender set to unknown, otherwise does *<false>*. If the optional argument is omitted, this simply compares if `\PersonTotalCount` is equal to `\PersonUnknownGenderCount`.

\person_if_exist:nTF { *<person-label>* } { *<true>* } { *<false>* }
\person_if_exist_p:n { *<person-label>* }
 person v3.0+

§9.7.1;
655

Tests if a person has been defined with the given label.

\PersonIfFemale { *<person-label>* } { *<true>* } { *<false>* }
 person v3.0+

§9.7.1;
654

Does *<true>* if the person identified by *<person-label>* was defined with the gender set to female, otherwise does *<false>*.

\PersonIfFemaleLabel { *<gender-label>* } { *<true>* } { *<false>* }
 person v3.0+

§9.4; 635

Does *<true>* if the one-level expansion of *<gender-label>* is recognised as a female gender identifier.

\PersonIfMale { *<person-label>* } { *<true>* } { *<false>* }
 person v3.0+

§9.7.1;
654

Does *<true>* if the person identified by *<person-label>* was defined with the gender set to male, otherwise does *<false>*.

\PersonIfMaleLabel { *<gender-label>* } { *<true>* } { *<false>* }
 person v3.0+

§9.4; 635

Does *<true>* if the one-level expansion of *<gender-label>* is recognised as a male gender identifier.

\PersonIfNonBinary { *<person-label>* } { *<true>* } { *<false>* }
 person v3.0+

§9.7.1;
655

Does *<true>* if the person identified by *<person-label>* was defined with the gender set to non-binary, otherwise does *<false>*.

\PersonIfNonBinaryLabel { *<gender-label>* } { *<true>* } { *<false>* } person v3.0+

§9.4; 635

Does *<true>* if the one-level expansion of *<gender-label>* is recognised as a non-binary gender identifier (does not include “unknown”).

\PersonIfUnknownGender { *<person-label>* } { *<true>* } { *<false>* } person v3.0+

§9.7.1;
655

Does *<true>* if the person identified by *<person-label>* was defined with the gender set to unknown, otherwise does *<false>*.

\person_language_all_text:n { *<type>* } person v3.0+

§9.7.3;
660

As `\person_language_all_text:n` but converts to sentence case.

\person_language_all_text:n { *<type>* } person v3.0+

§9.7.3;
660

Uses the plural form of the localisation text (`plural<type>`) if more than one person is defined otherwise uses the singular form *<type>*.

\person_language_text:nn { *<person-label>* } { *<type>* } person v3.0+

§9.7.3;
659

As `\person_language_text:nn` but converts the text to sentence case.

\person_language_text:nn { *<person-label>* } { *<type>* } person v3.0+

§9.7.3;
659

Expands to the localisation text for the given *<type>* for the gender label associated with the given person.

\personlastsep person

§9.5.2;
645

Separator to use between the last pair of people in a list that contains more that two people.

\PersonMaleCount person v3.0+

§9.4; 635

Expands to the number of male people who have been defined.

\personname [*⟨person-label⟩*]

person

§9.5.1;
644

Displays the person's name. If the label is omitted, anon is assumed.

\person_new_appto_end:n {*⟨code⟩*}

person v3.0+

§9.7.4;
661

Append *⟨code⟩* to hook used at the end of \newperson.

\person_new_appto_start:n {*⟨code⟩*}

person v3.0+

§9.7.4;
660

Append *⟨code⟩* to hook used at the start of \newperson.

\PersonNonBinaryCount

person v3.0+

§9.4; 635

Expands to the number of non-binary people who have been defined.

\Personobjpronoun [*⟨person-label⟩*]

person

§9.5.0.1;
639

As \personobjpronoun, but sentence-case.

\personobjpronoun [*⟨person-label⟩*]

person

§9.5.0.1;
638

Displays the third person singular objective pronoun according to the gender of the person identified by the label.

\Personobjpronounii [*⟨person-label⟩*]

person

§9.5.0.1;
639

As \personobjpronounii, but sentence-case.

\personobjpronounii [*⟨person-label⟩*]

person

§9.5.0.1;
639

Displays the second person singular objective pronoun according to the gender of the person identified by the label.

\Personparent [*<person-label>*]

person

§9.5.0.2;
642

As `\personparent` but starts with a capital.

\personparent [*<person-label>*]

person

§9.5.0.2;
642

Displays the person's relationship to their child.

\Personpossadj [*<person-label>*]

person

§9.5.0.1;
640

As `\personpossadj`, but sentence-case.

\personpossadj [*<person-label>*]

person

§9.5.0.1;
640

Displays the third person singular possessive adjective according to the gender of the person identified by the label.

\Personpossadjii [*<person-label>*]

person

§9.5.0.1;
640

As `\personpossadjii`, but sentence-case.

\personpossadjii [*<person-label>*]

person

§9.5.0.1;
640

Displays the second person singular possessive adjective according to the gender of the person identified by the label.

\Personposspronoun [*<person-label>*]

person

§9.5.0.1;
641

As `\personposspronoun`, but sentence-case.

\personposspronoun [*<person-label>*]

person

§9.5.0.1;
641

Displays the third person singular possessive pronoun according to the gender of the person identified by the label.

\Personposspronounii [*<person-label>*]

person

§9.5.0.1;
641

As `\personposspronounii`, but sentence-case.

\personposspronounii [*<person-label>*]

person

§9.5.0.1;
641

Displays the second person singular possessive pronoun according to the gender of the person identified by the label.

\Personpronoun [*<person-label>*]

person

§9.5.0.1;
636

As `\personpronoun`, but sentence-case.

\personpronoun [*<person-label>*]

person

§9.5.0.1;
636

Displays the third person singular subjective pronoun according to the gender of the person identified by the label.

\Personpronounii [*<person-label>*]

person

§9.5.0.1;
638

As `\personpronounii`, but sentence-case.

\personpronounii [*<person-label>*]

person

§9.5.0.1;
638

Displays the second person singular subjective pronoun according to the gender of the person identified by the label.

\person_remove_appto:n {*<code>*}

person v3.0+

§9.7.4;
661

Append *<code>* to hook used by `\removeperson` and `\removepeople`.

\personsep

initial: `\DTLlistformatsep` person

§9.5.2;
645

Separator to use between all but the last pair of people in a list.

\person_set_attribute:nnn { *<person-label>* } { *<attribute>* } { *<value>* }
variant: nnV person v3.0+

§9.7; 653

Sets the attribute for the given person to *<value>* (according to the `local` setting). No test is made to determine if the person exists or if the attribute is valid.

\PersonSetFemaleLabels { *<label list>* } person v3.0+

§9.4; 634

Sets the list of labels that may be used to identify the female gender.

\PersonSetLocalisation { *<gender-label>* } { *<type>* } { *<value>* } person v3.0+

§9.7.3;
657

Sets the localisation text for the given gender label (which must be one of the internal labels `male`, `female`, `nonbinary` or `unknown`) for the given *<type>*.

\PersonSetMaleLabels { *<label list>* } person v3.0+

§9.4; 633

Sets the list of labels that may be used to identify the male gender.

\PersonSetNonBinaryLabels { *<label list>* } person v3.0+

§9.4; 634

Sets the list of labels that may be used to identify the non-binary gender.

\Personsibling [*<person-label>*] person

§9.5.0.2;
643

As `\personsibling` but starts with a capital.

\personsibling [*<person-label>*] person

§9.5.0.2;
643

Displays the person's relationship to their siblings.

\personsurname [*<person-label>*] person v3.0+

§9.5.1;
644

Displays the person's surname. If the label is omitted, `anon` is assumed.

\persontitlesurname [*⟨person-label⟩*]

person

§9.5.1;
644

Displays the person's title and surname. If the label is omitted, `anon` is assumed.

\persontitlesurnamesep

initial: _ person

§9.5.1;
644

The separator between a person's title and surname.

\PersonTotalCount

person v3.0+

§9.3; 632

Expands to the number of people who have been defined.

\PersonUnknownGenderCount

person v3.0+

§9.4; 636

Expands to the number of people who have been defined with the gender set to unknown.

\person_unset_attribute:nn {*⟨person-label⟩*} {*⟨attribute⟩*}

person v3.0+

§9.7; 654

Undefines the attribute for the given person to *⟨value⟩* (according to the `local` setting). No test is made to determine if the person exists or if the attribute name is valid.

\Plural

datagidx v2.15+

§8.9.1;
620

Placeholder in `\printterms` that expands to the current entry's `Plural` value.

\postnewtermhook

datagidx v2.14+

§8.9.2;
620

A hook used at the end of `\newterm`.

\printterms [*⟨options⟩*]

datagidx

§8.8; 609

Displays index/glossary according to the given options.

\printtermsrestoreonecolumn

datagidx

§8.8; 611

Used to restore one column if `\twocolumn` was issued by `\printterms` and two-column mode wasn't already in effect.

R

\removeallpeople

person

§9.3; 633

Removes all defined people.

\removepeople { *⟨list⟩* }

person

§9.3; 633

Removes each person identified by their label in the given comma-separated list.

\removeperson [*⟨person-label⟩*]

person

§9.3; 632

Removes (undefines) the person identified by *⟨person-label⟩*. If omitted, this defaults to `anon`.

S

\See

datagidx v2.15+

§8.9.1;
620

Placeholder in `\printterms` that expands to the current entry's `See` value.

\SeeAlso

datagidx v2.15+

§8.9.1;
620

Placeholder in `\printterms` that expands to the current entry's `SeeAlso` value.

\seealso

datagidx

§8.8.1;
614

If not already defined, this will be defined to `\also` if that is defined or to “see also” otherwise.

\Short

datagidx v2.15+

§8.9.1;
619

Placeholder in \printterms that expands to the current entry's Short value.

\ShortPlural

datagidx v2.15+

§8.9.1;
620

Placeholder in \printterms that expands to the current entry's ShortPlural value.

\Siblings

(requires shortcuts option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for \Peoplesibling.

\siblings

(requires shortcuts option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for \peoplesibling.

\Sort

datagidx v2.15+

§8.9.1;
620

Placeholder in \printterms that expands to the current entry's Sort value.

\Symbol

datagidx v2.15+

§8.9.1;
619

Placeholder in \printterms that expands to the current entry's Symbol value.

T

\techreportname

initial: Technical report databib

Locale-sensitive command provided if it hasn't already been defined.

\Text

datagidx v2.15+

§8.9.1;
620

Placeholder in \printterms that expands to the current entry's Text value.

\Thee

(requires shortcuts option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for \Peopleobjpronounii.

\thee

(requires shortcuts option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for \peopleobjpronounii.

\Their

(requires shortcuts option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for \Peoplepossadj.

\their

(requires shortcuts option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for \peoplepossadj.

\Theirs

(requires shortcuts option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for \Peopleposspronoun.

\theirs

(requires shortcuts option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for \peopleposspronoun.

\Them

(requires shortcuts option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for \Peopleobjpronoun.

\them

(requires `shortcuts` option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for `\peopleobjpronoun`.

\They

(requires `shortcuts` option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for `\Peoplepronoun`.

\they

(requires `shortcuts` option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for `\peoplepronoun`.

\twopeoplesep

initial: `\DTLlistformatlastsep` person

§9.5.2;
645

Separator to use in a list of two people.

U

\Used

datagidx v2.15+

§8.9.1;
619

Placeholder in `\printterms` that expands to the current entry's `Used` value.

\USEentry { *⟨label⟩* } { *⟨col-key⟩* }

datagidx

§8.5; 600

As `\useentry` but the text is converted to uppercase.

\Useentry { *⟨label⟩* } { *⟨col-key⟩* }

datagidx

§8.5; 600

As `\useentry` but the text is converted to sentence case.

\useentry { *<label>* } { *<col-key>* }

datagidx

§8.5; 599

Displays the value of the given column in the row identified by *<label>* from the index/glossary database associated with *<label>*. This command also indexes and, if enabled, will provide a hyperlink to the relevant line in `\printterms`.

\USEentrynl { *<label>* } { *<col-key>* }

datagidx

§8.5; 600

As `\USEentry` but doesn't create a hyperlink.

\Useentrynl { *<label>* } { *<col-key>* }

datagidx

§8.5; 600

As `\Useentry` but doesn't create a hyperlink.

\useentrynl { *<label>* } { *<col-key>* }

datagidx

§8.5; 600

As `\useentry` but doesn't create a hyperlink.

V

\volumename

initial: volume databib

Locale-sensitive command provided if it hasn't already been defined.

X

\xdtlgetrowindex { *<row-cs>* } { *<db-name>* } { *<col idx>* } { *<value>* } datatool v2.28+

§3.16; 370

As `\dtlgetrowindex` but expands the value.

\xDTLinitials { *<text>* }

datatool-base v3.0+

§2.8.2;
140

Expands the first token in *<text>* before passing to `\DTLinitials`.

Y

\You

(requires `shortcuts` option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for `\Peoplepronounii`.

\you

(requires `shortcuts` option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for `\peoplepronounii`.

\Your

(requires `shortcuts` option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for `\Peoplepossadjii`.

\your

(requires `shortcuts` option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for `\peoplepossadjii`.

\Yours

(requires `shortcuts` option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for `\Peopleposspronounii`.

\yours

(requires `shortcuts` option)

person v3.0+

§9.5.0.1;
Table 9.1

Shortcut for `\peopleposspronounii`.

Environment Summary

```
\begin{DTLenvforeach} [⟨condition⟩] {⟨db-name⟩} {⟨assign-list⟩}     datatool
```

§3.8.2;
293,
294, 823

Environment alternative to `\DTLforeach`.

```
\begin{DTLenvforeach*} [⟨condition⟩] {⟨db-name⟩} {⟨assign-list⟩}     datatool
```

§3.8.2;
294, 823

Environment alternative to `\DTLforeach*`.

```
\begin{dtlenvgforint} {⟨count-reg⟩=⟨start⟩\to⟨end⟩\step⟨inc⟩}  
datatool-base
```

§3.16.2;
376, 823

Environment form of `\dtlgforint` where leading and trailing spaces are trimmed from the environment body.

```
\begin{dtlenvgforint*} {⟨count-reg⟩=⟨start⟩\to⟨end⟩\step⟨inc⟩}  
datatool-base
```

§3.16.2;
376, 823

Environment form of `\dtlgforint` where leading and trailing spaces aren't trimmed from the environment body.

```
\begin{DTLenvmapdata} [⟨key=value list⟩]     datatool v3.0+
```

§3.8.1;
286, 288,
289,
746, 823

Does `\DTLmapdata` [⟨key=value list⟩] {⟨body⟩} but leading and trailing spaces are trimmed from the body.

```
\begin{DTLthebibliography} [⟨condition⟩] {⟨db-name⟩}     databib
```

696, 823

Formats the bibliography (using `thebibliography`) according to the current style.

Package Option Summary

`\usepackage[<options>]{databar}`

§5;
399, 824

color

≡ databar

Initialises the default colour list to: red, green, blue, yellow, magenta, cyan, orange, white.

§5.1;
406, 824

gray

≡ databar

Initialises the default colour list to shades of grey.

§5.1;
406, 824

horizontal

≡ databar

Equivalent to `verticalbars=false`.

§5.1;
406, 824

vertical

≡ databar

Equivalent to `verticalbars=true`.

§5.1;
406, 824

verticalbars=*<boolean>*

default: true; initial: true ○ databar

If true, the bars will be vertical, otherwise they will be horizontal.

§5.1;
406, 824

`\usepackage[<options>]{datbib}`

§7;
526, 824

auto=*<value>*

default: true; initial: false ≡ datbib v3.0+

If true, bibtex will be run via the shell escape.

§7.1; 527,
572,
574, 824

style=*<name>*

≡ datbib

Sets the bibliography style to *<name>*, which may be one of: plain, abbrv or alpha.

§7.1; 527,
537, 542,
559, 824

`\usepackage[<options>]{datagidx}`

§8;
577, 824

child=*<value>*

initial: named ≡ datagidx

Sets child style, where the value may be named or noname.

§8.1.1.2;
579, 824

columns=*<n>*

≡ datagidx

§8.1.1.2;
579, 824

Sets the number of columns.

compositor= $\{ \langle character \rangle \}$

Sets the location compositor.

initial: .  datagidx

§8.1.1.1;
579, 825

counter= $\{ \langle counter name \rangle \}$

Sets the location counter.

initial: page  datagidx

§8.1.1.1;
579, 825

draft

Switches on draft mode.

 datagidx

§8.1.1.1;
579, 825

final

Switches off draft mode (default).

 datagidx

§8.1.1.1;
579, 825

location= $\langle value \rangle$

Sets the location list style, where the value may be one of: hide, list or first.

 datagidx

§8.1.1.2;
580, 825

namecase= $\langle value \rangle$

Sets name case style, where the value may be nochange, uc, lc, firstuc or capitalise.

 datagidx

§8.1.1.2;
580, 825

nowarn= $\langle boolean \rangle$

If true, switch off datagidx warnings.

default: true; *initial:* false  datagidx

§8.1.1.1;
579,
581, 825

optimize= $\langle value \rangle$

Sets the optimization mode.

default: high; *initial:* off  datagidx

§8.1.1.1;
578, 587,
617,
618, 825

optimize=**high**

Highest level of optimization.

618, 825

optimize=**low**

Some optimization.

618, 825

optimize=**off**

No optimization.

617, 825

postdesc= $\langle value \rangle$

Sets the post-description style, where the value may be one of: none or dot.

 datagidx

§8.1.1.2;
580, 825

postname= $\{ \langle value \rangle \}$

Sets the code to insert after the name.

 datagidx

§8.1.1.2;
580, 825

prelocation= $\langle value \rangle$

Sets the style of the pre-location content, where the value may be none, enspace, space, dotfill or hfill.

 datagidx

§8.1.1.2;
580, 825

see= $\langle value \rangle$

 datagidx

§8.1.1.2;
580, 825

Sets the “see” style, where the value may be one of: comma, brackets, dot, space, nosep, semicolon, or location.

symboldesc= $\{\langle value \rangle\}$  datagidx
Sets the symbol-description style, where the value may be one of: symbol, desc, (symbol) desc, desc (symbol), symbol desc, or desc symbol.

§8.1.1.2;
580, 826

`\usepackage [options] {datapie}`

§4;
377, 826

color  datapie
Initialises the default colour list to: red, green, blue, yellow, magenta, cyan, orange, white.

§4.1; 380,
396, 826

gray  datapie
Initialises the default colour list to shades of grey.

§4.1; 380,
396, 826

norotateinner  datapie
Don’t rotate inner labels.

§4.1;
380, 826

norotateouter  datapie
Don’t rotate outer labels.

§4.1;
380, 826

rotateinner  datapie
Rotate inner labels.

§4.1;
380, 826

rotateouter  datapie
Rotate outer labels.

§4.1;
380, 826

`\usepackage [options] {datatool-base}`

§2; 8, 826

lang-warn= $\{\langle boolean \rangle\}$ *default: true; initial: true*  datatool-base v3.0+
If false, suppresses localisation warnings (and also switches off tracklang warnings). If true, enables localisation warnings without affecting tracklang’s warning setting.

§2.1;
9, 826

lang= $\{\langle locale list \rangle\}$ *alias: locales*  datatool-base v3.0+
Synonym of **locales**.

see
locales

locales= $\{\langle locale list \rangle\}$  datatool-base v3.0+
Adds each locale to the list of tracked languages (using `\TrackLanguageTag`) and will load the corresponding localisation files if they are installed.

§2.1; 9,
10, 15, 27,
28, 118,
174, 826

math= $\langle processor \rangle$ *initial: varies*  datatool-base v2.10+

§2.1; 3, 8,
9, 21, 62,
83, 84, 97,
164,
321, 826

Determines which processor should be used for mathematical functions. The default is `l3fp` unless `\directlua` is available, in which case the default is `lua`.

`math=fp` datatool-base v2.10+

Use `fp` commands for floating point arithmetic.

`math=l3fp` datatool-base v3.0+

Use `LATEX3` commands for floating point arithmetic.

`math=lua` datatool-base v3.0+

Use `\directlua` for floating point arithmetic.

`math=pgfmath` datatool-base v2.10+

Use `pgfmath` commands for floating point arithmetic.

nolocale

≡ datatool-base v3.0+

Prevent localisation support from being loaded, even if the required localisation files are installed.

```
\usepackage[options]{datatool}
```

delimiter=*char*

The delimiter character in CSV files.

initial: " ≡

separator=*char*

The separator character in CSV files.

initial: , ≡

```
\usepackage[options]{person}
```

base-only

Only load `datatool-base`, not `datatool`.

≡ person

datatool

Load `datatool`.

≡ person

shortcuts

Define shortcut commands.

≡ person v3.0+

9, 62,
83–85, 98,
104, 827
3, 8, 9, 62,
82, 83, 98,
102,
104, 827
3, 8, 9, 62,
83, 85, 98,
104, 827

9, 62, 83,
84, 86, 98,
107, 827

§2.1; 9,
10, 827

§3;
176, 827

§3.1; 177,
352, 827

§3.1;
181, 827

§9;
628, 827

§9.1; 628,
629,
648, 827

§9.1;
629, 827

§9.1; 629,
630, 647,
649,
651, 827

Index

Symbols	
\\	255, 258, 275, 293, 298, 300, 301, 305, 306
_	Table 3.1; 43, 168, 683
—	Table 3.1
_ (backslash space)	168, 647, 651
\ (escape)	349, 350
\ (literal)	Table 3.1
\,	43, 683
, (number group)	11, 42, <i>see</i> number group character
, (separator)	337
? (unknown citation)	528
?? (unknown reference)	528
. (decimal point)	<i>see</i> decimal point (.)
^	Table 3.1
~ (literal)	Table 3.1
~ (non-breakable space)	70, 140, 594
~ (space token)	4, 6, 309
" (delimiter)	338
\{	Table 3.1; 5, 348
{	Table 3.1
\}	Table 3.1; 5, 348
}	Table 3.1
\\$	Table 3.1; 66, 67, 94, 123, 124, 126, 129, 160, 163, 168, 187, 198, 209, 228, 345, 348, 443
\$	Table 3.1; 124, 160, 187, 198
\&	Table 3.1; 59, 151, 152, 168, 597, 598, 621, 688, 744
&	Table 3.1; 269, 293, 305, 306
\#	Table 3.1; 168
#	Table 3.1
\%	Table 3.1; 168, 348
%	Table 3.1
0x<XX>	47, 48, 51, 164, 166–169, 171, 665, 673, 680, 684
@ (category code)	168, 341, 343, 355, 367
\@for	159, 628, 656
A	
\Acr	§8.7.1 ; 608, 667, 724
\acr	§8.7.1 ; 607, 608, 623, 667, 724
\acronymfont	§8.7 ; 606, 667
\Acrpl	§8.7.1 ; 608, 667, 724
\acrpl	§8.7.1 ; 607, 608, 623, 667, 724
actions	205, 224, <i>see</i> \DTLaction
add column	§3.3.1.1 ; 210
aggregate	§3.3.1.3 ; 206, 217, 220, 226–229, 321, 324
bar chart	§5 ; 224, 226, 401, 420
clear	§3.3.1.1 ; 207, 236
column data	§3.3.1.2 ; 214, 238
column index	§3.3.1.2 ; 206, 213, 214, 238
current row aggregate	§3.3.1.3 ; 220, 224, 244, 251, 281, 296, 302, 318, 371, 372
current row values	§3.3.1.2 ; 214, 216, 217, 264, 265, 269, 279, 280, 318, 372
delete	§3.3.1.1 ; 207, 235
display	§3.3.1.4 ; 205, 222, 223, 226, 242, 269, 276, 280, 284, 312, 314, 367, 369
display long	§3.3.1.4 ; 223, 242
find	§3.3.1.2 ; 212, 215, 300, 318, 319, 371
direction	§3.3.1.2 ; 212, 213
function	§3.3.1.2 ; 212, 213, 215

- inline §3.3.1.2; 213, 215
 select §3.3.1.2; 212, 215, 318, 319
 multibar chart §5; 224, 401, 403, 420
 new §3.3.1.1; 207, 232
 new entry .. §3.3.1.1; 178, 180, 208, 209, 210, 226, 232, 233, 317
 new row §3.3.1.1; 207, 208, 226, 232
 pie chart §4; 224, 226, 378
 plot §6; 224, 452, 453, 454, 473, 511–513
 row aggregate §3.3.1.3; 212, 214, 220, 291, 292, 302, 308, 309
 select row §3.3.1.2; 212, 214, 215, 220, 222, 228, 318, 319, 365, 366, 373, 601
 sort §3.3.1.5; 223, 273
 \addfemalelabel  ... §9.4; 634, 667
 \addmalelabel  §9.4; 633, 668
 afterpage package 577
 \alpha 597, 599, 621
 \alsoname 614, 817
 \and 89
 \andname 152, 688
 \appto 155, 258, 269, 280, 305, 306
 array environment 245
 ASCII . 31, 47, 51, 55, 56, 61, 148, 166, 170, 174, 336, 665, 673
 ⟨assign-list⟩ 665
- B**
- babel package 27, 60, 614, 802
 \babelprovide 27
 beamer class 608
 bib2gls 577, 588, 603
 \bibdata 527, 536
 biber 526, 529, 563
 \bibitem 546, 552, 571
 biblatex package 526, 529, 563
 \bibliographystyle . 543, 559, 745
 \bibstyle 527, 536
 bibtex 526, 527, 574, 745, 824
 booktabs package 248, 269, 271, 301
- C**
- \capitalisewords 606
 \caption 249
 \captions⟨dialect⟩ 60
 \c_datatool_apostrophe_ regex §2.3.2; 43, 668
 \c_datatool_empty_datum_tl 317, 668
 \c_datatool_string_int 668
 \children . §9.5.0.1; Table 9.1; 642, 668
 \Children (datagidx) .. §8.9.1; 583, 611, 620, 624, 668
 \Children (person) . §9.5.0.1; Table 9.1; 642, 668
 \citation 527, 553
 \cite .. 527, 529, 532, 536, 550, 553, 560, 567, 697
 \c_novalue_tl 13
 \color 382, 396, 446, 447, 460, 461, 521, 522, 712
 colortbl package 276, 281, 298, 308
 comma
 number group *see* number group character
 Oxford *see* Oxford comma
 separator *see* ,
 \c_person_female_label_tl 655, 669, 808
 \c_person_male_label_tl .. 655, 669, 808
 \c_person_nonbinary_label_tl 655, 669, 808
 \c_person_unknown_label_tl 655, 669, 808
 CSV 665
 currency symbol 15, 17, 42, 62, 64, 66–68, 97, 666, 706
 current row 212, 214–217, 220, 251, 263–265, 279, 296, 318, 371–374, 688, 689, 693, 703, 761, 773
 \CurrentLocation §8.9.1; 620, 624, 669

D

- databar package §5; 399, 824
 - color §5.1; 406, 824
 - gray §5.1; 406, 824
 - horizontal §5.1; 406, 824
 - vertical §5.1; 406, 824
 - verticalbars §5.1; 406, 824
- databib.bst ... §7; 526, 531, 532, 536, 537, 543, 559, 745
- databib-english.ldf ... §7.11; 30, 54, 574
- databib package §7; 526, 824
 - auto §7.1; 527, 572, 574, 824
 - style ... §7.1; 527, 537, 542, 559, 824
- databib column keys
 - Abstract §7.2.2; 532
 - Address §7.2.2; 532
 - Author ... §7.2.2; 532, 538–540, 545, 560, 562
 - BookTitle §7.2.2; 532
 - Chapter §7.2.2; 532
 - Citations §7.2.2; 532
 - CiteKey §7.2.2; 527, 532
 - Date §7.2.2; 533, 538, 546, 551, 694, 716
 - DOI §7.2.2; 533, 538, 547, 548, 694, 696
 - Edition §7.2.2; 533
 - Editor ... §7.2.2; 533, 538–540, 545
 - EID §7.2.2; 533
 - EntryType ... §7.2.2; 527, 532, 571
 - Eprints §7.2.2; 533, 547, 548, 694, 696
 - EprintType .. §7.2.2; 533, 547, 548
 - File §7.2.2; 533
 - HowPublished §7.2.2; 533
 - Institution §7.2.2; 534
 - ISBN §7.2.2; 534, 564
 - ISSN §7.2.2; 534
 - Journal §7.2.2; 534
 - Key §7.2.2; 534
 - Month §7.2.2; 534, 538, 546, 716
 - Note §7.2.2; 534
 - Number §7.2.2; 534
 - Organization §7.2.2; 534
 - Pages §7.2.2; 534, 546
 - Publisher §7.2.2; 535
 - PubMed §7.2.2; 535, 547, 696
 - School §7.2.2; 535
 - Series §7.2.2; 535
 - Title §7.2.2; 535, 540, 560
 - Type §7.2.2; 535
 - Url ... §7.2.2; 535, 538, 547, 549, 564, 696, 697
 - UrlDate .. §7.2.2; 535, 547, 549, 551, 694, 697
 - Volume §7.2.2; 535
 - Year §7.2.2; 535, 538, 541, 542, 546, 716
- datagidx package §8; 577, 824
 - child §8.1.1.2; 579, 824
 - columns §8.1.1.2; 579, 824
 - compositor §8.1.1.1; 579, 825
 - counter §8.1.1.1; 579, 825
 - draft §8.1.1.1; 579, 825
 - final §8.1.1.1; 579, 825
 - location §8.1.1.2; 580, 825
 - namecase §8.1.1.2; 580, 825
 - nowarn §8.1.1.1; 579, 581, 825
 - optimize ... §8.1.1.1; 578, 587, 617, 618, 825
 - high 618, 825
 - low 618, 825
 - off 617, 825
 - postdesc §8.1.1.2; 580, 825
 - postname §8.1.1.2; 580, 825
 - prelocation ... §8.1.1.2; 580, 825
 - see §8.1.1.2; 580, 825
 - symboldesc §8.1.1.2; 580, 826
- \datagidxconvertchars ... §8.9.2; 597, 598, 621, 669
- \datagidxcurrentgroup ... §8.9.1; 610, 619, 620, 624, 669
- \datagidxdictindent ... §8.8.2.5; 616, 670
- \datagidxend 611, 670
- \datagidxitem 610, 670

- `\datagidxlastlabel` §8.9.2; 620, 670
- `\datagidxlink` .. §8.9.3; 618, 622, 670
- `\datagidxlocalign` §8.8.1; 614, 670
- `\datagidxlocationwidth` . §8.8.1; 584, 614, 670
- `\datagidxmapdata` .. §8.9.1; 620, 670
- `\datagidxnewstyle` §8.9.3; 621, 671
- `\datagidxprevgroup` ... §8.9.1; 610, 619, 671
- `\datagidxsetstyle` §8.9.3; 610, 621, 671
- `\datagidxstart` 611, 671
- `\datagidxsymalign` §8.8.1; 614, 671
- `\datagidxsymbolwidth` §8.8.1; 587, 614, 671
- `\datagidxtarget` ... §8.9.3; 611, 618, 622, 671
- `\datagidxwordifygreek` ... §8.9.2; 597, 599, 621, 671
- `datapie` package §4; 377, 826
 - `color` §4.1; 380, 396, 826
 - `gray` §4.1; 380, 396, 826
 - `norotateinner` §4.1; 380, 826
 - `norotateouter` §4.1; 380, 826
 - `rotateinner` §4.1; 380, 826
 - `rotateouter` §4.1; 380, 826
- `dataplot` package §6; 451
- `\dataplot_apply_`
 - `pgftransform:` §6.3; 511, 524, 672
- `\dataplot_db_count:` §6.3.1; 512, 672
- `\dataplot_get_default_`
 - `legend:Nnnnn` §6.3.2; 513, 515, 516, 672
- `\dataplot_legend_add_begin:` §6.3.3; 514, 672
- `\dataplot_legend_add_end:` §6.3.3; 515, 672
- `\dataplot_x_key_count:` . §6.3.1; 512, 672
- `\dataplot_y_key_count:` . §6.3.1; 512, 672
- `datatool-base` package §2; 8, 826
 - `lang-warn` §2.1; 9, 826
 - `lang` *see* `locales`
 - `locales` . §2.1; 9, 10, 15, 27, 28, 118, 174, 826
 - `math` .. §2.1; 3, 8, 9, 21, 62, 83, 84, 97, 164, 321, 826
 - `fp` 9, 62, 83–85, 98, 104, 827
 - `l3fp` 3, 8, 9, 62, 82, 83, 98, 102, 104, 827
 - `lua` .. 3, 8, 9, 62, 83, 85, 98, 104, 827
 - `pgfmath` 9, 62, 83, 84, 86, 98, 107, 827
 - `nolocale` §2.1; 9, 10, 827
- `datatool-english` .. §2.3.5; 28, 29, 53, 54, 55, 174, 273, 331, 362, 559, 574, 624, 628, 657, 713
- `datatool-⟨locale⟩.ldf` 12, 27–31, 46–49, 51, 53–58, 60, 61, 124–126, 129, 130, 143, 162, 174, 675, 681, 688, 771
- `datatool-regions` 28, 29, 53, 54, 117, 124, 130, 362, 675
- `datatool` package §3; 176, 827
 - `delimiter` §3.1; 177, 352, 827
 - `separator` §3.1; 181, 827
- `\datatool_adjust_sign_fmt:n` §2.6; 128, 672, 681, 684
- `\datatoolasciend` §2.9.5.3; 166, 673
- `\datatoolasciistart` §2.9.5.3; 166, 673
- `\datatoolctrlboundary` . §2.9.5.3; 165, 166, 167, 168, 673
- `\datatoolcurrencysymbolprefixfmt` .. §2.3.2; 15, 45, 673, 681
- `\datatool_currency_symbol_region_prefix:n` §2.3.2; 44, 129, 673
- `\DataToolDateFmt` ... §2.7; 134, 137, 138, 673, 674
- `\DataToolDateTimeFmt` .. §2.7; 137, 673, 684, 685, *see also* `\DataToolTimeStampFmtSep`

`\datatool_datum_`
 `currency:Nnnnn` §2.2.4.1; 23, 674
`\datatool_datum_fp:nnn` . §2.2.4; 19, 21, 674, 765
`\datatool_datum_show:N` . §2.2.3; 18, 674
`\datatool_datum_`
 `string:Nnnnn` §2.2.4.1; 22, 674
`\datatool_datum_type:Nnnnn` §2.2.4.1; 23, 674
`\datatool_datum_value:Nnnnn` §2.2.4.1; 22, 271, 445, 523, 674
`\datatool_db_state:nnnn` .. §3.6; 236, 675
`\datatool_def_currency:nnn` §2.6; 125, 675
`\datatool_def_currency:nnnn` §2.6; 31, 125, 675
`\datatool_extract_time-`
 `stamp:NN` .. §2.2.4; 21, 136, 675
`\datatoolGBcurrencyfmt` 31, 124, 125, 675
`\datatool_get_first_`
 `grapheme:nN` . §2.8.3; 148, 675
`\datatool_get_first_`
 `letter:nN` §2.8.3; 49, 57, 148, 675
`\datatool_if_any_int_datum_`
 `type:nTF` §2.2.4; 22, 676
`\datatool_if_has_key:nnTF` §3.6; 237, 676
`\datatool_if_letter:nTF` §2.8.3; 51, 148, 676
`\datatool_if_null:NTF` .. §3.10.2; 316, 317, 676
`\datatool_if_null:nTF` .. §3.10.2; 317, 676
`\datatool_if_null_or_`
 `empty:NTF` §3.10.2; 316, 317, 676
`\datatool_if_null_or_`
 `empty:nTF` §3.10.2; 317, 632, 676
`\datatool_if_number_only_`
 `datum_type:nTF` §2.2.4; 22, 677
`\datatool_if_numeric_datum_`
 `type:nTF` §2.2.4; 22, 677
`\datatool_if_row_start:nnTF` §3.7.2; 253, 256, 277, 677
`\datatool_if_temporal_datum_`
 `_type:nTF` §2.2.4; 22, 677
`\datatool_if_valid_datum_`
 `type:nTF` §2.2.4; 21, 677
`\datatool_if_value_eq:NNTF` §2.2.4.2; 23, 677
`\datatool_if_value_eq:NnTF` §2.2.4.2; 23, 678
`\datatool_if_value_eq:nNTF` §2.2.4.2; 23, 678
`\datatool_if_value_eq:nnTF` §2.2.4.2; 23, 678
`\datatool_locale_define_`
 `keys:nn` 59, 678
`\datatool_map_keys_`
 `function:nN` §3.16.2; 375, 678
`\datatool_map_keys_in-`
 `line:nn` §3.16.2; 375, 679
`\datatool_max_known_type:` §2.2.4; 21, 679
`\datatool_measure:NNNn` . §2.8.3; 147, 679
`\datatool_measure_depth:Nn` §2.8.3; 147, 679, 793
`\datatool_measure_height:Nn` §2.8.3; 147, 679, 793
`\datatool_measure_ht_plus_`
 `dp:Nn` §2.8.3; 147, 679
`\datatool_measure_width:Nn` §2.8.3; 147, 679, 793
`\datatool_pad_trailing_`
 `zeros:Nn` §2.8.3; 146, 680
`\datatoolparen` ... §2.9.5.3; 167, 170, 171, 680
`\datatoolparenstart` §2.9.5.3; 167, 168, 170, 171, 598, 680
`\datatoolpersoncomma` §2.9.5.3; 47,

`\datatool_datum_currency:Nnnnn`832

`\datatoolpersoncomma`

- 166, 170, 598, 680
- \datatoolplacecomma §2.9.5.3; 166, 170, 598, 680
- \datatool_post_process_lettergroup:N §3.14.1.1; 326, 680
- \datatool_prefix_adjust_sign:nnn §2.6; 128, 672, 681, 703
- \datatool⟨Region⟩SetCurrency §2.3.2; 14, 45, 681
- \datatool⟨Region⟩SetNumberChars §2.3.2; 45, 54, 681
- \datatool⟨Region⟩symbolprefix §2.3.2; 44, 129, 681
- \datatool_register_regional_currency_code:nn . §2.3.2; 44, 681
- \datatool_set_apos_group_decimal_char:n §2.3.2; 43, 681
- \datatoolSetCurrencySort §2.9.5.3; 168, 682
- \datatool_set_currency_symbol:nn . §2.6; 125, 127, 682, *see also* \datatool_set_currency_symbol:nn & \DTLdefcurrency
- \datatool_set_fp:Nn . §2.2.4.3; 21, 26, 682
- \datatool_set_number_chars:nn ... §2.3.2; 42, 43, 681, 682, 683
- \datatool_set_number_chars:nnnn . §2.3.2; 42, 45, 682
- \datatool_set_numberchars_regex:nnnn §2.3.2; 42, 682
- \datatool_set_numberchars_regex_tl:nnnn §2.3.2; 43, 683
- \datatool_set_numberchars_tl_regex:nnnn §2.3.2; 43, 683
- \datatool_set_thinspace_group_decimal_char:n §2.3.2; 43, 683
- \datatool_set_underscore_group_decimal_char:n §2.3.2; 43, 683
- \datatool_sort_preprocess:Nn §2.9.5; 49, 51, 162, 683
- \datatool_sort_preprocess:NnN . §2.9.5; 162, 684
- \datatoolsubjectcomma . §2.9.5.3; 167, 170, 598, 684
- \datatool_suffix_adjust_sign:nnn §2.6; 128, 672, 684, 704
- \DataToolTimeFmt ... §2.7; 134, 137, 138, 674, 684
- \DataToolTimeStampFmtSep §2.7; 134, 137, 684
- \DataToolTimeStampNoZoneFmt §2.7; 134, 137, 684
- \DataToolTimeStampWithZoneFmt §2.7; 134, 137, 685
- \DataToolTimeZoneFmt . §2.7; 134, 137, 138, 674, 685
- datatooltk §3; b, 126, 176, 325, 337, 588
- datetime2 package 27, 134, 136, 339, 550, 651
- datetime2–english 136, 651
- datum control sequence 16–26, 62, 665
- datum item . 16–26, 151, 182, 227, 337, 338, 343, 345, 353, 665
- \DBIBCitekey §7.8; 550, 571, 685, 715
- \DBIBentrytype . §7.8; 550, 551, 571, 685, 715
- \DBIBname 685, 715
- decimal character . 12, 15, 16, 28, 42, 43, 45, 53, 62–65, 97, 108, 115, 116, 118, 119, 121, 128, 130, 360, 363, 369, 666, 681–683, 766
- decimal point (.) ... 12, 42, 62, 97, 146, 666
- \Description §8.9.1; 612, 619, 623, 685
- \directlua 3, 7, 8, 98, 104, 827
- \dotfill 586
- \DTLabs §2.5.2; 110, 111, 686, 718

- \dtlabs §2.5.1; 100, 111, 686
- \DTLaction §3.3; 177, 205–231, 337, 686
 - assign ... §3.3.2; 207, 208, 212, 213, 223, 226, 378, 400, 401
 - column ... §3.3.2; 209, 210, 214, 215, 217, 224, 225, 378, 401
 - column2 .. §3.3.2; 217, 219, 224, 225
 - columns §3.3.2; 216, 217, 220, 225, 401
 - datum §3.3.2; 217, 227, 228, 230, 282, 321
 - currency §3.3.2; 228
 - locale-decimal ... §3.3.2; 228
 - locale-integer ... §3.3.2; 228
 - round §3.3.2; 228, 229
 - expand-once-value §3.3.2; 178, 209, 210, 215, 227, 233, 317, 753
 - expand-value ... §3.3.2; 178, 209, 210, 215, 227, 233, 317, 753
 - key ... §3.3.2; 209, 210, 213–215, 217, 224, 378, 401
 - key2 §3.3.2; 217, 219, 224
 - keys .. §3.3.2; 216, 217, 220, 224, 225, 292, 401, 452
 - name .. §3.3.2; 207, 209, 210, 212, 214, 215, 217, 218, 220, 223, 224, 378, 400, 452
 - options §3.3.2; 206, 212, 215, 218–221, 223, 226, 242, 269, 378, 400, 401, 452, 453
 - return ... §3.3.2; 206, 207, 209, 211, 217, 219, 220, 227, 269, 310, 475
 - row §3.3.2; 212–215, 225
 - row2 §3.3.2; 212, 213, 226
 - type §3.3.2; 209, 210, 227
 - value §3.3.2; 180, 209, 210, 215, 226, 227, 367
- \DTLadd .. §2.5.2; 16, 19, 20, 42, 97, 108, 114, 269, 686, 718
- \dtladd .. §2.5.1; 8, 9, 20, 42, 97, 98, 108, 114, 686
- \dtladdalign ... §3.7.2; 256, 257, 686
- \DTLaddall .. §2.5.2; 20, 108, 110, 131, 149, 150, 686, 718
- \dtladdall §2.5.1; 20, 98, 99, 149, 150, 687
- \DTLaddcolumn ... §3.6; 240, 620, 687
- \DTLaddcolumnwithheader .. §3.6; 240, 687
- \dtladdheaderalign §3.7.1.3; 247, 687
- \DTLaddtoplotlegend . §6.3.2; 512, 513, 687
- \dtlaftercols §3.7.2; 246, 255, 257, 687
- \DTLafterinitialbefore-hyphen §2.8.2; 142, 687
- \DTLafterinitials §2.8.2; 141, 142, 688
- \dtlafterrow .. §3.16.1; 215, 318, 320, 371, 373, 688, 721, 761
- \DTLandlast §7.7.1; 545, 546, 688
- \DTLandname ... §2.9.2; 29, 58, 59, 151, 152, 688, 744
- \DTLandnotlast §7.7.1; 546, 688
- \DTLangLatnLocaleGetGroupString ... §2.3.5; 51, 52, 57, 688
- \DTLaposinitialpunc . §2.8.2; 141, 142, 144, 688
- \dtlappendentrytocurrentrow §3.16.1; 320, 374, 688, 773
- \DTLappendtorow . §3.8.2.1; 297, 689
- \DTLassign §3.16.1; 370, 373, 689
- \DTLassignfirstmatch §3.16; 370, 601, 689
- \DTLassignfromcurrentrow §3.16.1; 244, 372, 373, 689
- \DTLassignlettergroup §2.9.5; 49, 157, 158, 326, 689, 701, 702, 706, 744, 753, 759, 760, 772
- \DTLbaratbegintikz ... §5.4.4; 448, 449, 689
- \DTLbaratendtikz . §5.4.4; 438, 448, 449, 690
- \DTLbarchart .. §5; 400, 401, 406–409, 414, 415, 419, 423, 427, 443, 445, 446, 448, 449, 689, 690, 713, 714

- `\DTLbarchartlength` §5.4.1; 442, 690
- `\DTLbarchartwidth` §5.4.4; 449, 690
- `\DTLbardisplayYticklabel` §5.4.2; 406, 431, 445, 690
- `\DTLbargroupindex` §5.4.2; 443, 448, 690
- `\DTLbargrouplabelalign` . §5.4.2; 445, 690
- `\DTLbargroupwidth` §5.4.4; 449, 691
- `\DTLbarindex` §5.4.2; 437, 443, 448, 691
- `\DTLbarlabeloffset` ... §5.4.2; 406, 415, 416, 443, 691
- `\DTLbarmax` §5.4.1; 442, 691
- `\DTLbaroutlinecolor` §5.4.3; 446, 691
- `\DTLbaroutlinewidth` §5.4.3; 446, 691
- `DTLbarroundvar counter` §5.4.2; 406, 417, 442
- `\DTLbarsetupupperlabelalign` §5.4.2; 445, 691
- `\DTLBarStyle` ... §5.4.1; 412, 414, 435, 436, 442, 691
- `\DTLbartotalvariables` ... §5.4.4; 450, 692
- `\DTLbarvalue` ... §5.4.2; 401, 414, 415, 417, 420, 422, 424, 429, 443, 448, 692
- `\DTLbarvariable` ... §5.4.2; 401, 414, 415, 424, 443, 448, 692
- `\DTLbarwidth` ... §5.4.2; 443, 449, 692
- `\DTLBarXAxisStyle` §5.4.1; 442, 692
- `\DTLbarXlabelalign` ... §5.4.2; 406, 409, 443, 444, 692
- `\DTLbarXneglabelalign` ... §5.4.2; 409, 444, 692
- `\DTLbarXnegupperlabelalign` §5.4.2; 444, 445, 691, 692
- `\DTLbarXupperlabelalign` §5.4.2; 444, 445, 691, 693
- `\DTLBarYAxisStyle` §5.4.1; 442, 693
- `\DTLbarYticklabelalign` . §5.4.2; 445, 693
- `\dtlbeforecols` §3.7.2; 246, 255, 257, 693
- `\dtlbeforerow` §3.16.1; 215, 318, 320, 371, 373, 693, 721, 761
- `\dtlbetweencols` ... §3.7.2; 246, 255, 257, 693
- `\DTLbetweeninitials` . §2.8.2; 141, 142, 693
- `\DTLbibaccessedname` §7.7.1; 549, 693
- `\DTLbibdatefield` §7.8; 546, 551, 694
- `\DTLBIBdbname` §7.4; 537, 694, 745, 752
- `\DTLbibdoi` §7.7.1; 547, 548, 694
- `\DTLbibdoihome` §7.7.1; 548, 694
- `\DTLbibdoitag` §7.7.1; 548, 694
- `\DTLbibprints` §7.7.1; 547, 548, 694
- `\DTLbibfield` §7.8; 547, 550, 551, 694, 713
- `\DTLbibfieldcontains` §7.6; 541, 694
- `\DTLbibfieldexists` §7.6; 540, 695
- `\DTLbibfieldisseq` ... §7.6; 541, 695
- `\DTLbibfieldisge` ... §7.6; 542, 695
- `\DTLbibfieldisgt` ... §7.6; 542, 695
- `\DTLbibfieldisle` ... §7.6; 541, 695
- `\DTLbibfieldislt` ... §7.6; 541, 695
- `\DTLbibfieldlet` §7.8; 551, 695
- `\DTLbibformatdigital` §7.7.1; 547, 548, 549, 551, 696
- `\DTLbibitem` §7.7.1; 546, 696
- `\DTLbibliography` ... §7.6; 526, 536, 540, 542, 543, 546, 549, 552, 553, 560, 565, 696, 713, 716, 748
- `\DTLbibliographystyle` §7.7; 527, 543, 696
- `\DTLbibpubmed` .. §7.7.1; 547, 548, 696
- `\DTLbibpubmedhome` §7.7.1; 548, 696
- `\DTLbibpubmedtag` .. §7.7.1; 547, 696
- `DTLbibrow counter` §7.8; 542, 552, 564, 571, 715
- `\DTLbibsorthencap` §7.5; 326, 539, 697

- \DTLbibsortname §7.5; 539, 560, 697
- \DTLbibsortnamesep §7.5; 539, 697
- \DTLbiburl §7.7.1; 547, 548, 697
- \DTLbiburldate §7.7.1; 547, 549, 693, 697
- \dtlbreak §3.8.2; 286, 296, 375, 549, 565, 697
- \DTLcite §7.9; 553, 572, 697, 753
- \DTLclearbarcolors §5.4.3; 408, 413, 447, 698
- \DTLcleardb §3.5; 236, 698
- \DTLclearnegbarcolors §5.4.3; 447, 698
- \DTLclip §2.5.2; 112, 698, 718
- \dtlclip §2.5.1; 100, 112, 698
- \dtlcol §3.8.2; 297, 698
- \DTLcolumncount §3.6; 211, 237, 291, 698
- \dtlcolumnheader §3.7.1.3; 247, 252, 269, 698, 730
- \dtlcolumnindex §3.6; 214, 238, 319, 372, 515, 699
- \dtlcolumnnum 215, 243, 288, 699, 746
- \dtlcompare §2.9.5.1; 69, 160, 163, 164, 169, 170, 172, 174, 335, 699, 767, *see also* \dtlicompare
- \DTLcomputebounds §3.13; 321, 324, 699
- \DTLcomputewidestbibentry §7.8; 552, 571, 699
- \DTLconverttodecimal §2.2.2; 15, 16, 98, 111, 699
- \DTLcurr §2.6; 126, 700
- \DTLcurrChar §2.6; 126, 700
- \DTLcurrCodeOrSymOrChar §2.6; 15, 44, 130, 700, 703
- \dtlcurrdefaultfmt §2.6; 124–127, 128, 131, 675, 700, 708
- \DTLcurrency §2.6; 12, 66, 90, 117, 129, 130, 349, 362, 700
- \dtlcurrencyalign §3.7.2; 246, 254, 257, 700
- \DTLCurrencyCode §2.6; 44, 127, 700
- \dtlcurrencyformat §3.7.2; 253, 314, 700
- \dtlcurrencygroup §2.3.3; 53, 58, 158, 701, 759
- \DTLCurrencySymbol §2.6; 127, 701
- \DTLcurrencytype §3.6; 239, 701
- \DTLcurrentindex §3.8.2; 296, 701
- \DTLCurrentLocaleCurrencyDP §2.3.2; 28, 46, 115, 130, 701
- \DTLCurrentLocaleFormatDate 134, 138, 673, 701
- \DTLCurrentLocaleFormatTime 134, 138, 684, 701
- \DTLCurrentLocaleFormatTime-StampNoZone 134, 137, 684, 702
- \DTLCurrentLocaleFormatTime-StampWithZone 134, 137, 685, 702
- \DTLCurrentLocaleFormatTime-Zone 134, 138, 685, 702
- \DTLCurrentLocaleGetGroup-String §2.3.3; 49, 51, 60, 162, 702
- \DTLCurrentLocaleGetInitial-Letter §2.3.3; 49, 51, 57, 60, 142, 143, 158, 702, 759
- \DTLCurrentLocaleTimeStamp-FmtSep 134, 137, 684, 702
- \DTLCurrentLocaleWord-Handler §2.3.3; 46, 60, 158, 165, 702
- \dtlcurrentrow §3.16.1; 212, 214–216, 220, 244, 296, 318, 320, 371, 372–374, 688, 689, 693, 703, 720, 721, 761, 762, 771
- \DTLcurrEUR §2.6; 126, 703
- \dtlcurrfmtsep §2.6; 128, 130, 703
- \dtlcurrfmtsymsep §2.6; 129, 703
- \DTLcurr<ISO> §2.6; 124, 126, 129, 130, 166, 700, 703
- \dtlcurrprefixfmt §2.6; 128, 681, 700, 703
- \DTLcurrStr §2.6; 126, 703
- \dtlcurrsuffixfmt §2.6; 128, 684, 704

- \DTLcurrSym §2.6; 126, 704
- \DTLcurrXBT §2.6; 126, 704
- \DTLcurrXXX §2.6; 126, 704
- \DTLcustombibitem .. §7.8; 552, 704
- \DTLcustomlegend . §6.3.3; 457, 488, 514, 704
- \DTLdatatypecurrencyname §2.2.3; 18, 58, 704
- \DTLdatatypedatename . §2.2.3; 18, 58, 704
- \DTLdatatypedatetimenamename §2.2.3; 18, 58, 705
- \DTLdatatypedecimalname §2.2.3; 18, 58, 705
- \DTLdatatypeintegernamename §2.2.3; 18, 58, 705
- \DTLdatatypeinvalidnamename §2.2.3; 18, 58, 705
- \DTLdatatypestringname . §2.2.3; 18, 58, 705
- \DTLdatatypedetimenamename . §2.2.3; 18, 58, 705
- \DTLdatatypeunsetname §2.2.3; 18, 58, 705
- \dtldategroup 58, 706
- \dtldatetimegroup 58, 706
- \DTLdatumcurrency .. §2.2.3; 17, 19, 119, 706, 754
- \DTLdatumtype §2.2.3; 17, 19, 62, 288, 706, 754
- \DTLdatumvalue §2.2.3; 17, 19, 20, 26, 228, 230, 282, 288, 706, 754
- \dtldbcolreconstruct . §3.15.1.3; 345, 706, 707
- \dtldbdatumreconstruct §3.15.1.3; 345, 706
- \dtldbheaderreconstruct §3.15.1.3; 344, 706
- \dtldbname §3.16.1; 214–217, 220, 224, 286, 291, 372, 707, 721, 761
- \DTLdbNewEntry .. §3.15.1.2; 339, 707
- \DTLdbNewRow §3.15.1.2; 339, 707
- \DTLdbProvideData .. §3.15.1.2; 339, 344, 707
- \dtldbconstructkeyindex §3.15.1.3; 344, 707
- \dtldbrowreconstruct . §3.15.1.3; 345, 706, 707
- \DTLdbSetHeader §3.15.1.2; 340, 707
- \dtldbvaluereconstruct §3.15.1.3; 345, 707
- \DTLdecimaltocurrency §2.3.2; 15, 16, 45, 46, 115, 118, 130, 347, 350, 701, 708
- \DTLdecimaltolocale ... §2.3.2; 15, 16, 45, 115, 227, 347, 708
- \DTLdefaultEURcurrencyfmt §2.6; 117, 127, 708
- \dtldefaultkey ... §3.15.2; 210, 291, 346, 347, 356, 708
- \DTLDefaultLocaleWord-Handler .. §2.9.5.2; 46, 164, 165, 166, 167, 708, 767, 768
- \DTLdefcurrency ... §2.6; 15, 31, 44, 123, 124, 125, 127, 166, 675, 703, 708
- \DTLdeletedb §3.5; 235, 708
- \dtldisplayafterhead §3.7.2; 247, 248, 253, 257, 709
- \DTLdisplaybargrouplabel §5.4.2; 446, 709
- \dtldisplaycr . §3.7.2; 244, 249, 255, 257, 709
- \DTLdisplaydb §3.7; 238, 241–260, 709
- \DTLdisplaydb* . §3.7; 222, 223, 226, 241–260, 298, 709
- \DTLdisplaydbAddBegin ... §3.7.2; 257, 258, 709
- \DTLdisplaydbAddEnd . §3.7.2; 249, 257, 258, 709
- \DTLdisplaydbAddItem §3.7.2; 243, 253, 255, 256, 264, 267, 270, 271, 275–280, 282, 562, 677, 709
- \dtldisplaydbenv . §3.7.2; 243, 245, 253, 257, 258, 710
- \dtldisplayendtab §3.7.2; 249, 253, 258, 710

- \DTLdisplayinnerlabel §4.5; 395, 710
- \DTLdisplaylongdb .. §3.7; 223, 242, 241–260, 305, 686, 687, 693, 698, 700, 709, 710, 711, 738, 754, 760, 770
- \DTLdisplaylongdbAddBegin §3.7.2; 249, 250, 258, 710
- \DTLdisplaylongdbAddEnd §3.7.2; 258, 710
- \dtldisplaylongdbenv §3.7.2; 245, 254, 710
- \DTLdisplaylowerbarlabel §5.4.2; 445, 711
- \DTLdisplaylowermultibar-label §5.4.2; 446, 711
- \DTLdisplayouterlabel §4.5; 395, 711
- \dtldisplaystartrow . §3.7.2; 244, 253, 711
- \dtldisplaystarttab . §3.7.2; 247, 253, 257, 711
- \DTLdisplayTBrowidxmap . §3.7.2; 244, 258, 260, 711
- \DTLdisplayupperbarlabel §5.4.2; 446, 711
- \DTLdisplayuppermultibar-label §5.4.2; 446, 711
- \dtldisplayvalign §3.7.2; 245, 253, 257, 712
- \DTLdiv §2.5.2; 110, 712, 719
- \dtldiv §2.5.1; 99, 110, 712
- \DTLdobarcolor §5.4.3; 447, 448, 712
- \DTLdocurrentbarcolor ... §5.4.3; 447, 712
- \DTLdocurrentpiesegment-color §4.6; 397, 712
- \DTLdopiesegmentcolor §4.6; 396, 397, 712
- \DTLencapbibfield §7.8; 550, 551, 713
- \DTLendbibitem §7.7.1; 536, 546, 547, 564, 713
- \DTLendpt §5.4.4; 436, 449, 713
- \DTLenLocaleGetGroupString 56, 60, 713
- \DTLenLocaleHandler ... §2.3.5; 46, 48, 56, 60, 61, 713
- DTLenvforeach environment §3.8.2; 293, 294, 823
- DTLenvforeach* environment §3.8.2; 294, 823
- dtlenvgforint environment .. §3.16.2; 376, 823
- dtlenvgforint* environment . §3.16.2; 376, 823
- DTLenvmapdata environment ... §3.8.1; 286, 288, 289, 746, 823
- \DTLeverybargrouphook ... §5.4.4; 408, 449, 713
- \DTLeverybarhook . §5.4.4; 407, 408, 438, 439, 448, 713, 748, 769
- \DTLeveryprebarhook . §5.4.4; 436, 448, 714
- \dtlexpandnewvalue §3.1; 179, 714
- \dtlfallbackaction ... §2.9.5; 159, 167, 174, 175, 714
- \DTLfetchlistelement §2.9.3; 153, 714
- \DTLfmtcurr §2.6; 12, 14, 90, 117, 129, 714
- \DTLfmtcurrency §2.6; 12, 14, 44, 90, 116, 117, 127, 128–131, 700, 714
- \dtlforcolumn §3.16.2; 375, 714
- \dtlforcolumnidx §3.16.2; 375, 715
- \DTLforeach §3.8.2; 212, 220, 221, 238, 264, 285, 286, 288, 293–298, 301, 302, 304, 318, 370, 373, 377, 394, 397, 399, 448, 451, 526, 540, 549, 577, 689, 697, 701, 703, 715, 732, 734, 736, 761–763, 823
- \DTLforeachbibentry ... §7.8; 526, 536, 542, 543, 549, 550, 552, 553, 571, 685, 694–696, 715, 716, 731, 777
- \dtlforeachkey §3.16.2; 297, 374, 715
- \DTLforeachkeyinrow 297, 698, 715, 730, 743, 773, *see also* \dtlkey, \dtlcol, \dtltype & \dtlheader

- \dtlforeachlevel . §3.8.2; 294, 295, 715, 763, 772
- \dtlforint §3.16.2; 375, 716, 721
- \DTLformatabbrvforenames §7.7.1; 544, 716
- \DTLformatauthor . §7.7.1; 543, 544, 545, 562, 567, 716
- \DTLformatauthorlist §7.7.1; 545, 716
- \DTLformatbibentry §7.8; 551, 552, 571, 716, 718, 777
- \DTLformatbibnamelist ... §7.7.1; 545, 562, 716
- \DTLformatdate §7.7.1; 546, 551, 716
- \DTLformatteditor . §7.7.1; 543, 544, 545, 717
- \DTLformatteditorlist §7.7.1; 545, 716, 717
- \DTLformatforenames . §7.7.1; 544, 716, 717
- \DTLformatjr §7.7.1; 544, 717
- \DTLformatlegend §6.3.3; 489, 514, 717
- \DTLformatlist §2.9.2; 151, 152, 153, 155, 170, 717, 744, 745
- \DTLformatpages ... §7.7.1; 546, 717
- \DTLformatsurname §7.7.1; 544, 717
- \DTLformatsurnameonly ... §7.7.1; 544, 718
- \DTLformatthisbibentry ... §7.8; 543, 552, 685, 718
- \DTLformatvon §7.7.1; 544, 718
- \DTLgabs §2.5.2; 111, 718
- \DTLgadd §2.5.2; 108, 718
- \DTLgaddall §2.5.2; 110, 718
- \DTLgcleardb §3.5; 236, 718
- \DTLgclip §2.5.2; 112, 718
- \DTLgdeletedb §3.5; 235, 719
- \DTLgdiv §2.5.2; 110, 719
- \DTLget §3.3; 206, 719
- \DTLgetbarcolor §5.4.3; 447, 712, 719
- \DTLgetcolumnindex §3.6; 213, 237, 238, 719
- \DTLgetdatatype §3.6; 238, 719
- \DTLgetDataTypeName §2.2.3; 18, 719
- \dtlgetentryfromcurrentrow §3.16.1; 216, 244, 251, 372, 720
- \dtlgetentryfromrow 720
- \DTLGetInitialLetter . §2.8.2; 10, 49, 141, 142, 145, 702, 720, 769
- \DTLgetkeyforcolumn §3.6; 238, 720
- \DTLgetlocation §3.16; 370, 720
- \DTLgetnegbarcolor ... §5.4.3; 447, 712, 720
- \DTLgetpiesegmentcolor ... §4.6; 396, 720
- \dtlgetrow §3.16.1; 212, 214, 216, 220, 318, 319, 372, 373, 689, 703, 721
- \dtlgetrowforvalue .. §3.16.1; 215, 222, 319, 372, 373, 721, 776
- \DTLgetrowindex §3.16; 370, 721
- \dtlgetrowindex ... §3.16; 369, 370, 373, 721, 821
- \DTLgetvalue §3.16; 370, 721
- \dtlgetforint §3.16.2; 375, 376, 721, 823
- \DTLgidxAcrStyle §8.7; 606, 609, 722
- \DTLgidxAssignList ... §8.9.1; 604, 610, 619, 620, 623, 624, 722
- \DTLgidxCategoryNameFont §8.8.2.5; 616, 722
- \DTLgidxCategorySep §8.8.2.5; 616, 722
- DTLgidxChildCount counter . §8.9.3; 612, 622
- \DTLgidxChildCountLabel §8.8.1; 579, 583, 612, 622, 722
- \DTLgidxChildSep §8.8.2.4; 615, 722
- \DTLgidxChildStyle ... §8.8.1; 612, 622, 722
- \DTLgidxCounter ... §8.5.2; 603, 722
- \DTLgidxCurrentdb .. §8.8; 610, 616, 617, 619, 723
- \DTLgidxDictHead §8.8.2.5; 615, 616, 723
- \DTLgidxDictPostItem ... §8.8.2.5;

- 616, 723
- \DTLgidxDisableHyper .. §8.9; 600, 618, 723
- \DTLgidxEnableHyper §8.9; 618, 723
- \DTLgidxEndItem ... §8.8.1; 613, 723
- \DTLgidxFetchEntry §8.5; 601, 723
- \DTLgidxForeachEntry §8.9.1; 610, 619, 620, 671, 722, 723
- \DTLgidxFormatAcr §8.7.1; 608, 724
- \DTLgidxFormatAcrUC §8.7.1; 609, 724
- \DTLgidxFormatDesc §8.8.1; 612, 724
- \DTLgidxFormatSee §8.8.1; 613, 614, 724, 727
- \DTLgidxFormatSeeAlso ... §8.8.1; 613, 724
- \DTLgidxGobble §8.9.2; 621, 724
- \DTLgidxGroupHeaderTitle §8.8; 610, 616, 724
- \DTLgidxIgnore §8.4.1; 596, 599, 724
- \DTLgidxLocation 603, 725
- \DTLgidxMac §8.4.1; 595, 598, 725
- \DTLgidxName ... §8.4.1; 591, 597, 598, 610, 725
- \DTLgidxNameCase . §8.8.1; 585, 611, 612, 725
- \DTLgidxNameFont . §8.8.1; 585, 611, 612, 725
- \DTLgidxNameNum §8.4.1; 592, 599, 725
- \DTLgidxNoFormat . §8.9.2; 597, 598, 621, 725
- \DTLgidxOffice §8.4.1; 593, 594, 597, 598, 610, 725
- \DTLgidxParen §8.4.1; 595, 597, 598, 726
- \DTLgidxParticle . §8.4.1; 594, 597, 599, 726
- \DTLgidxPlace . §8.4.1; 592, 597, 598, 610, 726
- \DTLgidxPostChild §8.8.2.4; 615, 726
- \DTLgidxPostChildName ... §8.8.1; 612, 726
- \DTLgidxPostDescription §8.8.1; 585, 612, 726
- \DTLgidxPostLocation §8.8.1; 613, 726
- \DTLgidxPostName §8.8.1; 585, 612, 726
- \DTLgidxPreLocation . §8.8.1; 586, 613, 727
- \DTLgidxRank ... §8.4.1; 594, 599, 727
- \DTLgidxSaint .. §8.4.1; 595, 598, 727
- \DTLgidxSeeTagFont §8.8.1; 613, 727
- \DTLgidxSetColumns ... §8.8.1; 583, 611, 727
- \DTLgidxSetCompositor ... §8.5.2; 604, 727
- \DTLgidxSetDefaultDB 584, 588, 727
- \DTLgidxStripBackslash . §8.4.1; 596, 727
- \DTLgidxSubCategorySep §8.8.2.5; 616, 728
- \DTLgidxSubject ... §8.4.1; 592, 593, 597, 598, 610, 728
- \DTLgidxSymDescSep §8.8.1; 612, 728
- \DTLgmax §2.5.2; 113, 728
- \DTLgmaxall §2.5.2; 113, 728
- \DTLgmeanforall ... §2.5.2; 113, 728
- \DTLgmin §2.5.2; 112, 728
- \DTLgminall §2.5.2; 113, 728
- \DTLgmul §2.5.2; 110, 729
- \DTLgneg §2.5.2; 111, 729
- \DTLgnewdb §3.4; 232, 729, 752
- \DTLground §2.5.2; 112, 729
- \DTLgsdforall §2.5.2; 114, 729
- \DTLgsqrt §2.5.2; 111, 729
- \DTLgsub §2.5.2; 110, 729
- \DTLgtrunc §2.5.2; 112, 729
- \DTLgvarianceforall §2.5.2; 114, 730
- \dtlheader §3.8.2; 297, 730

- \dtlheaderformat . §3.7.2; 247, 252, 257, 730
- \dtlcompare . §2.9.5.1; 69, 160, 163, 164, 169–172, 335, 730, 767, *see also* \dtlcompare
- \DTLidxFormatSeeItem §8.8.1; 613, 730
- \DTLidxSeeLastSep §8.8.1; 614, 730
- \DTLidxSeeSep §8.8.1; 614, 730
- \DTLifaction §3.3; 206, 730
- \DTLifAllLowerCase §2.4.1.2; 71, 731
- \DTLifAllUpperCase §2.4.1.2; 71, 731
- \DTLifanybibfieldexists .. §7.8; 547, 551, 731
- \DTLifbibfieldexists .. §7.8; 547, 551, 731
- \DTLifcasedatatype .. §2.4.1.1; 62, 64, 68, 731
- \DTLifclosedbetween . §2.4.1.5; 88, 92, 731, 738, 740
- \DTLifcurrency . §2.4.1.1; 63, 66, 67, 90, 94, 731, 739
- \DTLifcurrencyunit .. §2.4.1.1; 63, 66, 90, 732, 739
- \DTLifdbempty §3.6; 236, 732
- \DTLifdbexists .. §3.6; 236, 372, 732
- \DTLifEndsWith . §2.4.1.2; 71, 79, 93, 732, 741, 743
- \DTLifeq §2.4.1.5; 62, 87, 90, 732, 739, 740
- \DTLiffirstrow §3.8.2; 296, 301, 732
- \DTLifFPclosedbetween . §2.4.1.4; 84, 732, *see* \dtlifnumclosedbetween
- \DTLifFPopenbetween §2.4.1.4; 83, 733, *see* \dtlifnumopenbetween
- \DTLifgt . §2.4.1.5; 88, 91, 282, 733, 740
- \DTLifhaskey §3.6; 237, 733
- \DTLifinlist §2.4.1.2; 69, 71, 93, 149, 150, 733, 740
- \DTLifint §2.4.1.1; 62, 64, 89, 733, 741
- \dtlifintclosedbetween §2.4.1.4; 84, 733
- \dtlifintopenbetween ... §2.4.1.4; 83, 733
- \DTLiflastrow §3.8.2; 296, 734
- \DTLiflt . §2.4.1.5; 87, 91, 734, 740, 741
- \DTLifnull .. §3.10; 214, 288, 289, 311, 315–317, 734
- \DTLifnullloempty §3.10; 311, 314–317, 734
- \DTLifnumclosedbetween §2.4.1.3; 82, 88, 93, 731, 734, 741
- \dtlifnumclosedbetween §2.4.1.4; 82, 83, 84, 732, 734
- \DTLifnumeq ... §2.4.1.3; 62, 81, 87, 90, 732, 734, 742
- \dtlifnumeq §2.4.1.4; 62, 81, 83, 86, 87, 271, 735
- \DTLifnumerical §2.4.1.1; 63, 64, 67, 90, 735, 742
- \DTLifnumgt . §2.4.1.3; 81, 88, 91, 733, 735, 742
- \dtlifnumgt §2.4.1.4; 81, 83, 735
- \DTLifnumlt ... §2.4.1.3; 81, 87, 91, 92, 300, 734, 735, 742
- \dtlifnumlt §2.4.1.4; 81, 83, 735
- \DTLifnumopenbetween §2.4.1.3; 81, 88, 92, 735, 736, 742
- \dtlifnumopenbetween §2.4.1.4; 81, 83, 733, 736
- \DTLifoddrow ... §3.8.2; 296, 298, 736
- \DTLifopenbetween §2.4.1.5; 88, 92, 736, 741, 742
- \DTLifreal §2.4.1.1; 63, 65, 90, 736, 743
- \DTLifStartsWith .. §2.4.1.2; 70, 71, 77, 93, 567, 736, 741, 743
- \DTLifstring §2.4.1.1; 63, 68, 90, 736, 743
- \DTLifstringclosedbetween §2.4.1.2; 70, 88, 731, 736
- \DTLifstringeq . §2.4.1.2; 23, 62, 70, 87, 163, 386, 423, 455, 567, 732, 737
- \DTLifstringgt §2.4.1.2; 70, 88, 159,

- 163, 174, 714, 733, 737
`\DTLifstringlt` . §2.4.1.2; 62, 70, 74, 87, 163, 734, 737
`\DTLifstringopenbetween` §2.4.1.2; 70, 88, 736, 737
`\DTLifSubString` §2.4.1.2; 70, 71, 76, 93, 737, 741, 743
`\DTLiftemporal` . . . §2.4.1.1; 63, 737
`\DTLinitialhyphen` §2.8.2; 142, 737
`\DTLinitialpunc` §2.8.2; 141, 142, 738
`\DTLinitials` §2.8.2; 50, 140, 687, 688, 693, 737, 738, 821
`\dtlininsertinto` §2.9.4; 155, 162, 738
`\dtlintalign` §3.7.2; 245, 254, 257, 738
`\dtlintformat` . . §3.7.2; 252, 314, 738
`\DTLinttype` §3.6; 239, 738
`\DTLisclosedbetween` §2.4.2; 92, 738
`\DTLiscurrency` . . §2.4.2; 90, 94, 739
`\DTLiscurrencyunit` §2.4.2; 90, 739
`\DTLiseq` §2.4.2; 90, 739
`\DTLisFPclosedbetween` . . §2.4.2; 93, 739, *see*
`\DTLisnumclosedbetween`
`\DTLisFPeq` §2.4.2; 90, 739, *see*
`\DTLisnumeq`
`\DTLisFPgt` §2.4.2; 92, 739, *see*
`\DTLisnumgt`
`\DTLisFPgteq` §2.4.2; 92, 739, *see*
`\DTLisnumeq`
`\DTLisFPlt` §2.4.2; 91, 739, *see*
`\DTLisnumlt`
`\DTLisFPlteq` §2.4.2; 91, 740, *see*
`\DTLisnumeq`
`\DTLisFPopenbetween` §2.4.2; 92, 740, *see*
`\DTLisnumopenbetween`
`\DTLisgt` §2.4.2; 91, 740
`\DTLisiclosedbetween` . . . §2.4.2; 92, 740
`\DTLisieq` §2.4.2; 90, 740
`\DTLisigt` §2.4.2; 91, 740
`\DTLisilt` §2.4.2; 91, 740
`\DTLisinlist` §2.4.2; 93, 740
`\DTLisint` §2.4.2; 89, 741
`\DTLisiopenbetween` §2.4.2; 92, 741
`\DTLisiPrefix` §2.4.2; 93, 741
`\DTLisiSubString` . . . §2.4.2; 93, 741
`\DTLisiSuffix` §2.4.2; 93, 741
`\DTLislt` §2.4.2; 91, 741
`\DTLisnumclosedbetween` . . §2.4.2; 93, 739, 741
`\DTLisnumeq` §2.4.2; 90, 739, 742
`\DTLisnumerical` . . . §2.4.2; 90, 742
`\DTLisnumgt` §2.4.2; 91, 92, 739, 742
`\DTLisnumgteq` . . . §2.4.2; 92, 739, 742
`\DTLisnumlt` §2.4.2; 91, 739, 742
`\DTLisnumlteq` . . . §2.4.2; 91, 740, 742
`\DTLisnumopenbetween` . . §2.4.2; 92, 740, 742
`\DTLisopenbetween` . . §2.4.2; 92, 742
`\DTLisPrefix` §2.4.2; 93, 569, 743
`\DTLisreal` §2.4.2; 90, 743
`\DTLisstring` §2.4.2; 90, 743
`\DTLisSubString` §2.4.2; 93, 743
`\DTLisSuffix` §2.4.2; 93, 743
`\dtlkey` §3.8.2; 297, 743
`\dtllastloadeddb` §3.15.3; 339, 342, 344, 360, 584, 743
`\DTLlegendxoffset` . . . §6.4.1; 487, 520, 743
`\DTLlegandyoffset` . . . §6.4.1; 487, 520, 744
`\dtllettergroup` . §2.3.3; 52, 58, 158, 160, 744, 759
`\dtlletterindexcompare` §2.9.5.1; 163, 165, 166, 169, 173, 175, 673, 680, 684, 744, 768, *see also*
`\dtlwordindexcompare`
`\DTLlistand` §2.9.2; 151, 152, 546, 744
`\DTLlistelement` . . . §2.9.3; 153, 744
`\DTLlistformatitem` . . . §2.9.2; 151, 152, 744
`\DTLlistformatlastsep` . . §2.9.2; 151, 152, 645, 744
`\DTLlistformatoxford` §2.9.2; 152,

- 645, 744
- `\DTLlistformatsep` §2.9.2; 151, 645, 745
- `\DTLloadbbl` . §7.3; 526, 527, 536, 537, 538, 540, 541, 553, 560, 694, 745, 752
- `\DTLloaddb`  ... §3.15.3; 364, 745, *see* `\DTLread`
- `\DTLloaddbtex`  .. §3.15.3; 363, 745, *see* `\DTLread`
- `\DTLloadmbbl` §7.9; 526, 537, 538, 553, 745, 748
- `\DTLloadrawdb`  §3.15.3; 350, 364, 745, *see* `\DTLread`
- `\DTLmajorgridstyle` ... §6.4.3; 466, 498, 524, 745
- `\dtlmapcurrentrow` §3.16.1; 374
- `\DTLmapdata` §3.8.1; 220, 285–293, 746
- `allow-edits` §3.8.1; 285, 289, 297, 318
- `name` §3.8.1; 285
- `read-only` ... §3.8.1; 285, 286, 289, 619, 762, 765
- `\DTLmapdata edit options`
- `column` §3.8.1.2; 290, 291
- `expand-once-value` ... §3.8.1.2; 290, 291
- `expand-value` .. §3.8.1.2; 290, 291
- `key` §3.8.1.2; 290, 291
- `value` §3.8.1.2; 290, 291
- `\DTLmapdatabreak` . §3.8.1; 285, 286, 289, 746
- `\DTLmapget` §3.8.1.1; 287, 288, 290, 310, 311, 315, 422, 454, 647, 746
- `column` §3.8.1.1; 287, 288
- `key` §3.8.1.1; 287, 288
- `return` §3.8.1.1; 288, 311, 315
- `\DTLmapgetvalues` . §3.8.1.1; 17, 289, 420, 454, 746
- `\DTLmaprow` §3.8.1.1; 288, 289, 290, 746
- `\DTLmaprowbreak` §3.8.1.1; 289, 296, 746
- `\DTLmax` §2.5.2; 113, 728, 746
- `\dtlmax` §2.5.1; 100, 113, 747
- `\DTLmaxall` §2.5.2; 113, 728, 747
- `\dtlmaxall` §2.5.1; 100, 747
- `DTLmaxauthors counter` §7.7.1; 545
- `DTLmaxeditors counter` §7.7.1; 545
- `\DTLmaxforcolumn` .. §3.13; 324, 747
- `\DTLmaxforkeys` §3.13; 218, 238, 323, 747
- `\DTLmaxX` §6.3.1; 452, 511, 747
- `\DTLmaxY` §6.3.1; 452, 512, 747
- `\DTLmbibitem` §7.7.1; 546, 748
- `\DTLmbibliography` .. §7.9; 543, 546, 553, 572, 713, 748
- `\DTLmeanforall` §2.5.2; 113, 114, 728, 748
- `\dtlmeanforall` §2.5.1; 101, 748
- `\DTLmeanforcolumn` . §3.13; 322, 748
- `\DTLmeanforkeys` §3.13; 322, 748
- `\DTLmidpt` §5.4.4; 436, 449, 748
- `\DTLmin` §2.5.2; 112, 113, 728, 749
- `\dtlmin` §2.5.1; 100, 112, 113, 749
- `\DTLminall` §2.5.2; 113, 728, 749
- `\dtlminall` §2.5.1; 100, 749
- `\DTLminforcolumn` .. §3.13; 323, 749
- `\DTLminforkeys` §3.13; 323, 749
- `\DTLminminortickgap` . §6.4.1; 461, 520, 749
- `\DTLminorgridstyle` ... §6.4.3; 466, 498, 524, 750
- `\DTLminorticklength` §6.4.1; 520, 750
- `\DTLmintickgap` §6.4.1; 419, 461, 520, 750
- `\DTLminX` §6.3.1; 452, 511, 750
- `\DTLminY` §6.3.1; 452, 511, 750
- `\DTLmonthname` §7.3; 534, 537, 538, 750
- `\DTLmul` §2.5.2; 110, 269, 729, 750
- `\dtlmul` §2.5.1; 20, 99, 110, 751
- `\DTLmultibarchart` ... §5; 400, 401, 406–409, 411, 414, 415, 419, 427, 428, 443, 445, 446, 448–450, 689–692, 711, 713, 714, 751, 772
- `\DTLmultibibs` §7.9; 552, 553, 697, 751

- \DTLneg §2.5.2; 111, 729, 751 398, 755
- \dtlneg §2.5.1; 101, 111, 751 \DTLpieatendtikz ... §4.7; 398, 755
- \DTLnegextent §5.4.1; 442, 751 \DTLpieatsegment ... §4.7; 397, 755
- \DTLnewbibitem §7.4; 533, 535, 537, 752 \DTLpiechart .. §4; 377, 378, 379, 381, 394, 397, 401, 423, 755
- \DTLnewbibliteralitem §7.4; 533, 535, 538, 548, 752 \DTLpieoutlinecolor §4.6; 397, 755
- \DTLnewbibrow §7.4; 537, 752 \DTLpieoutlinewidth ... §4.6; 393, 397, 755
- \DTLnewcurrencysymbol . §2.6; 12, 66, 123, 125, 752 \DTLpiepercent . §4.4; 378, 383, 385, 394, 395, 755
- \DTLnewdb §3.4; 207, 232, 337–339, 354, 537, 559, 587, 752 DTLpieroundvar counter §4.4; 395
- \DTLnewdbentry . §3.4; 208–210, 232, 233, 235, 339, 343, 707, 752 \DTLpievariable §4.4; 378, 383, 394, 755
- \DTLnewrow §3.4; 208, 232, 339, 707, 752 \DTLplot ... §6; 451, 452–455, 458, 473, 484, 510–515, 524, 525, 672, 747, 750, 756, 785
- \DTLnocite §7.9; 553, 753 \DTLplotatbegintikz ... §6.3; 453, 457, 510, 511, 672, 756
- \dtlnoexpandnewvalue §3.1; 179, 753 \DTLplotatendtikz .. §6.3; 453, 511, 513, 756
- \dtlnonlettergroup . §2.3.3; 53, 56, 58, 158, 160, 753, 760 \DTLplotdisplayticklabel §6.4.3; 462, 504, 523, 756
- \dtlnovalue . §3.10.2; 17, 18, 315, 316, 317, 753 \DTLplotdisplayXticklabel §6.4.3; 462, 523, 756
- \dtlnumbergroup . §2.3.3; 53, 58, 157, 753, 759 \DTLplotdisplayYticklabel §6.4.3; 462, 523, 756
- \DTLnumbernulld §3.10.2; 17, 213, 316, 317, 753 \dtlplothandlermark §6.5; 524, 756
- \DTLnumcompare ... §2.9.5.1; 163, 164, 335, 753 \DTLplotheight §6.4.1; 464, 465, 521, 756
- \dtlnumericformat §3.7.2; 252, 253, 314, 700, 738, 754, 760 \DTLplotlegendname ... §6.3.3; 484, 516, 757, 784
- \DTLnumitemsinlist §2.9.3; 153, 754 \DTLplotlegendnamesep ... §6.3.3; 517, 757
- \dtlpadleadingzeros ... §2.5.1; 98, 175, 754 \DTLplotlegendsetname ... §6.3.3; 516, 757, 784
- \dtlpadleadingzerosminus §2.5.1; 98, 754 \DTLplotlegendsetxlabel §6.3.3; 517, 757, 785
- \dtlpadleadingzerosplus §2.5.1; 98, 754 \DTLplotlegendsetylabel §6.3.3; 518, 757, 758, 785
- \DTLpar §3.4; 235, 549, 754 \DTLplotlegendx ... §6.3.3; 480, 481, 484, 513, 517, 757, 785
- \DTLparse §2.2.3; 10, 11, 14, 16, 17, 19, 26, 44, 62, 63, 114, 116, 119, 137, 228, 273, 706, 754, 773, 775 \DTLplotlegendxy . §6.3.3; 485, 487,
- \DTLpieatbegintikz §4.7; 397,

- 517, 757, *see also*
`\DTLplotlegendx`,
`\DTLplotlegendy` &
`\DTLplotlegendxysep`
`\DTLplotlegendxysep` . §6.3.3; 485,
517, 758
`\DTLplotlegendy` ... §6.3.3; 480, 481,
484, 518, 757, 758, 785
`\DTLplotlinecolors` ... §6.4.3; 460,
521, 758
`\DTLplotlines` .. §6.4.3; 461, 522, 758
`\DTLplotmarkcolors` ... §6.4.3; 461,
522, 758
`\DTLplotmarks` §6.4.3; 461, 522,
523, 758
DTLplotroundXvar counter §6.4.2; 521
DTLplotroundYvar counter §6.4.2; 521
`\DTLplotstream` §6.5; 524, 758
`\DTLplotwidth` .. §6.4.1; 464, 521, 759
`\DTLpostpagename` .. §7.7.1; 546, 759
`\DTLpostvon` §7.7.1; 544, 759
`\DTLPreProcessCurrencyGroup`
§2.9.5; 158, 759
`\DTLPreProcessDecimalGroup`
§2.9.5; 158, 759
`\DTLPreProcessIntegerGroup`
§2.9.5; 157, 759
`\DTLPreProcessLetterGroup`
§2.9.5; 158, 759
`\DTLPreProcessNonLetter-`
Group §2.9.5; 158, 760
`\DTLprotectedsaverawdb` 
§3.15.4; 365, 760, *see* `\DTLwrite`
`\DTLrawmap` §3.15.2; 350, 760
`\DTLread` ... §3.15.3; 177, 181, 184, 188,
203, 232, 239, 336–339, 344,
346–349, 351–356, 358, 360, 364,
708, 743, 760
`\dtlrealalign` §3.7.2; 246, 254,
257, 760
`\dtlrealformat` §3.7.2; 252, 270,
314, 760
`\DTLrealtype` §3.6; 239, 760
`\dtlrecombine` §3.16.1; 318, 320,
373, 761
`\dtlrecombineomitcurrent`
§3.16.1; 318, 320, 373, 761
`\DTLreconstructdatabase`
§3.15.1.3; 344, 706, 707, 761
`\DTLreconstructdbdata`  761, *see*
`\DTLreconstructdata-`
base
`\DTLremovecurrentrow` ... §3.8.2.1;
298, 761
`\DTLremoveentryfromrow` §3.8.2.1;
297, 298, 761
`\dtlremoveentryincurrentrow`
§3.16.1; 374, 761
`\DTLreplaceentryforrow` §3.8.2.1;
297, 762
`\dtlreplaceentryincurrent-`
row ... §3.16.1; 319, 374, 703, 762
`\DTLresetLanguage` ... §2.3; 28, 762
`\DTLresetRegion` §2.3; 28, 762
`\DTLrmentry` §3.8.1.2; 290, 762
`\DTLrmrow` §3.8.1.2; 285, 290, 762
`\dtlroot` §2.5.1; 99, 111, 762
`\DTLround` §2.5.2; 111, 112, 269,
729, 763
`\dtlround` §2.5.1; 99, 111, 227, 763
DTLrow counter §3.8.2; 295, 763
`\DTLrowcount` §3.6; 237, 290, 372, 763
DTLrowi counter §3.8.2; 294, 295, 304
DTLrowii counter §3.8.2; 295, 304
DTLrowiii counter §3.8.2; 295, 304
`\DTLrowincr` §3.8.2; 286, 295, 763
`\dtlrownum` §3.16.1; 215, 220, 243, 244,
250, 286, 365, 371, 408, 437, 448,
721, 746, 763
`\DTLrowreset` §3.8.2; 266, 286,
295, 763
`\DTLsavedb`  ... §3.15.4; 364, 763, *see*
`\DTLwrite`
`\DTLsavelastrowcount` §3.8.2;
294, 763
`\DTLsaverawdb`  §3.15.4; 364,
760, 764, *see* `\DTLwrite`
`\DTLsavetexdb`  .. §3.15.4; 365, 764,

- see* \DTLwrite
- \DTLscinum §2.2.1; 12, 14, 764
- \DTLsdforall ... §2.5.2; 114, 729, 764
- \dtlsdforall §2.5.1; 101, 764
- \DTLsdforcolumn ... §3.13; 323, 764
- \DTLsdforkeys §3.13; 323, 764
- \DTLsetbarcolor ... §5.4.3; 412, 413, 446, 447, 765
- \DTLsetcurrencydatum §2.2.3; 19, 765
- \DTLsetdecimaldatum... §2.2.3; 18, 19, 765
- \DTLsetdefaultcurrency . §2.3.2; 15, 44, 126, 127, 129, 765, *see also* \DTLdefcurrency
- \DTLsetdelimiter §3.15.2; 177, 353, 765
- \DTLsetentry . §3.8.1.2; 285, 291, 765
- \DTLsetfpdatum §2.2.3; 18, 19, 26, 765
- \DTLsetheader ... §3.6; 240, 339, 358, 707, 765
- \DTLsetintegerdatum §2.2.3; 18, 766
- \DTLsetLocaleOptions §2.3; 8, 30, 119, 657, 766
- \DTLsetnegbarcolor §5.4.3; 447, 766
- \DTLsetnumberchars . §2.3.2; 11, 12, 15, 42, 54, 65, 108, 666, 766
- \DTLsetpiesegmentcolor ... §4.6; 383, 396, 766
- \DTLsetseparator §3.15.2; 181, 359, 766
- \DTLsetstringdatum §2.2.3; 19, 766
- \DTLsettabseparator §3.15.2; 181, 337, 359, 766
- \DTLsetup §2.1; 8, 767, *see also*
 - \DTLsetLocaleOptions
 - auto-reformat-types* §2.1; 10, 11, 14, 133, 134, 358, 368
 - bar* §5.2; 401, 406
 - axes* §5.2.4; 416
 - both* 416
 - none* 416
 - x* 416
 - y* 416
 - bar-colors* §5.2.2; 407, 412, 432, 435, 446
 - bar-default-colors* . §5.2.2; 406, 413, 446
 - bar-default-gray* §5.2.2; 406, 413, 446
 - bar-gap* §5.2.2; 411
 - bar-label* §5.2.3; 414
 - bar-width* .. §5.2.2; 399, 411, 443
 - bargap* *see* *bar-gap*
 - barlabel* *see* *bar-label*
 - barwidth* *see* *bar-width*
 - color-style* ... §5.2.2; 411, 414, 435, 436, 719
 - cycle* 412
 - default* 412
 - single* 412
 - group-gap* §5.2.2; 411
 - group-label-align* ... §5.2.3; 415, 445
 - groupgap* *see* *group-gap*
 - horizontal* §5.2.2; 409
 - include-if-fn* §5.2.1; 400, 409, 424
 - include-if* §5.2.1; 400, 409, 423, 424
 - init* §5.2; 406, 407, 432
 - label-offset* .. §5.2.3; 415, 443
 - length* §5.2.2; 411, 442
 - lower-label-style* ... §5.2.2; 409, 416, 424, 444
 - above* 410
 - below* 410
 - opposite* 410
 - same* 410
 - max-depth* .. §5.2.4; 418, 419, 442
 - max* §5.2.4; 418, 419, 442
 - maxdepth* *see* *max-depth*
 - multi-bar-labels* §5.2.3; 414, 415, 427
 - multibarlabels* *see* *multi-bar-labels*

- negative-bar-colors §5.2.2; 413, 435
- negative-color-style §5.2.2; 413, 720
 - cycle 413
 - default 414
 - single 413
- outline-color §5.2.2; 414, 435, 446
- outline-width §5.2.2; 412, 414, 435, 446
- pre-init §5.2; 407, 408, 432
- round ... §5.2.4; 417, 420, 424, 442, 443, 692
- upper-bar-label §5.2.3; 414, 415
- upper-label-align ... §5.2.2; 410, 424, 427, 444, 445, 691
- upper-label-offset . §5.2.3; 416, 427
- upper-multi-bar-labels §5.2.3; 415, 427
- upperbarlabel see upper-bar-label
- uppermultibarlabels ... see upper-multi-bar-labels
- variable ... §5.2.1; 400, 401, 408, 443, 690
- variables . §5.2.1; 400, 401, 408, 439, 751
- vertical §5.2.2; 409
- verticalbars §5.2.2; 399, 409, 441
- x-axis-style 442
- y-axis-style 442
- y-axis §5.2.4; 417, 418, 419
- y-tic-label-align see y-tick-label-align
- y-tick-gap §5.2.4; 418, 419
- y-tick-label-align . §5.2.4; 419, 420, 445
- y-tick-labels §5.2.4; 419, 445
- y-tick-points §5.2.4; 418, 419
- y-tick-round §5.2.4; 417, 442, 443
- y-ticks §5.2.4; 417, 418, 419
- yaxis see y-axis
- ylabel-position .. §5.2.4; 418
 - center 418
 - max 418
 - min 418
 - zero 418, 431
- ylabel §5.2.4; 417
- ytic-label-align see y-tick-label-align
- ytic-round see y-tick-round
- yticgap see y-tick-gap
- yticlabels see y-tick-labels
- yticpoints see y-tick-points
- ytics see y-ticks
- compare ... §2.1; 11, 69, 72, 164, 169
 - skip-cs §2.9.5.1; 73, 164, 169, 170
- datetime §2.1; 10, 11, 131, 134
 - auto-reformat §2.7; 10, 11, 132, 133, 134, 137, 358, 359
 - datetime2 134
 - false 133
 - iso 134
 - region 133
 - true 133
- parse .. §2.7; 12, 53, 132, 133, 673, 684, 685
 - auto-reformat 132
 - false 132
 - iso-only 132
 - iso+region 133
 - parse-only 132
 - region-only 133
 - true 132
- default-name . §3.1; 177, 205, 223, 224, 226, 285, 325, 339, 357, 358, 360, 364, 366, 536, 578, 588, 707
- display 243
 - after-head ... §3.7.1.3; 247, 248, 253, 269

- `align-specs` . §3.7.1.2; 246, 254, 255, 257, 260, 269, 284, 686, 687, 693, 700, 738, 760, 770
- `caption` ... §3.7.1.3; 248, 249, 258, 259, 786
- `cont-caption` §3.7.1.3; 249, 258, 259, 787
- `contcaption` *see* `cont-caption`
- `currency-align` §3.7.1.2; 246, 255
- `decimal-align` §3.7.1.2; 246, 254
- `foot` ... §3.7.1.3; 249, 258, 259, 269, 271, 790
- `header-row` ... §3.7.1.3; 247, 248, 257, 278, 687
- `init` §3.7.1.1; 243, 269
- `int-align` *see* `integer-align`
- `integer-align` ... §3.7.1.2; 245, 246, 254
- `inter-col` §3.7.1.2; 246, 247, 255
- `label` . §3.7.1.3; 249, 258, 259, 792
- `last-foot` §3.7.1.3; 250, 258, 259, 792
- `lastfoot` *see* `last-foot`
- `longtable-env` §3.7.1.1; 245, 254
- `no-header` §3.7.1.3; 248, 252, 253, 257, 259, 791
- `omit-columns` §3.7.1.4; 251, 263
- `omit-keys` §3.7.1.4; 251, 263
- `omit`  *see* `omit-keys`
- `only-columns` §3.7.1.4; 251, 252, 263
- `only-keys` §3.7.1.4; 251, 252, 263, 269, 562
- `per-row` ... §3.7.1.1; 243, 244, 256, 258, 259, 273, 275, 277, 677, 711, 788
- `post-col` . §3.7.1.2; 246, 247, 255
- `post-head` §3.7.1.3; 247, 248, 259, 269, 795
- `post-row-function` . §3.7.1.1; 244, 250, 253, 269, 283
- `post-row-inline` §3.7.1.1; 244, 250, 253
- `pre-col` §3.7.1.2; 246, 247, 255, 268
- `pre-content` .. §3.7.1.1; 242, 243
- `pre-head` §3.7.1.3; 247, 248, 253, 269
- `real-align` *see* `decimal-align`
- `row-condition-function` §3.7.1.4; 251, 263, 275
- `row-condition-inline` §3.7.1.4; 250, 251, 263, 275
- `row-idx-map-function` §3.7.1.1; 244, 258, 274, 711
- `row-idx-map-inline` §3.7.1.1; 243
- `short-caption` ... §3.7.1.3; 249, 258, 259, 797
- `shortcaption` *see* `short-caption`
- `string-align` §3.7.1.2; 245, 246, 254, 260
- `tabular-env` .. §3.7.1.1; 245, 253
- `global` . 208–210, 231, 232, 235, 236, 289, 297, 325, 334, 338, 339, 354, 373, 374, 688, 689, 721, 752, 761, 762
- `index` §8.1; 578, 579, 581, 583
 - `balance` ... §8.1.2.2; 581, 582, 583, 611, 623
 - `child-sort` §8.1.2.3; 583
 - `child` . §8.1.2.3; 580, 583, 611, 612, 622, 722
 - `childsort` ... *see* `child-sort`
 - `columns` ... §8.1.2.3; 579, 581, 582, 583, 611
 - `compositor` §8.1.2.1; 579, 581, 604
 - `condition` §8.1.2.3; 583, 584, 610, 611, 619
 - `counter` ... §8.1.2.1; 579, 581, 603

- database .. §8.1.2.3; 583, 588, 610
- heading §8.1.2.2; 582, 622
- include-if-fn ... §8.1.2.3; 584, 610, 611, 619
- include-if ... §8.1.2.3; 584, 610, 611, 619
- location-width . §8.1.2.3; 584, 614, 615
- location §8.1.2.3; 580, 584, 604, 613
- locationwidth *see* location-width
- name-case §8.1.2.3; 580, 585, 611
- name-font §8.1.2.3; 585, 611
- namecase *see* name-case
- namefont *see* name-font
- post-desc §8.1.2.3; 580, 585, 586
- post-heading §8.1.2.2; 582, 623
- post-name ... §8.1.2.3; 580, 585, 611, 612
- postdesc *see* post-desc
- postheading *see* post-heading
- postname *see* post-name
- pre-location ... §8.1.2.3; 580, 586, 613
- prelocation *see* pre-location
- see* §8.1.2.3; 580, 586
- show-groups . §8.1.2.2; 582, 610, 615, 623
- showgroups *see* show-groups
- sort ... §8.1.2.2; 578, 582, 588, 590, 610, 616–618, 623, 624
- style . §8.1.2.2; 582, 615, 623, 671, 722, 723, 726, 728
- symbol-desc . §8.1.2.3; 580, 581, 586, 587, 589, 612
- symbol-width §8.1.2.3; 587, 614, 615
- symboldesc *see* symbol-desc
- symbolwidth *see* symbol-width
- warn §8.1.2.1; 579, 581
- initial-purify §2.1; 10, 142
- io §3.1; 177, 181, 337, 346
 - add-delimiter §3.15.2; 338, 346
 - always 346
 - detect 347
 - never 346, 353
- auto-keys . §3.15.2; 347, 348, 356
- autokeys *see* auto-keys
- convert-numbers .. §3.15.2; 14, 133, 347, 349, 350, 363
- csv-blank §3.15.2; 347
 - empty-row 348
 - end 348
 - ignore 348
- csv-content . §3.15.2; Table 3.1; 14, 133, 198, 347, 348, 349, 352, 360–363, 538, 760, 798
 - literal . 187, 337, 349, 350, 352
 - no-parse 349
 - tex 187, 337, 347, 349, 352
- csv-escape-chars ... §3.15.2; 348, 349, 350
 - delim 351
 - delim+bksl 351
 - double-delim 351
 - none 349, 351
- csv-skip-lines .. §3.15.2; 202, 351, 352, 358, 359
- data-types ... §3.15.2; 347, 349, 352, 361
- delimiter §3.15.2; 177, 338, 349–351, 352, 354
- expand . §3.15.2; 179, 318, 353, 769
 - full 353
 - none 353
 - protected 353
- format §3.15.2; 195, 197, 198, 203, 337, 353, 354, 355, 365
 - csv .. 337, 346–348, 350, 352, 353, 354, 355, 356, 358
 - dbtex 352, 355
 - dbtex-2 340, 344, 354, 358
 - dbtex-3 . 182, 337, 340, 355, 358
 - dtltex 352, 354

- dtltex-2 338, 354, 357, 358
- dtltex-3 338, 354, 358
- tsv .. 181, 337, 346–348, 350, 352, 354, 355, 356, 358, 360
- headers §3.15.2; 239, 355, 358
- keys §3.15.2; 346–348, 356, 358
- load-action .. §3.15.2; 339, 356, 366, 782
- append 338, 340, 357
- create 357
- detect 357
- old-style 357
- overwrite 357
- name ... §3.15.2; 338–340, 344, 357, 360, 364, 366, 774
- no-header §3.15.2; 339, 340, 356, 358
- noheader *see* no-header
- omitlines §3.15.2; 359
- only-reformat-columns §3.15.2; 358, 359, 368
- overwrite §3.15.2; 359, 366
 - allow 359
 - error 359
 - warn 359
- separator §3.15.2; 181, 337, 349, 350, 354, 359
- trim §3.15.2; 360
- lists ... §2.1; 11, 150, 152, 157, 744
 - and §2.9.1; 151, 152, 651, 744
 - expand-cs §2.9.5.1; 73, 164, 169, 170
 - skip-empty §2.9.1; 150, 153, 355, 356
 - sort-datum . §2.9.1; 53, 151, 157, 159, 161
 - sort-reverse .. §2.9.1; 150, 157
 - trim §2.9.1; 150, 153, 180, 355, 356
- new-value-expand §3.1; 177, 178, 209, 231, 233, 291, 297, 317, 353, 714, 753
- new-value-trim .. §3.1; 180, 231, 233, 317, 338, 360
- numeric §2.1; 10, 11, 13, 16, 45, 114, 130
- auto-reformat .. §2.2.1; 10–12, 14, 16, 119–121, 358, 359, 363, 368
- currency-symbol-style §2.2.1; 15, 129, 130
- region-currency-prefix §2.2.1; 15, 45, 119
- region-currency §2.2.1; 14, 45, 114, 796
- region-number-chars 45, 796
- set-currency §2.2.1; 15
- person 629
 - global §9.2; 630
 - local §9.2; 629, 630, 633–635, 646, 653, 654, 814, 815
 - shortcuts §9.2; 630, 636, 638–643, 668, 802, 817–819, 821, 822
- pie §4.2; 378, 380
 - cutaway-offset *see* cutawayoffset
 - cutaway-ratio *see* cutawayratio
 - cutaway §4.2.2; 381
 - cutawayoffset §4.2.2; 381, 382
 - cutawayratio .. §4.2.2; 381, 382
 - include-if-fn §4.2.1; 378, 381, 386, 387
 - include-if §4.2.1; 378, 381, 386, 387
 - inner-label *see* innerlabel
 - inner-offset *see* inneroffset
 - inner-ratio *see* innerratio
 - innerlabel §4.2.3; 383, 384
 - inneroffset §4.2.3; 384
 - innerratio §4.2.3; 382, 384
 - outer-label *see* outerlabel
 - outer-offset *see* outeroffset
 - outer-ratio *see* outerratio
 - outerlabel §4.2.3; 384
 - outeroffset ... §4.2.3; 384, 385

- outerratio §4.2.3; 382, 384, 385
- outline-color §4.2.2; 382, 397
- outline-width §4.2.2; 382, 393, 394, 397
- radius §4.2.2; 382, 384
- radius* §4.2.2; 382, 384
- rotate-inner *see* rotateinner
- rotate-outer *see* rotateouter
- rotateinner ... §4.2.3; 380, 385
- rotateouter ... §4.2.3; 380, 385
- round §4.2.3; 385
- segment-colors §4.2.2; 382, 396
- segment-default-colors §4.2.2; 380, 383
- segment-default-gray §4.2.2; 380, 383
- start §4.2.2; 383
- variable ... §4.2.1; 377, 378, 381, 394, 755
- plot §6.1; 453
 - axes §6.1.4; 462, 519
 - both 462
 - none 462
 - x 462
 - y 462
 - axis-style §6.1.4; 464, 496, 524
 - bounds .. §6.1.2; 456, 461, 495, 509
 - box-ticks §6.1.4; 465, 500
 - in 465
 - match-axes 465
 - none 465
 - out 465
 - box . §6.1.4; 463, 464, 465, 499, 501, 506, 518
 - colors §6.1.3; 460
 - extend-axes §6.1.4; 465
 - extend-x-axis §6.1.4; 464, 497
 - extend-y-axis §6.1.4; 464, 497
 - extra-assign §6.1.1; 454
 - grid §6.1.4; 466, 497, 518
 - group-styles . §6.1.3; 453, 458, 459, 460, 492, 494
 - all 459
 - line-color 459, 460
 - line-style 459, 460
 - mark-color 459, 460
 - mark-style 459, 460
 - none 459
 - height §6.1.2; 455, 464, 521
 - include-if-fn §6.1.1; 452, 454, 455
 - include-if §6.1.1; 452, 454, 455
 - legend-labels §6.1.3; 453, 457, 481–483, 513, 672
 - legend-offset §6.1.3; 457, 487, 520
 - legend §6.1.3; 457, 473, 480, 487–489, 514, 704
 - legendlabels *see* legend-labels
 - line-colors ... §6.1.3; 453, 459, 460, 521
 - linecolors *see* line-colors
 - lines §6.1.3; 453, 458–460, 461, 522
 - major-grid-style §6.1.4; 466, 498, 524
 - mark-colors ... §6.1.3; 453, 459, 460, 461, 494, 522
 - markcolors *see* mark-colors
 - marks §6.1.3; 453, 458, 459, 461, 522
 - max-x-label-style ... §6.1.4; 467
 - max-x-label §6.1.4; 467
 - max-x §6.1.2; 456, 470
 - max-y-label-style ... §6.1.4; 467
 - max-y-label §6.1.4; 467
 - max-y §6.1.2; 456, 472, 495
 - maxx *see* max-x
 - maxy *see* max-y
 - min-x-label-style ... §6.1.4; 467

- min-x-label §6.1.4; 466
- min-x §6.1.2; 456, 470
- min-y-label-style ... §6.1.4; 467
- min-y-label §6.1.4; 467
- min-y §6.1.2; 456, 472, 495
- minor-grid-style §6.1.4; 466, 499, 524
- minor-ticks ... §6.1.4; 468, 496
- minorticks . *see* minor-ticks
- minx *see* min-x
- miny *see* min-y
- omit-zero-label §6.1.4; 462, 509
 - auto 463
 - both 463
 - false 463
 - x 463
 - y 463
- round-x §6.1.4; 467, 521
- round-y §6.1.4; 467, 468, 521
- round §6.1.4; 467
- side-axes §6.1.4; 463, 501, 505–507
- style-resets . §6.1.3; 453, 458, 459, 494
 - all 459
 - line-color 459
 - line-style 459
 - mark-color 459
 - mark-style 459
 - none 458
- style ... §6.1.3; 458, 460, 461, 490, 494, 518, 519
- tic-label-style *see* tick-label-style
- ticdir *see* tick-dir
- ticgap *see* tick-gap
- tick-dir ... §6.1.4; 468, 469, 496
- tick-gap §6.1.4; 469
- tick-label-offset ... §6.1.4; 468
- tick-label-style §6.1.4; 468, 503, 523
- ticks §6.1.4; 468
- tics *see* ticks
- width §6.1.2; 455, 464, 521
- x-axis-style §6.1.4; 463, 464, 524
- x-label *see* xlabel
- x-minor-ticks §6.1.4; 468, 469, 519
- x-tic-label-style *see* x-tick-label-style
- x-tick-dir §6.1.4; 468, 470, 519
- x-tick-gap §6.1.4; 461, 469, 470, 520, 750
- x-tick-label-style . §6.1.4; 469
- x-tick-labels §6.1.4; 462, 467, 470, 521, 523
- x-tick-points §6.1.4; 461, 470, 471, 520, 750
- x-ticks §6.1.4; 468, 469, 519, 521
- x ... §6.1.1; 451–453, 454, 479–481, 512–517, 672, 757, 758, 785
- xlabel §6.1.4; 466, 473
- xminorticks *see* x-minor-ticks
- xticdir *see* x-tick-dir
- xticgap *see* x-tick-gap
- xticlabels *see* x-tick-labels
- xticpoints *see* x-tick-points
- xticks *see* x-ticks
- y-axis-style .. §6.1.4; 464, 524
- y-label *see* ylabel
- y-minor-ticks §6.1.4; 468, 471, 519
- y-tic-label-style *see* y-tick-label-style
- y-tick-dir §6.1.4; 468, 471, 520
- y-tick-gap §6.1.4; 461, 469, 472, 520, 750
- y-tick-label-style . §6.1.4; 471, 504
- y-tick-labels §6.1.4; 462, 468,

- 472, 521, 523
- y-tick-points §6.1.4; 461, 472, 520, 750
- y-ticks §6.1.4; 468, 471, 520, 521
- y §6.1.1; 451–453, 454, 479–481, 487, 512–516, 524, 672, 757, 758, 785
- ylabel §6.1.4; 466, 473
- yminortics *see* y-minor-ticks
- yticdir *see* y-tick-dir
- yticgap *see* y-tick-gap
- yticlabels *see* y-tick-labels
- yticpoints *see* y-tick-points
- ytics *see* y-ticks
- store-datum .. §3.1; 182, 183, 184, 188, 191, 194, 195, 197, 198, 200, 202, 217, 231, 233, 238, 270, 271, 281, 288, 291, 297, 317, 321–324, 347, 349, 361, 362, 366, 368, 370, 373, 386, 455, 562
- verbose .. §2.1; 9, 327, 341, 478, 588, 597, 598
- \DTLsort §3.14.2; 335, 526, 538, 543, 767
- \dtlsort ... §3.14.2; 162, 325, 334, 335, 336, 578, 582, 617, 767
- \DTLsortdata .. §3.14.1; 162, 164, 223, 325, 326–328, 334, 335, 526, 538, 543, 560, 578, 582, 616–618, 624, 689, 767
- encap .. §3.14.1.1; 326, 539, 560, 697
- function §3.14.1.1; 164, 325
- missing-column-action §3.14.1.1; 326
- replace §3.14.1.1; 326, 328, 330
- save-group-column .. §3.14.1.1; 326
- save-group-key .. §3.14.1.1; 326, 327, 582, 610, 616
- save-group §3.14.1.1; 327
- save-sort-column §3.14.1.1; 327
- save-sort-key §3.14.1.1; 327
- save-sort §3.14.1.1; 327
- \DTLsortdata column criteria options
 - asc §3.14.1.2; 327
 - ascending §3.14.1.2; 327, 333
 - desc §3.14.1.2; 328, 333
 - descending §3.14.1.2; 327, 328, 333
 - replacements .. §3.14.1.2; 326, 328, 329, 616
- \DTLsortedactual .. §2.9.5; 159, 767
- \DTLsortedletter .. §2.9.5; 159, 767
- \DTLsortedvalue ... §2.9.5; 159, 767
- \DTLsortlettercasehandler §2.9.5.2; 160, 165, 767
- \DTLsortletterhandler .. §2.9.5.2; 160, 163, 165, 171, 596, 768
- \dtlsortlist ... §2.9.5; 150, 155, 156, 157, 159, 162, 165, 167, 169, 173, 175, 744, 768, 774
- \DTLsortwordcasehandler §2.9.5.2; 160, 163, 165, 768
- \dtlSortWordCommands ... §2.9.5.3; 168, 538, 768
- \DTLsortwordhandler §2.9.5.2; 160, 163, 164, 165, 173, 325, 768
- \DTLsortwordlist .. §2.9.5; 51, 150, 151, 156, 157, 159, 162–167, 171, 173, 174, 325, 666, 673, 680, 684, 689, 767, 768, 772
- \dtlspecialvalue .. §3.11; 317, 318, 353, 604, 769
- \DTLsplitstring §2.8.1; 139, 769, 775
- \DTLsqrt §2.5.2; 111, 729, 769
- \dtlsqrt §2.5.1; 99, 101, 111, 769
- \DTLstartpt §5.4.4; 436, 448, 769
- \DTLStoreInitialGetLetter §2.8.2; 141, 143, 144, 769
- \DTLstoreinitials §2.8.2; 140, 141, 142, 544, 687, 688, 693, 716, 737, 738, 769, *see also*
 - \DTLstoreinitials, \DTLStoreInitialGetLetter,
 - \DTLbetweeninitials,
 - \DTLafterinitials,
 - \DTLinitialhyphen,

- \DTLafterinitialbeforehyphen, \DTLinitialpunc & \DTLaposinitialpunc
- \dtlstringalign ... §3.7.2; 245, 254, 255, 257, 770
- \dtlstringformat . §3.7.2; 252, 255, 314, 770
- \DTLstringnull ... §3.10.2; 213, 316, 317, 770
- \DTLstringtype . §3.6; 239, 668, 770
- \DTLsub §2.5.2; 110, 729, 770
- \dtlsub §2.5.1; 99, 110, 770
- \DTLsubstitute §2.8.1; 138, 770
- \DTLsubstituteall §2.8.1; 139, 771
- \DTLsumcolumn §3.13; 322, 771
- \DTLsumforkeys §3.13; 321, 322–324, 771
- \dtlswapentriesincurrentrow §3.16.1; 374, 771
- \dtlswaprows §3.16; 371, 771
- \DTL<tag>LocaleHook 54, 60, 771
- \DTLtemporalvalue . §2.2.4; 21, 771
- \dtltexorsort §2.9.5.3; 124, 166, 772
- DTLthebibliography environment ... 696, 823
- \DTLtherow §3.8.2; 295, 772
- \DTLticklabeloffset . §6.4.1; 468, 521, 772
- \DTLticklength §6.4.1; 521, 772
- \dtltimegroup 58, 772
- \DTLtotalbargroups §5.4.4; 450, 772
- \DTLtotalbars §5.4.4; 449, 772
- \DTLtrunc §2.5.2; 112, 729, 772
- \dtltrunc §2.5.1; 100, 112, 773
- \DTLtwoand §7.7.1; 546, 773
- \dtltype §3.8.2; 297, 773
- \DTLunsettype §3.6; 239, 773
- \dtlupdateentryincurrentrow §3.16.1; 320, 374, 773
- \DTLuse ... §3.3; 206, 208, 209, 211, 213, 214, 228, 773
- \DTLusedatum . §2.2.3; 17, 19, 754, 773
- \DTLvarianceforall ... §2.5.2; 114, 730, 774
- \dtlvarianceforall §2.5.1; 101, 774
- \DTLvarianceforcolumn ... §3.13; 322, 774
- \DTLvarianceforkeys §3.13; 322, 774
- \dtlwordindexcompare ... §2.9.5.1; 163, 165, 166, 169, 173, 175, 617, 673, 680, 684, 768, 774, *see also* \dtlwordindexcompare
- \DTLwrite . §3.15.4; 176, 177, 181, 318, 336–338, 340, 342, 343, 346, 351, 353–355, 357–359, 364, 538, 769, 774
- \DTLXAxisStyle §6.4.3; 524, 774
- \DTLxparse §2.2.3; 16, 17, 62, 706, 773, 775
- \DTLxsetcurrencydatum ... §2.2.3; 19, 775
- \DTLxsetdecimaldatum . §2.2.3; 18, 20, 775
- \DTLxsetintegerdatum §2.2.3; 18, 775
- \DTLxsetstringdatum §2.2.3; 19, 775
- \DTLxsplitstring . §2.8.1; 139, 775
- \DTLYAxisStyle §6.4.3; 524, 775
- \DTMdate 550
- \DTMnow 339

E

- \eappto 269, 280, 305, 306
- \edef 84
- \editionname 776
- \editorname 545, 776
- \editorsname 545, 776
- \edtlgetrowforvalue ... 373, 776
- encoding 31, 665, 666
- \endfirsthead 249
- \endfoot 249
- \endhead 249
- \endlastfoot 250
- \enspace 586
- \etalname 545, 716, 776

etoolbox package 150, 155, 159, 265, 267, 268,
280, 283, 305, 311, 316, 386, 455
expandable *see* expansion
\expandonce 269, 280
expansion .. 15–17, 20, 44, 69, 84, 86, 87, 89,
90, 94, 98, 127, 130, 138, 139, 141,
142, 144, 149, 152, 156, 159, 160,
166–168, 170, 665, 666, 675, 714,
733, 769–772
\ExplSyntaxOff §1.2; 4, 5–7, 26,
54, 60
\ExplSyntaxOn ... §1.2; 4, 5, 6, 26, 54,
146, 343

F

\f (form feed) Table 3.1; 350
\fboxrule 514
\fcolorbox 514
\femalelabels ☹ §9.4; 628, 634, 776
\field §8.6; 604, 776
file formats
 aux 527, 546, 552, 603
 bbl 526, 528, 537
 bib 526–528, 536
 bst 527
 csv 337, 354
 dbtex ... 337, 340, 353–355, 357, 360
 dltex 337, 338, 354, 358
 ldf 27
 tex 336
 tsv 337, 354
\FirstId §8.9.1; 620, 624, 777
\forallpeople .. §9.7.2; 657, 777, *see*
 also \foreachpersonbreak
\forcsvlist 150, 159
\foreachperson ... §9.7.2; 628, 657,
 659, 777, *see also*
 \forallpeople &
 \foreachpersonbreak
\foreachpersonbreak §9.7.2;
 656, 777
formatted number .. 15, 16, 19, 26, 42, 45, 46,
 62, 63, 81, 82, 87, 88, 91, 92, 97, 108,

 110–116, 128, 164, 218, 228, 230,
 321–324, 349, 368, 424, 443, 445,
 462, 523, 666, 686, 692, 698, 699,
 708, 712, 731–736, 746–751, 753,
 763, 764, 769–772, 774
fp package 3, 9, 21, 62, 104, 827
\fp_to_decimal:n 7

G

\gDTLforeachbibentry .. §7.8; 549,
 551, 777
\gDTLformatbibentry §7.8;
 551, 777
\getpersonforenames §9.5.1;
 645, 777
\getpersonfullname §9.5.1;
 645, 778
\getpersongender §9.5.1; 644,
 659, 778
\getpersongenderlabel ... §9.5.1;
 644, 778
\getpersonname §9.5.1; 644, 778
\getpersonsurname §9.5.1; 645, 778
\getpersontitle ... §9.5.1; 645, 778
glossaries–extra package 577, 588, 603, 606
glossaries package ... 2, 577, 602, 603, 608
\Gls §8.5.1; 602, 778
\gls §8.5.1; 589, 602, 605, 607, 608,
 623, 778
\glsadd ... §8.5; 597, 598, 601, 602, 779
\glsaddall §8.5; 602, 779
\Glsdispenentry §8.5; 601, 779
\glsdispenentry §8.5; 601, 779
\glslink §8.5; 600, 779
\Glsnl §8.5.1; 603, 779
\glsnl §8.5.1; 602, 779
\Glspl §8.5.1; 603, 779
\glspl ... §8.5.1; 589, 602, 605, 623, 780
\Glsplnl §8.5.1; 603, 780
\glsplnl §8.5.1; 602, 780
\glsreset §8.7.2; 609, 780
\glsresetall §8.7.2; 609, 780
\Glsym §8.5.1; 603, 780

- `\glssym` §8.5.1; 589, 603, 623, 780
`\glunset` §8.7.2; 609, 780
`\glunsetall` §8.7.2; 609, 781
`\g_person_female_label_`
`clist` §9.4; 634, 776, 781
`\g_person_male_label_clist`
§9.4; 633, 781, 800
- ## H
- `\hfil` 252
`\hfill` 586
`\HierSort` §8.9.1; 620, 624, 781
`\hline` 305, 306
`\hyperlink` 147, 618
`hyperref` package 295, 547–549, 565, 600, 603,
624, 626
`\hypertarget` 147, 618
- ## I
- `\ifdefempty` 311
`\ifdefstring` 23, 386, 455
`\ifDTLbarxaxis` 781
`\ifDTLbaryaxis` 781
`\ifDTLbarytics` 781
`\ifDTLbox` §6.4; 518, 782
`\ifDTLgrid` §6.4; 518, 782
`\ifDTLnewdbonload`  §3.15.2;
357, 782
`\ifDTLshowlines` §6.4; 518, 519, 782
`\ifDTLshowmarkers` .. §6.4; 519, 782
`\ifDTLverticalbars` .. §5.4.1; 427,
441, 444, 445, 782
`\ifDTLxaxis` §6.4; 519, 782
`\ifDTLxminortics` .. §6.4; 519, 782
`\ifDTLxtics` §6.4; 519, 783
`\ifDTLxticsin` §6.4; 519, 783
`\ifDTLyaxis` §6.4; 519, 783
`\ifDTLyminortics` .. §6.4; 519, 783
`\ifDTLytics` §6.4; 520, 783
`\ifDTLyticsin` §6.4; 520, 783
`\ifentryused` §8.9.1; 619, 783
`\iffemalelabel`  §9.4; 635, 783
`\ifmalelabel`  §9.4; 635, 784
- ## J
- `\ifnum` 83, 84, 111–113
`\ifpersonexists` .. §9.7.1; 654, 784
`\iftermexists` §8.9.1; 618, 784
`ifthen` package 62, 89–93, 542, 715
`\ifthenelse` .. 62, 89, 94, 95, 296, 306,
322, 324, 378, 400, 452, 540, 552,
583, 694, 695, 738–743
`\input` 337–339
`inputenc` package 31, 173
`\inputencoding` 55
`\InputIfFileExists` 337
`\item` 552, 704
- ## J
- `JD (Julian Date)` 13, 666
`JDN (Julian Day Number)` 13, 666, 666
`JF (Julian Time Fraction)` 13, 666, 666
`\jobname` 536, 745
- ## L
- `l3keys` package 242
`\Label` .. §8.9.1; 619, 620, 622, 623, 784
`\label` 147, 249, 294, 300
`\LaTeX` 168
`\l_dataplot_legend_names_`
`prop` §6.3.3; 516, 784
`\l_dataplot_legend_name_tl`
§6.3.3; 516, 784
`\l_dataplot_legend_tl` .. §6.3.2;
513, 514, 515, 672, 784
`\l_dataplot_legend_xlabel_`
`prop` §6.3.3; 517, 785
`\l_dataplot_legend_xlabel_`
`tl` §6.3.3; 517, 785
`\l_dataplot_legend_ylabel_`
`prop` §6.3.3; 518, 785
`\l_dataplot_legend_ylabel_`
`tl` §6.3.3; 518, 785
`\l_dataplot_stream_index_`
`int` §6.3.2; 513, 516, 785
`\l_dataplot_x_key_int` .. §6.3.2;
513, 514, 785
- `\glssym` 856
`\l_dataplot_x_key_int`

- \l_dataplot_y_key_int ... §6.3.2; 513, 514, 785
- \l_datatool_austral_str §2.3.1; 38, 785
- \l_datatool_austral_tl . §2.3.1; 38, 786
- \l_datatool_baht_str §2.3.1; 33, 786
- \l_datatool_baht_tl §2.3.1; 33, 786
- \l_datatool_bitcoin_str §2.3.1; 41, 786
- \l_datatool_bitcoin_tl . §2.3.1; 41, 786
- \l_datatool_caption_tl . §3.7.2; 259, 786
- \l_datatool_cedi_str §2.3.1; 39, 786
- \l_datatool_cedi_tl §2.3.1; 39, 787
- \l_datatool_cent_str §2.3.1; 32, 787
- \l_datatool_cent_tl §2.3.1; 32, 787
- \l_datatool_colonsign_str §2.3.1; 34, 787
- \l_datatool_colonsign_tl §2.3.1; 34, 787
- \l_datatool_cont_caption_tl §3.7.2; 259, 787
- \l_datatool_cruzerio_str §2.3.1; 34, 787
- \l_datatool_cruzerio_tl §2.3.1; 34, 788
- \l_datatool_currencysigns_regex §2.3.1; 42, 788
- \l_datatool_currency_str §2.3.1; 32, 788
- \l_datatool_currency_tl §2.3.1; 32, 788
- \l_datatool_current_language_tl §2.3.5; 29, 47, 59, 60, 788
- \l_datatool_current_region_tl 28, 788
- \l_datatool_display_per_row_int §3.7.2; 256, 259, 788
- \l_datatool_display_tab_rows_int §3.7.2; 259, 788
- \l_datatool_dong_str §2.3.1; 36, 789
- \l_datatool_dong_tl §2.3.1; 36, 789
- \l_datatool_drachma_str §2.3.1; 37, 789
- \l_datatool_drachma_tl . §2.3.1; 37, 789
- \l_datatool_ecu_str §2.3.1; 33, 789
- \l_datatool_ecu_tl §2.3.1; 33, 789
- \l_datatool_euro_str §2.3.1; 36, 789
- \l_datatool_euro_tl §2.3.1; 36, 790
- \l_datatool_florin_str . §2.3.1; 33, 790
- \l_datatool_florin_tl ... §2.3.1; 33, 790
- \l_datatool_foot_tl §3.7.2; 259, 790
- \l_datatool_frenchfranc_str §2.3.1; 34, 790
- \l_datatool_frenchfranc_tl §2.3.1; 34, 790
- \l_datatool_germanpenny_str §2.3.1; 37, 790
- \l_datatool_germanpenny_tl §2.3.1; 37, 791
- \l_datatool_guarani_str §2.3.1; 38, 791
- \l_datatool_guarani_tl . §2.3.1; 38, 791
- \l_datatool_hryvnia_str §2.3.1; 38, 791
- \l_datatool_hryvnia_tl . §2.3.1; 38, 791
- \l_datatool_include_header_bool §3.7.2; 259, 791
- \l_dataplot_y_key_int 857
- \l_datatool_include_header_bool

<code>\l_datatool_indianrupee_str</code>	§2.3.1 ; 40, 791	<code>\l_datatool_peseta_str</code>	§2.3.1 ; 35, 795
<code>\l_datatool_indianrupee_tl</code>	§2.3.1 ; 40, 792	<code>\l_datatool_peseta_tl</code>	§2.3.1 ; 35, 795
<code>\l_datatool_kip_str</code>	§2.3.1 ; 37, 792	<code>\l_datatool_peso_str</code>	§2.3.1 ; 38, 795
<code>\l_datatool_kip_tl</code>	§2.3.1 ; 37, 792	<code>\l_datatool_peso_tl</code>	§2.3.1 ; 38, 795
<code>\l_datatool_label_tl</code>	§3.7.2 ; 259, 792	<code>\l_datatool_post_head_tl</code>	§3.7.2 ; 247, 259, 795
<code>\l_datatool_lari_str</code>	§2.3.1 ; 41, 792	<code>\l_datatool_pound_str</code>	§2.3.1 ; 32, 168, 795
<code>\l_datatool_lari_tl</code>	§2.3.1 ; 41, 792	<code>\l_datatool_pound_tl</code>	§2.3.1 ; 31, 32, 796
<code>\l_datatool_last_foot_tl</code>	§3.7.2 ; 259, 792	<code>\l_datatool_region_set_cur-</code> <code>rency_bool</code>	§2.3.2 ; 45, 681, 796
<code>\l_datatool_lira_str</code>	§2.3.1 ; 34, 793	<code>\l_datatool_region_set_num-</code> <code>berchars_bool</code>	§2.3.2 ; 45, 54, 681, 796
<code>\l_datatool_lira_tl</code>	§2.3.1 ; 34, 793	<code>\l_datatool_ruble_str</code>	§2.3.1 ; 41, 796
<code>\l_datatool_livretournois_</code> <code>str</code>	§2.3.1 ; 39, 793	<code>\l_datatool_ruble_tl</code>	§2.3.1 ; 41, 796
<code>\l_datatool_livretournois_</code> <code>tl</code>	§2.3.1 ; 39, 793	<code>\l_datatool_rupee_str</code>	§2.3.1 ; 35, 796
<code>\l_datatool_manat_str</code>	§2.3.1 ; 40, 793	<code>\l_datatool_rupee_tl</code>	§2.3.1 ; 35, 796
<code>\l_datatool_manat_tl</code>	§2.3.1 ; 40, 793	<code>\l_datatool_shekel_str</code>	§2.3.1 ; 36, 797
<code>\l_datatool_measure_hook_tl</code>	§2.8.3 ; 147, 679, 793	<code>\l_datatool_shekel_tl</code>	§2.3.1 ; 36, 797
<code>\l_datatool_middot_str</code>	§2.3.1 ; 33, 794	<code>\l_datatool_short_caption_</code> <code>tl</code>	§3.7.2 ; 259, 797
<code>\l_datatool_middot_tl</code>	§2.3.1 ; 33, 794	<code>\l_datatool_som_str</code>	§2.3.1 ; 41, 797
<code>\l_datatool_mill_str</code>	§2.3.1 ; 35, 794	<code>\l_datatool_som_tl</code>	§2.3.1 ; 41, 797
<code>\l_datatool_mill_tl</code>	§2.3.1 ; 35, 794	<code>\l_datatool_spesmilo_str</code>	§2.3.1 ; 39, 797
<code>\l_datatool_naira_str</code>	§2.3.1 ; 35, 794	<code>\l_datatool_spesmilo_tl</code>	§2.3.1 ; 39, 797
<code>\l_datatool_naira_tl</code>	§2.3.1 ; 35, 794	<code>\l_datatool_str_csv_regex_</code> <code>cases_tl</code>	§3.15.2 ; 350, 798
<code>\l_datatool_nordicmark_str</code>	§2.3.1 ; 40, 794	<code>\l_datatool_tenge_str</code>	§2.3.1 ;
<code>\l_datatool_nordicmark_tl</code>			
<code>\l_datatool_indianrupee_str</code>	858	<code>\l_datatool_tenge_str</code>	

- 39, 798
`\l_datatool_tenge_tl` ... §2.3.1; 39, 798
`\l_datatool_tugrik_str` ... §2.3.1; 37, 798
`\l_datatool_tugrik_tl` ... §2.3.1; 37, 798
`\l_datatool_turkishlira_str` ... §2.3.1; 40, 798
`\l_datatool_turkishlira_tl` ... §2.3.1; 40, 798
`\l_datatool_won_str` ... §2.3.1; 36, 799
`\l_datatool_won_tl` §2.3.1; 36, 799
`\l_datatool_yen_str` ... §2.3.1; 32, 799
`\l_datatool_yen_tl` §2.3.1; 32, 799
`\loadgidx` ... §8.3; 584, 588, 799
`\Location` ... §8.9.1; 620, 624, 799
`\Long` ... §8.9.1; 619, 623, 799
`\LongPlural` ... §8.9.1; 619, 623, 799
longtable environment ... 223, 242, 243, 245, 246, 248–250, 253, 256, 258, 275, 303, 677, 710
longtable package ... 242
longtabu environment ... 245, 254
lowercase ... 47, 71, 148, 163, 164, 169, 731
`\l_person_label_tl` ... §9.7.4; 660, 800
- ## M
- `\MakeUppercase` ... 530
`\malelabels` ☹ ... §9.4; 628, 633, 800
`\meaning` ... 84
mfirstuc package ... 585, 600, 606
`\midrule` ... 248, 269, 301
`\mscthesisname` ... 800
multicol package ... 170, 581
multicols environment ... 581–583, 623
`\multicolumn` ... 247
- ## N
- `\n` (newline) ... Table 3.1; 350
`\Name` ... §8.9.1; 611, 619, 623, 800
`\newacro` ... §8.7; 590, 604, 606, 667, 800
`\newgidx` ... §8.2; 578, 581, 583, 584, 587, 588, 604, 609, 622, 623, 800
`\newperson` ... §9.3; 629, 630, 633–635, 648, 653, 659–661, 800, 801, 811
`\newperson*` ... §9.3; 630, 633, 634, 661, 801
expand-forenames ... §9.3; 631
expand-fullname ... §9.3; 631
expand-name ... §9.3; 631
expand-once-forenames ... §9.3; 631
expand-once-fullname ... §9.3; 631
expand-once-name ... §9.3; 631
expand-once-surname ... §9.3; 632
expand-once-title ... §9.3; 632
expand-surname ... §9.3; 632
expand-title ... §9.3; 632
forenames ... §9.3; 631, 644
fullname ... §9.3; 630, 631, 643, 653
gender ... §9.3; 630, 632, 633, 634, 647, 653
name ... §9.3; 630, 631, 644, 653
surname ... §9.3; 631, 632, 644, 653
title ... §9.3; 631, 632, 653
`\newterm` ... §8.4; 588, 590, 597–599, 601, 604–606, 620, 621, 669–671, 724, 725, 800, 801, 816
database ... §8.4; 588, 589
description ... §8.4; 589
label ... §8.4; 588, 589, 591–594, 596, 597, 621
long ... §8.4; 590, 606
longplural ... §8.4; 590, 606
parent ... §8.4; 589
plural ... §8.4; 589, 606
see ... §8.4; 589
seealso ... §8.4; 590
short ... §8.4; 590, 606
shortplural ... §8.4; 590, 606
sort ... §8.4; 588, 590, 591–594, 596, 598, 621, 624
`\newterm`

- symbol §8.4; 589
 text §8.4; 589, 591, 605, 606
 \newtermaddfield ... §8.6; 604, 620, 776, 801
 \newtermlabelhook §8.4.2; 597, 801
 \newtermsorthook §8.4.3; 598, 599, 801
 \noalign, Misplaced error . 293, 305, 308
 \nobreakspace 70, 140, 168
 \nocite 527, 529, 536, 550, 553, 560, 562, 567, 753
 \noexpand 280
 \not 89
 \num 14, 19, 20, 26, 764
 \number 19
 number group character 11, 12, 15, 16, 28, 42, 43, 45, 53, 62–65, 97, 108, 115, 116, 118, 120, 121, 128, 130, 368, 369, 666, 681–683, 766
 \numbername 801
 \numexpr 108, 110, 111
- O**
- \onecolumn 611
 \or 89
 Oxford comma 152
- P**
- \pagename §7.7.1; 546, 759, 802
 \pageref 147
 \pagesname §7.7.1; 546, 759, 802
 \par 235, 549, 754
 \Parent . §8.9.1; 610–612, 620, 624, 802
 \Parents .. §9.5.0.1; Table 9.1; 643, 802
 \parents .. §9.5.0.1; Table 9.1; 643, 802
 \path 435, 436, 442, 691
 \Peoplechild . §9.5.0.2; Table 9.1; 642, 668, 802
 \peoplechild . §9.5.0.2; Table 9.1; 642, 668, 802
 \peopleforenames §9.5.2; 644, 646, 802
 \peoplefullname §9.5.2; 643, 646, 803
 \peoplename §9.5.2; 644, 646, 651, 803
 \Peopleobjpronoun §9.5.0.1; Table 9.1; 639, 803, 819
 \peopleobjpronoun §9.5.0.1; Table 9.1; 639, 803, 819
 \Peopleobjpronounii ... §9.5.0.1; Table 9.1; 639, 803, 818
 \peopleobjpronounii ... §9.5.0.1; Table 9.1; 639, 803, 818
 \Peopleparent §9.5.0.2; Table 9.1; 643, 802, 803
 \peopleparent §9.5.0.2; Table 9.1; 643, 802, 803
 \Peoplepossadj .. §9.5.0.1; Table 9.1; 640, 804, 818
 \peoplepossadj .. §9.5.0.1; Table 9.1; 640, 804, 818
 \Peoplepossadjii §9.5.0.1; Table 9.1; 641, 804, 821
 \peoplepossadjii §9.5.0.1; Table 9.1; 640, 641, 804, 821
 \Peopleposspronoun §9.5.0.1; Table 9.1; 641, 804, 818
 \peopleposspronoun §9.5.0.1; Table 9.1; 641, 804, 819
 \Peopleposspronounii ... §9.5.0.1; Table 9.1; 642, 804, 821
 \peopleposspronounii ... §9.5.0.1; Table 9.1; 642, 804, 822
 \Peoplepronoun .. §9.5.0.1; Table 9.1; 638, 649, 660, 805, 819
 \peoplepronoun .. §9.5.0.1; Table 9.1; 638, 660, 805, 819
 \Peoplepronounii §9.5.0.1; Table 9.1; 638, 805, 821
 \peoplepronounii §9.5.0.1; Table 9.1; 638, 805, 821
 \Peoplesibling .. §9.5.0.2; Table 9.1; 643, 805, 817
 \peoplesibling .. §9.5.0.2; Table 9.1; 643, 805, 817
 \peoplesurname §9.5.2; 644, 646, 805

- \peopletitlesurname . §9.5.2; 644, 646, 805
- person-english.ldf §9.7.3; 54, 628, 657
- person package §9; 628, 827
 - base-only . §9.1; 628, 629, 648, 827
 - datatool §9.1; 629, 827
 - shortcuts . §9.1; 629, 630, 647, 649, 651, 827
- \PersonAddFemaleLabel . 634, 667, 776, 806
- \PersonAddMaleLabel 633, 668, 800, 806
- \PersonAddNonBinaryLabel . 806
- \person_all_gender_
 - case:nnnn §9.7.1; 656, 806
- \Personchild §9.5.0.2; 642, 806
- \personchild §9.5.0.2; 642, 806
- \person_do_if_valid_
 - gender:nT §9.7.1; 656, 806
- \PersonFemaleCount §9.4; 635, 654, 656, 807, 808
- \personforenames §9.5.1; 644, 646, 807
- \personfullname ... §9.5.1; 643, 646, 647, 807
- \Persongender 659, 807
- \persongender .. §9.5.1; 644, 659, 807
- \person_gender_case:Nnnnn
 - §9.7.1; 656, 807
- \person_gender_case:nnnnn
 - §9.7.1; 656, 807
- \person_gender_case:Nnnnnn
 - §9.7.1; 655, 656, 807, 808
- \person_gender_case:nnnnnn
 - §9.7.1; 656, 808
- \person_get_attribute:nn §9.7; 653, 808
- \person_get_attribute:Nnn
 - §9.7; 654, 808
- \PersonIfAllFemale §9.7.1; 654, 808
- \PersonIfAllMale .. §9.7.1; 654, 808
- \PersonIfAllNonBinary ... §9.7.1; 655, 809
- \PersonIfAllUnknownGender
 - §9.7.1; 655, 809
- \person_if_exist:nTF §9.7.1; 655, 809
- \PersonIfFemale ... §9.7.1; 654, 809
- \PersonIfFemaleLabel .. §9.4; 635, 783, 809, *see also*
 - \PersonAddFemaleLabel &
 - \PersonSetFemaleLabels
- \PersonIfMale §9.7.1; 654, 809
- \PersonIfMaleLabel §9.4; 635, 784, 809, *see also*
 - \PersonAddMaleLabel &
 - \PersonSetMaleLabels
- \PersonIfNonBinary §9.7.1; 655, 810
- \PersonIfNonBinaryLabel .. §9.4; 635, 810, *see also* \PersonAddNonBinaryLabel &
 - \PersonSetNonBinaryLabels
- \PersonIfUnknownGender . §9.7.1; 655, 810
- \person_Language_all_text:n
 - §9.7.3; 660, 810
- \person_language_all_text:n
 - §9.7.3; 660, 810
- \person_Language_text:nn
 - §9.7.3; 659, 810
- \person_language_text:nn
 - §9.7.3; 659, 810
- \personlastsep §9.5.2; 645, 811
- \PersonMaleCount ... §9.4; 635, 654, 656, 808, 811
- \personname §9.5.1; 630, 644, 646, 811
- \person_new_appto_end:n §9.7.4; 661, 811
- \person_new_appto_start:n
 - §9.7.4; 660, 811
- \PersonNonBinaryCount §9.4; 635, 655, 656, 809, 811
- \Personobjpronoun §9.5.0.1; 639, 811

- `\personobjpronoun` ... §9.5.0.1; 638, 639, 811
- `\Personobjpronounii` §9.5.0.1; 639, 812
- `\personobjpronounii` §9.5.0.1; 639, 812
- `\Personparent` §9.5.0.2; 642, 812
- `\personparent` §9.5.0.2; 642, 643, 812
- `\Personpossadj` ... §9.5.0.1; 640, 812
- `\personpossadj` ... §9.5.0.1; 640, 812
- `\Personpossadjii` §9.5.0.1; 640, 812
- `\personpossadjii` §9.5.0.1; 640, 812
- `\Personposspronoun` §9.5.0.1; 641, 813
- `\personposspronoun` §9.5.0.1; 641, 813
- `\Personposspronounii` ... §9.5.0.1; 641, 813
- `\personposspronounii` ... §9.5.0.1; 641, 642, 813
- `\Personpronoun` ... §9.5.0.1; 636, 649, 659, 813
- `\personpronoun` ... §9.5.0.1; 636, 638, 659, 813
- `\Personpronounii` §9.5.0.1; 638, 813
- `\personpronounii` §9.5.0.1; 638, 813
- `\person_remove_appto:n` . §9.7.4; 654, 661, 814
- `\personsep` §9.5.2; 645, 814
- `\person_set_attribute:nnn` §9.7; 653, 661, 814
- `\PersonSetFemaleLabels` ... §9.4; 628, 634, 659, 776, 814
- `\PersonSetLocalisation` . §9.7.3; 636, 657, 814
- `\PersonSetMaleLabels` .. §9.4; 628, 633, 634, 659, 800, 814
- `\PersonSetNonBinaryLabels` §9.4; 634, 659, 814
- `\Personsibling` ... §9.5.0.2; 643, 814
- `\personsibling` §9.5.0.2; 643, 814, 815
- `\personsurname` §9.5.1; 644, 646, 815
- `\persontitlesurname` . §9.5.1; 644, 646, 815
- `\persontitlesurnamesep` . §9.5.1; 631, 644, 815
- `\PersonTotalCount` §9.3; 632, 654–656, 808, 809, 815
- `\PersonUnknownGenderCount` §9.4; 636, 655, 809, 815
- `\person_unset_attribute:nn` §9.7; 654, 661, 815
- pgf package . 3, 444, 445, 448, 511, 514, 525, 713, 748, 756, 758, 769
- pgfmath package 3, 9, 107, 827
- pgfpicture environment 512
- `\pgfplothandlermark` 524, 756
- `\pgfpointxy` 448, 449
- `\pgfsetdash` 458, 461, 522
- `\pgftext` .. 415, 419, 431, 444, 445, 690, 692, 693
- `\pgftransformreset` 511
- `\pgfuseplotmark` 458, 461, 523
- plain number 2, 11, 12, 15, 17–19, 21, 42, 45, 46, 62, 83, 97–101, 108, 110–115, 118, 121, 146, 200, 203, 204, 218, 228, 230, 282, 321, 324, 345, 347–350, 360, 362, 395, 397, 418, 445, 461, 470, 472, 523, 666, 680, 686, 687, 698, 699, 706, 708, 712, 733–736, 747–749, 751, 754, 762–764, 769, 770, 773, 774
- `\Plural` §8.9.1; 620, 623, 815
- polyglossia package 27, 60
- `\postnewtermhook` .. §8.9.2; 620, 816
- `\pounds` 12, 19, 31, 66, 123, 124, 126, 130, 160–162, 168
- `\preto` 155
- `\printnoidxglossary` 577
- `\printterms` . §8.8; 578–584, 587–595, 600, 601, 603, 609, 610, 611, 615, 616, 618–623, 668–671, 685, 722, 723, 777, 781, 784, 799, 800, 802, 815, 816, 817–820
- `\printtermsrestoreonecolumn` §8.8; 611, 816
- `\printunsrtglossary` 577
- `\personobjpronoun` 862
- `\printunsrtglossary`

probsoln package 176
 \protect 94
 purify 10, 70, 142, 148, 164, 165, 666
 SQL b, 176, 666
 \stepcounter 147, 304, 306
 substr package 4, 69
 \Symbol §8.9.1; 619, 623, 817

R

\r (carriage return) Table 3.1; 350
 \ref 147
 \refstepcounter 147, 267, 294, 300, 304
 \regex_replace_all:nnN 5
 \regex_replace_case_all:nN 350
 \relax 238, 458, 460, 461, 512, 522, 523
 relsize package 15
 \removeallpeople §9.3; 633, 661, 816
 \removepeople §9.3; 633, 661, 814, 816
 \removeperson §9.3; 632, 661, 814, 816
 robust 62, 81–83, 86, 87, *see also* expansion
 \rowcolor 276, 277, 281, 282, 298, 308

S

\See §8.9.1; 613, 620, 624, 816
 \SeeAlso §8.9.1; 620, 624, 817
 \seelisoname §8.8.1; 614, 817
 \seename §8.8.1; 614
 \setcounter 442, 521
 \setkeys 242
 \setlength 393, 442, 443, 514, 520
 \settodepth 146, 679
 \settoheight 146, 679
 \settowidth 146, 679
 \Short §8.9.1; 619, 623, 817
 \ShortPlural §8.9.1; 620, 623, 817
 \show 159, 242, 306
 \Siblings §9.5.0.1; Table 9.1; 643, 817
 \siblings §9.5.0.1; Table 9.1; 643, 817
 siunitx package 14, 15, 26, 45, 108, 484, 764
 \Sort §8.9.1; 620, 624, 817
 sorted element 157, 159, 666
 \space 70, 140, 151, 166–168, 680, 684

T

\t (tab) Table 3.1; 350
 tab character Table 3.1; 181, 182, 202, 337, 350, 353, 354, 359, 360, 666, 766
 tabu package 245, 254
 tabular environment 223, 235, 241–243, 245, 246, 249, 250, 253, 254, 256–258, 260, 275, 277, 279, 282, 284, 285, 293, 301, 303–306, 318, 370, 394, 514, 515, 549, 677, 709, 710, 712
 \tabularnewline 250, 255, 275, 282
 tabularray package 245, 254, 258
 tblr environment 245, 254, 258
 \techreportname 818
 \TeX 168
 \Text §8.9.1; 620, 623, 818
 \textasciicircum Table 3.1
 \textasciitilde Table 3.1
 \textbackslash Table 3.1; 348, 349
 \textcurrency 123
 \textdollar 123, 210
 \texteuro 123
 \text_lowercase:n 148
 \text_map_inline:nn 148
 \textminus 128
 \textperiodcentered 119
 \text_purify:n 666
 \textsc 15, 545, 547, 548
 \textsmaller 15
 \textsterling 66, 123
 \text_titlecase_first:n 52, 58
 \text_uppercase:n 4, 52, 148
 \textwon 123
 \textyen 123
 thebibliography environment 552, 571, 823
 \Thee §9.5.0.1; Table 9.1; 639, 818
 \thee §9.5.0.1; Table 9.1; 639, 818
 \Their §9.5.0.1; Table 9.1; 640, 818

- `\their` §9.5.0.1; Table 9.1; 640, 818
`\Theirs` §9.5.0.1; Table 9.1; 641, 818
`\theirs` §9.5.0.1; Table 9.1; 641, 819
`\Them` §9.5.0.1; Table 9.1; 639, 819
`\them` §9.5.0.1; Table 9.1; 639, 819
`\They` §9.5.0.1; Table 9.1; 638, 819
`\they` §9.5.0.1; Table 9.1; 638, 819
`tikz` package ... 377, 435, 442, 451, 463, 464,
466, 467, 469, 471, 488, 496, 498,
503, 511, 514, 524, 692, 693, 745,
750, 774, 775
`tikzpicture` environment .. 398, 449, 457, 510,
511, 689, 690, 755, 756
`\today` 339
`\toprule` 269
`tracklang` package .. 9, 10, 27, 28, 31, 54, 55,
174, 574, 628, 651, 657, 826
`\TrackLangAddToCaptions` 54, 60
`\TrackLangEncodingName` 55, 338
`\TrackLangProvidesResource`
54
`\TrackLangShowWarningstrue` 9
`\TrackLanguageTag` 10, 826
`TSV` .. 177, 181, 182, 201, 202, 239, 336, 338,
347, 350, 354, 359, 360, 475, 666,
745, 766, 782
`\two@digits` 98, 175, 754
`\twocolumn` 582, 583, 611, 816
`\twopeoplesep` §9.5.2; 645, 819
- U**
- `\unexpanded` 280
`uppercase` 4, 47, 71, 148, 169, 175, 731
`\url` 548, 549, 565
`url` package 547–549, 565
`\Used` §8.9.1; 619, 624, 819
`\USEentry` §8.5; 600, 820
`\Useentry` §8.5; 600, 601–603,
778–780, 820
`\useentry` §8.5; 578, 589, 599, 600–603,
605, 607, 778–780, 820
`\USEentrynl` §8.5; 600, 820
`\Useentrynl` §8.5; 600, 603, 779,
780, 820
`\useentrynl` §8.5; 600, 602, 779,
780, 820
`UTF-8` .. 4, 29, 31, 51, 55, 56, 144, 147, 148,
169, 172–174, 273, 666
- V**
- `\volumename` 820
- W**
- `\whiledo` 89
- X**
- `\xcapitalisewords` 585
`\xdtlgetrowindex` .. §3.16; 370, 821
`\xDTLinitials` §2.8.2; 140, 821
`xfor` package 628, 656
`xkeyval` package 4, 242, 377, 399, 451,
526, 577
`\xmakefirstuc` 585
- Y**
- `\You` §9.5.0.1; Table 9.1; 638, 639, 821
`\you` §9.5.0.1; Table 9.1; 638, 639, 821
`\Your` §9.5.0.1; Table 9.1; 641, 821
`\your` §9.5.0.1; Table 9.1; 641, 821
`\Yours` §9.5.0.1; Table 9.1; 642, 821
`\yours` §9.5.0.1; Table 9.1; 642, 822