datatool v 2.02: Databases and data manipulation

Nicola L.C. Talbot

School of Computing Sciences
University of East Anglia
Norwich. Norfolk
NR4 7TJ. United Kingdom.
http://theoval.cmp.uea.ac.uk/~nlct/

13th July 2009

Contents

List of Examples

List of Figures

List of Tables

1 Introduction

The datatool bundle consists of the following packages: datatool, datapie, dataplot, databar and databib.

The datatool 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.
- Determine whether an argument is an integer, a real number, currency or a string. (Scientific notation is currently not supported.) Locale dependent number settings are supported (such as a comma as a decimal character and a full stop as a number group character).
- Convert locale dependent numbers/currency to the decimal format required by the fp package, enabling fixed point arithmetic to be performed on elements of the database.
- Names can be converted to initials.
- Strings can be tested to determine if they are all upper or lower case.
- String comparisons (both case sensitive and case insensitive) can be performed.

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

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

(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 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.

The databib package can be used to convert a BibTeX database into a datatool database.

2 Data Types

The datatool package recognises four data types: integers, real numbers, currency and strings.

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` (see below).

Real Numbers A real number is an integer followed by the decimal character followed by one or more digits. The decimal character is a full stop (.) by default. The number group and decimal characters may be changed using

`\DTLsetnumberchars`

`\DTLsetnumberchars{`⟨*number group character*⟩`}{`⟨*decimal character*⟩`}`

Note that scientific notation is not supported, and the number group character may not be used after the decimal character.

Currency A currency symbol followed by an integer or real number is considered to be the currency data type. There are two predefined currency symbols, `\$` and `\pounds`. In addition, if any of the following commands are defined at the start of the document, they are also considered to be a currency symbol: `\texteuro`, `\textdollar`, `\textstirling`, `\textyen`, `\textwon`, `\textcurrency`, `\euro` and `\yen`. Additional currency symbols can be defined using

`\DTLnewcurrencysymbol`

`\DTLnewcurrencysymbol{`⟨*symbol*⟩`}`

Strings Anything that doesn't belong to the above three types is considered to be a string.

2.1 Conditionals

The following conditionals are provided by the datatool package:

`\DTLifint`

`\DTLifint{`⟨*text*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

If ⟨*text*⟩ is an integer then do ⟨*true part*⟩, otherwise do ⟨*false part*⟩. For example

```
\DTLifint{2536}{integer}{not an integer}
```

produces: integer.

The number group character may appear in the number, for example:

```
\DTLifint{2,536}{integer}{not an integer}
```

produces: integer. However, the number group character may only be followed by a group of three digits. For example:

```
\DTLifint{2,5,3,6}{integer}{not an integer}
```

produces: not an integer. The number group character may be changed. For example:

```
\DTLsetnumberchars{.}{,}%
\DTLifint{2,536}{integer}{not an integer}
```

this now produces: not an integer, since 2,536 is now a real number.

Note that nothing else can be appended or prepended to the number. For example:

```
\DTLsetnumberchars{,}{.}%
\DTLifint{2,536m}{integer}{not an integer}
```

produces: not an integer.

`\DTLifreal`

\DTLifreal{⟨*text*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

If ⟨*text*⟩ is a real number then do ⟨*true part*⟩, otherwise do ⟨*false part*⟩. For example

```
\DTLifreal{1000.0}{real}{not real}
```

produces: real.

Note that an integer is not a real number:

```
\DTLifreal{1,000}{real}{not real}
```

produces: not real.

Whereas

```
\DTLifreal{1,000.0}{real}{not real}
```

produces: real.

However

```
\DTLsetnumberchars{.}{,}%
\DTLifreal{1,000}{real}{not real}
```

produces: real since the comma is now the decimal character.

Currency is not considered to be real:

```
\DTLsetnumberchars{,}{.}%
\DTLifreal{\$1.00}{real}{not real}
```

produces: not real.

`\DTLifcurrency`

\DTLifcurrency{⟨*text*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

If ⟨*text*⟩ is currency, then do ⟨*true part*⟩, otherwise do false part. For example:

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

produces: currency. Similarly:

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

produces: currency. Note, however, that

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

produces: not currency. If you want this to be considered currency, you will have to add the sequence `US\$` to the set of currency symbols:

```
\DTLnewcurrencysymbol{US\$}%
\DTLifcurrency{US\$5.99}{currency}{not currency}
```

this now produces: currency.

This document has used the `textcomp` package which defines `\texteuro`, so this is also considered to be currency. For example:

```
\DTLifcurrency{\texteuro5.99}{currency}{not currency}
```

produces: currency.

The preferred method is to display the euro symbol in a sans-serif font, but

```
\DTLifcurrency{\textsf{\texteuro}5.99}{currency}{not currency}
```

will produce: not currency.

It is better to define a new command, for example:

```
\DeclareRobustCommand*{\euro}{\textsf{\texteuro}}
```

and add that command to the list of currency symbols. In fact, in this case, if you define the command `\euro` in the preamble, it will automatically be added to the list of known currency symbols. If however you define `\euro` in the document, you will have to add it using `\DTLnewcurrencysymbol`. For example:

```
\newcommand*{\euro}{\textsf{\texteuro}}%
\DTLnewcurrencysymbol{\euro}%
\DTLifcurrency{\euro5.99}{currency}{not currency}
```

produces: currency.

`\DTLifcurrencyunit`

`\DTLifcurrencyunit{`⟨*text*⟩`}{`⟨*symbol*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

If ⟨*text*⟩ is currency, and uses ⟨*symbol*⟩ as the unit of currency, then do ⟨*true part*⟩ otherwise do ⟨*false part*⟩. For example:

```
\DTLifcurrencyunit{\$6.99}{\$}{dollars}{not dollars}
```

produces: dollars. Another example:

```
\def\cost{\euro10.50}%
\DTLifcurrencyunit{\cost}{\euro}{euros}{not euros}
```

produces: euros.

`\DTLifnumerical`

`\DTLifnumerical{`⟨*text*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

If ⟨*text*⟩ is numerical (either an integer, real number or currency) then do ⟨*true part*⟩ otherwise do ⟨*false part*⟩. For example:

```
\DTLifnumerical{1,000.0}{number}{string}.
```

produces: number. Whereas

```
\DTLsetnumberchars{.}{,}%
\DTLifnumerical{1,000.0}{number}{string}.
```

produces: string. Since the number group character is now a full stop, and the decimal character is now a comma. (The number group character may only appear before the decimal character, not after it.)

Currency is also considered to be numerical:

```
\DTLsetnumberchars{,}{.}%
\DTLifnumerical{\$1,000.0}{number}{string}.
```

produces: number.

`\DTLifstring`

`\DTLifstring{`⟨*text*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

This is the opposite of `\DTLifnumerical`. If ⟨*text*⟩ is not numerical, do ⟨*true part*⟩, otherwise do ⟨*false part*⟩.

`\DTLifcasedatatype`

`\DTLifcasedatatype{`⟨*text*⟩`}{`⟨*string case*⟩`}{`⟨*int case*⟩`}{`⟨*real case*⟩`}{`⟨*currency case*⟩`}`

If ⟨*text*⟩ is a string do ⟨*string case*⟩, if ⟨*text*⟩ is an integer do ⟨*int case*⟩, if ⟨*text*⟩ is a real number do ⟨*real case*⟩, if ⟨*text*⟩ is currency do ⟨*currency case*⟩. For example:

```
\DTLifcasedatatype{1,000}{string}{integer}{real}{currency}
```

produces: integer.

`\DTLifnumeq`

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

If ⟨*num1*⟩ is equal to ⟨*num2*⟩, then do ⟨*true part*⟩, otherwise do ⟨*false part*⟩. Note that both ⟨*num1*⟩ and ⟨*num2*⟩ must be numerical (either integers, real numbers or currency). The currency symbol is ignored when determining equality. For example

```
\DTLifnumeq{\pounds10.50}{10.5}{true}{false}
```

produces: true, since they are considered to be numerically equivalent. Likewise:

```
\DTLifnumeq{\pounds10.50}{\$10.50}{true}{false}
```

produces: true.

`\DTLifstringeq`

`\DTLifstringeq{`⟨*string1*⟩`}{`⟨*string2*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

`\DTLifstringeq*`

`\DTLifstringeq*{`⟨*string1*⟩`}{`⟨*string2*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

If ⟨*string1*⟩ and ⟨*string2*⟩ are the same, then do ⟨*true part*⟩, otherwise do ⟨*false part*⟩. The starred version ignores the case, the unstarred version is case sensitive. Both ⟨*string1*⟩ and ⟨*string2*⟩ are considered to be strings, so for example:

```
\DTLifstringeq{10.50}{10.5}{true}{false}
```

produces: false.

Note that

```
\DTLifstringeq{Text}{text}{true}{false}
```

produces: false, whereas

```
\DTLifstringeq*{Text}{text}{true}{false}
```

produces: true, however it should also be noted that many commands will be ignored, so:

```
\DTLifstringeq{\uppercase{t}ext}{text}{true}{false}
```

produces: true.

Spaces are considered to be equivalent to `\space` and `~`. For example:

```
\DTLifstringeq{an apple}{an~apple}{true}{false}
```

produces: true. Consecutive spaces are treated as the same, for example:

```
\DTLifstringeq{an  apple}{an apple}{true}{false}
```

produces: true.

`\DTLifeq`

`\DTLifeq{`⟨*arg1*⟩`}{`⟨*arg2*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

`\DTLifeq*`

`\DTLifeq*{`⟨*arg1*⟩`}{`⟨*arg2*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

If both ⟨*arg1*⟩ and ⟨*arg2*⟩ are numerical, then this is equivalent to `\DTLifnumeq`, otherwise it is equivalent to `\DTLifstringeq` (when using `\DTLifeq`) or `\DTLifstringeq*` (when using `\DTLifeq*`).

`\DTLifnumlt`

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

If ⟨*num1*⟩ is less than ⟨*num2*⟩, then do ⟨*true part*⟩, otherwise do ⟨*false part*⟩. Note that both ⟨*num1*⟩ and ⟨*num2*⟩ must be numerical (either integers, real numbers or currency).

`\DTLifstringlt`

`\DTLifstringlt{`⟨*string1*⟩`}{`⟨*string2*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

`\DTLifstringlt*`

`\DTLifstringlt*{`⟨*string1*⟩`}{`⟨*string2*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

If ⟨*string1*⟩ is alphabetically less than ⟨*string2*⟩, then do ⟨*true part*⟩, otherwise do ⟨*false part*⟩. The starred version ignores the case, the unstarred version is case sensitive. For example:

```
\DTLifstringlt{aardvark}{zebra}{less}{not less}
```

produces: less.

Note that both ⟨*string1*⟩ and ⟨*string2*⟩ are considered to be strings, so for example:

```
\DTLifstringlt{2}{10}{less}{not less}
```

produces: not less, since the string `2` comes after the string `10` when arranged alphabetically.

The case sensitive (unstarred) version considers uppercase characters to be less than lowercase characters, so

```
\DTLifstringlt{B}{a}{less}{not less}
```

produces: less, whereas

```
\DTLifstringlt*{B}{a}{less}{not less}
```

produces: not less.

`\DTLiflt`

`\DTLiflt{`⟨*arg1*⟩`}{`⟨*arg2*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

`\DTLiflt*`

`\DTLiflt*{`⟨*arg1*⟩`}{`⟨*arg2*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

If ⟨*arg1*⟩ and ⟨*arg2*⟩ are both numerical, then this is equivalent to `\DTLifnumlt`, otherwise it is equivalent to `\DTLstringlt` (when using `\DTLiflt`) or `\DTLstringlt*` (when using `\DTLiflt*`).

`\DTLifnumgt`

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

If ⟨*num1*⟩ is greater than ⟨*num2*⟩, then do ⟨*true part*⟩, otherwise do ⟨*false part*⟩. Note that both ⟨*num1*⟩ and ⟨*num2*⟩ must be numerical (either integers, real numbers or currency).

`\DTLifstringgt`

`\DTLifstringgt{`⟨*string1*⟩`}{`⟨*string2*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

`\DTLifstringgt*`

`\DTLifstringgt*{`⟨*string1*⟩`}{`⟨*string2*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

If ⟨*string1*⟩ is alphabetically greater than ⟨*string2*⟩, then do ⟨*true part*⟩, otherwise do ⟨*false part*⟩. The starred version ignores the case, the unstarred version is case sensitive. For example:

```
\DTLifstringgt{aardvark}{zebra}{greater}{not greater}
```

produces: not greater.

Note that both ⟨*string1*⟩ and ⟨*string2*⟩ are considered to be strings, so for example:

```
\DTLifstringgt{2}{10}{greater}{not greater}
```

produces: greater, since the string 2 comes after the string 10 when arranged alphabetically.

As with \DTLifstringlt, uppercase characters are considered to be less than lower case characters when performing a case sensitive comparison so:

```
\DTLifstringgt{B}{a}{greater}{not greater}
```

produces: not greater, whereas

```
\DTLifstringgt*{B}{a}{greater}{not greater}
```

produces: greater.

\DTLifgt

\DTLifgt{⟨*arg1*⟩}{⟨*arg2*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

\DTLifgt*

\DTLifgt*{⟨*arg1*⟩}{⟨*arg2*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

If ⟨*arg1*⟩ and ⟨*arg2*⟩ are both numerical, then this is equivalent to \DTLifnumgt, otherwise it is equivalent to \DTLstringgt (when using \DTLifgt) or \DTLstringgt* (when using \DTLifgt*).

\DTLifnumclosedbetween

\DTLifnumclosedbetween{⟨*num*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

If ⟨*min*⟩ $\leq$ ⟨*num*⟩ $\leq$ ⟨*max*⟩ then do ⟨*true part*⟩, otherwise do ⟨*false part*⟩. Note that ⟨*num*⟩, ⟨*min*⟩ and ⟨*max*⟩ must be numerical (either integers, real numbers or currency). The currency symbol is ignored when determining equality. For example:

```
\DTLifnumclosedbetween{5.4}{5}{7}{inside}{outside}
```

produces: inside. Note that the closed range includes end points:

```
\DTLifnumclosedbetween{5}{5}{7}{inside}{outside}
```

produces: inside.

\DTLifstringclosedbetween

\DTLifstringclosedbetween{⟨*string*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

\DTLifstringclosedbetween*

\DTLifstringclosedbetween*{⟨*string*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

This determines if ⟨*string*⟩ is between ⟨*min*⟩ and ⟨*max*⟩ in the alphabetical sense, or is equal to either ⟨*min*⟩ or ⟨*max*⟩. The starred version ignores the case, the unstarred version is case sensitive.

\DTLifclosedbetween

\DTLifclosedbetween{⟨*arg*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

\DTLifclosedbetween*

\DTLifclosedbetween*{⟨*arg*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

If ⟨*arg*⟩, ⟨*min*⟩ and ⟨*max*⟩ are numerical, then this is equivalent to \DTLifnumclosedbetween, otherwise it is equivalent to \DTLifstringclosedbetween (when using \DTLifclosedbetween) or \DTLifstringclosedbetween* (when using \DTLifclosedbetween*).

\DTLifnumopenbetween

\DTLifnumopenbetween{⟨*num*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

If ⟨*min*⟩ < ⟨*num*⟩ < ⟨*max*⟩ then do ⟨*true part*⟩, otherwise do ⟨*false part*⟩. Note that ⟨*num*⟩, ⟨*min*⟩ and ⟨*max*⟩ must be numerical (either integers, real numbers or currency). Again, the currency symbol is ignored when determining equality. For example:

```
\DTLifnumopenbetween{5.4}{5}{7}{inside}{outside}
```

produces: inside. Note that end points are not included. For example:

```
\DTLifnumopenbetween{5}{5}{7}{inside}{outside}
```

produces: outside.

\DTLifstringopenbetween

\DTLifstringopenbetween{⟨*string*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

\DTLifstringopenbetween*

\DTLifstringopenbetween*{⟨*string*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

This determines if ⟨*string*⟩ is between ⟨*min*⟩ and ⟨*max*⟩ in the alphabetical sense. The starred version ignores the case, the unstarred version is case sensitive.

\DTLifopenbetween

\DTLifopenbetween{⟨*arg*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

\DTLifopenbetween*

\DTLifopenbetween*{⟨*arg*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

If ⟨*arg*⟩, ⟨*min*⟩ and ⟨*max*⟩ are numerical, then this is equivalent to \DTLifnumopenbetween, otherwise it is equivalent to \DTLifstringopenbetween (when using \DTLifopenbetween) \DTLifstringopenbetween* (when using \DTLifopenbetween*).

\DTLifFPclosedbetween

\DTLifFPclosedbetween{⟨*num*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

If ⟨*min*⟩ ≤ ⟨*num*⟩ ≤ ⟨*max*⟩ then do ⟨*true part*⟩, otherwise do ⟨*false part*⟩ where ⟨*num*⟩, ⟨*min*⟩ and ⟨*max*⟩ are all in standard fixed point notation (i.e. no number group separator, no currency symbols and a full stop as a decimal point).

\DTLifFPopenbetween

\DTLifFPopenbetween{⟨*num*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

If ⟨*min*⟩ < ⟨*num*⟩ < ⟨*max*⟩ then do ⟨*true part*⟩, otherwise do ⟨*false part*⟩ where ⟨*num*⟩, ⟨*min*⟩ and ⟨*max*⟩ are all in standard fixed point notation (i.e. no number group separator, no currency symbols and a full stop as a decimal point).

`\DTLifAllUpperCase`

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

Tests if ⟨*string*⟩ is all upper case. For example:

```
\DTLifAllUpperCase{WORD}{all upper}{not all upper}
```

produces: all upper, whereas

```
\DTLifAllUpperCase{Word}{all upper}{not all upper}
```

produces: not all upper. Note also that:

```
\DTLifAllUpperCase{\MakeUppercase{word}}{all upper}{not all upper}
```

also produces: all upper. `\MakeTextUppercase` (defined in David Carlisle's textcase package) and `\uppercase` are also detected, otherwise, if a command is encountered, the case of the command is considered. For example:

```
\DTLifAllUpperCase{MAN{\OE}UVRE}{all upper}{not all upper}
```

produces: all upper.

`\DTLifAllLowerCase`

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

Tests if ⟨*string*⟩ is all lower case. For example:

```
\DTLifAllLowerCase{word}{all lower}{not all lower}
```

produces: all lower, whereas

```
\DTLifAllLowerCase{Word}{all lower}{not all lower}
```

produces: not all lower. Note also that:

```
\DTLifAllLowerCase{\MakeLowercase{WORD}}{all lower}{not all lower}
```

also produces: all lower. `\MakeTextLowercase` (defined in David Carlisle's textcase package) and `\lowercase` are also detected, otherwise, if a command is encountered, the case of the command is considered. For example:

```
\DTLifAllLowerCase{man{\oe}uvre}{all lower}{not all lower}
```

produces: all lower.

`\DTLifSubString`

`\DTLifSubString{`⟨*string*⟩`}{`⟨*substring*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

This tests if ⟨*substring*⟩ is a sub-string of ⟨*string*⟩. This command performs a case sensitive match. For example:

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

produces: is substring. Note that spaces are considered to be equivalent to \space or ~, so

```
\DTLifSubString{An apple}{n~a}{is substring}{isn't substring}
```

produces: is substring, but other commands are skipped, so

```
\DTLifSubString{An \uppercase{a}pple}{app}{is substring}{isn't
substring}
```

produces: is substring, since the \uppercase command is ignored. Note also that grouping is ignored, so:

```
\DTLifSubString{An {ap}ple}{app}{is substring}{isn't substring}
```

produces: is substring.

\DTLifSubString is case sensitive, so:

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

produces: isn't substring.

\DTLifStartsWith

\DTLifStartsWith{⟨*string*⟩}{⟨*substring*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

This is like \DTLifSubString, except that ⟨*substring*⟩ must occur at the start of ⟨*string*⟩. This command performs a case sensitive match. For example,

```
\DTLifStartsWith{An apple}{app}{prefix}{not a prefix}
```

produces: not a prefix. All the above remarks for \DTLifSubString also applies to \DTLifStartsWith. For example:

```
\DTLifStartsWith{\uppercase{a}n apple}{an~}{prefix}{not a prefix}
```

produces: prefix, since \uppercase is ignored, and ~ is considered to be the same as a space, whereas

```
\DTLifStartsWith{An apple}{an~}{prefix}{not a prefix}
```

produces: not a prefix.

2.2 ifthen conditionals

The commands described in the previous section can not be used as the conditional part of the \ifthenelse or \whiledo commands provided by the ifthen package. This section describes analogous commands which may only be used in the conditional argument of \ifthenelse and \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.

\DTLisstring

\DTLisstring{⟨*text*⟩}

Tests if ⟨*text*⟩ is a string. For example:

```
\ifthenelse{\DTLisstring{some text}}{string}{not a string}
```

produces: string.

\DTLisnumerical

\DTLisnumerical{⟨*text*⟩}

Tests if ⟨*text*⟩ is numerical (i.e. not a string). For example:

```
\ifthenelse{\DTLisnumerical{\$10.95}}{numerical}{not numerical}
```

produces: numerical.

Note however that \DTLisnumerical requires more care than \DTLifnumerical when used with some of the other currency symbols. Consider:

```
\DTLifnumerical{\pounds10.95}{numerical}{not numerical}
```

This produces: numerical. However

```
\ifthenelse{\DTLisnumerical{\pounds10.95}}{numerical}{not numerical}
```

produces: not numerical. This is due to the expansion that occurs within \ifthenelse. This can be prevented using \noexpand, for example:

```
\ifthenelse{\DTLisnumerical{\noexpand\pounds10.95}}{numerical}{not numerical}
```

produces: numerical.

Likewise:

```
\def\cost{\pounds10.95}%
\ifthenelse{\DTLisnumerical{\noexpand\cost}}{numerical}{not numerical}
```

produces: numerical.

\DTLiscurrency

\DTLiscurrency{⟨*text*⟩}

Tests if ⟨*text*⟩ is currency. For example:

```
\ifthenelse{\DTLiscurrency{\$10.95}}{currency}{not currency}
```

produces: currency.

The same warning given above for \DTLisnumerical also applies here.

\DTLiscurrencyunit

\DTLiscurrencyunit{⟨*text*⟩}{⟨*symbol*⟩}

Tests if ⟨*text*⟩ is currency and that currency uses ⟨*symbol*⟩ as the unit of currency. For example:

```
\ifthenelse{\DTLiscurrencyunit{\$6.99}{\$}}{dollars}{not dollars}
```

produces: dollars. Another example:

```
\def\cost{\euro10.50}%
\ifthenelse{\DTLiscurrencyunit{\noexpand\cost}{\noexpand\euro}}%
{euros}{not euros}
```

produces: euros. Again note the use of `\noexpand`.

`\DTLisreal`

`\DTLisreal{`⟨*text*⟩`}`

Tests if ⟨*text*⟩ is a fixed point number (again, an integer is not considered to be a fixed point number). For example:

```
\ifthenelse{\DTLisreal{1.5}}{real}{not real}
```

produces: real.

`\DTLisint`

`\DTLisint{`⟨*text*⟩`}`

Tests if ⟨*text*⟩ is an integer. For example:

```
\ifthenelse{\DTLisint{153}}{integer}{not an integer}
```

produces: integer.

`\DTLislt`

`\DTLislt{`⟨*arg1*⟩`}{`⟨*arg2*⟩`}`

This checks if ⟨*arg1*⟩ is less than ⟨*arg2*⟩. As with `\DTLiflt`, if ⟨*arg1*⟩ and ⟨*arg2*⟩ are numerical, a numerical comparison is used, otherwise a case sensitive alphabetical comparison is used. (Note that there is no starred version of this command, but you can instead use `\DTLisilt` to ignore the case.)

`\DTLisilt`

`\DTLisilt{`⟨*arg1*⟩`}{`⟨*arg2*⟩`}`

This checks if ⟨*arg1*⟩ is less than ⟨*arg2*⟩. As with `\DTLiflt*`, if ⟨*arg1*⟩ and ⟨*arg2*⟩ are numerical, a numerical comparison is used, otherwise a case insensitive alphabetical comparison is used.

`\DTLisgt`

`\DTLisgt{`⟨*arg1*⟩`}{`⟨*arg2*⟩`}`

This checks if ⟨*arg1*⟩ is greater than ⟨*arg2*⟩. As with `\DTLifgt`, if ⟨*arg1*⟩ and ⟨*arg2*⟩ are numerical, a numerical comparison is used, otherwise a case sensitive alphabetical comparison is used. (Note that there is no starred version of this command, instead use `\DTLisigt` to ignore the case.)

`\DTLisigt`

`\DTLisigt{`⟨*arg1*⟩`}{`⟨*arg2*⟩`}`

This checks if ⟨*arg1*⟩ is greater than ⟨*arg2*⟩. As with `\DTLifgt*`, if ⟨*arg1*⟩ and ⟨*arg2*⟩ are numerical, a numerical comparison is used, otherwise a case insensitive alphabetical comparison is used.

`\DTLiseq`

`\DTLiseq{`⟨*arg1*⟩`}{`⟨*arg2*⟩`}`

This checks if ⟨*arg1*⟩ is equal to ⟨*arg2*⟩. As with `\DTLifeq`, if ⟨*arg1*⟩ and ⟨*arg2*⟩ are numerical, a numerical comparison is used, otherwise a case sensitive

alphabetical comparison is used. (Note that there is no starred version of this command, instead use `\DTLisieq`.)

`\DTLisieq`

`\DTLisieq{`⟨*arg1*⟩`}{`⟨*arg2*⟩`}`

This checks if ⟨*arg1*⟩ is equal to ⟨*arg2*⟩. As with `\DTLifeq*`, if ⟨*arg1*⟩ and ⟨*arg2*⟩ are numerical, a numerical comparison is used, otherwise a case insensitive alphabetical comparison is used.

`\DTLisclosedbetween`

`\DTLisclosedbetween{`⟨*arg*⟩`}{`⟨*min*⟩`}{`⟨*max*⟩`}`

This checks if ⟨*arg*⟩ lies between ⟨*min*⟩ and ⟨*max*⟩ (end points included). As with `\DTLifclosedbetween`, if the arguments are numerical, a numerical comparison is used, otherwise a case sensitive alphabetical comparison is used. (Note that there is no starred version of this command, instead use `\DTLisiclosedbetween`.)

`\DTLisiclosedbetween`

`\DTLisiclosedbetween{`⟨*arg*⟩`}{`⟨*min*⟩`}{`⟨*max*⟩`}`

This checks if ⟨*arg*⟩ lies between ⟨*min*⟩ and ⟨*max*⟩ (end points included). As with `\DTLifclosedbetween*`, if the arguments are numerical, a numerical comparison is used, otherwise a case insensitive alphabetical comparison is used.

`\DTLisopenbetween`

`\DTLisopenbetween{`⟨*arg*⟩`}{`⟨*min*⟩`}{`⟨*max*⟩`}`

This checks if ⟨*arg*⟩ lies between ⟨*min*⟩ and ⟨*max*⟩ (end points excluded). As with `\DTLifopenbetween`, if the arguments are numerical, a numerical comparison is used, otherwise a case sensitive alphabetical comparison is used. (Note that there is no starred version of this command, instead use `\DTLisiopenbetween`.)

`\DTLisiopenbetween`

`\DTLisiopenbetween{`⟨*arg*⟩`}{`⟨*min*⟩`}{`⟨*max*⟩`}`

This checks if ⟨*arg*⟩ lies between ⟨*min*⟩ and ⟨*max*⟩ (end points excluded). As with `\DTLifopenbetween*`, if the arguments are numerical, a numerical comparison is used, otherwise a case insensitive alphabetical comparison is used.

`\DTLisFPlt`

`\DTLisFPlt{`⟨*num1*⟩`}{`⟨*num2*⟩`}`

This checks if ⟨*num1*⟩ is less than ⟨*num2*⟩, where both numbers are in standard fixed point format (i.e. no number group separators, no currency and a full stop as a decimal point).

`\DTLisFPlteq`

`\DTLisFPlteq{`⟨*num1*⟩`}{`⟨*num2*⟩`}`

This checks if ⟨*num1*⟩ is less than or equal to ⟨*num2*⟩, where both numbers are in standard fixed point format (i.e. no number group separators, no currency and a full stop as a decimal point).

`\DTLisFPgt`

```
\DTLisFPgt{⟨num1⟩}{⟨num2⟩}
```

This checks if ⟨*num1*⟩ is greater than ⟨*num2*⟩, where both numbers are in standard fixed point format (i.e. no number group separators, no currency and a full stop as a decimal point).

`\DTLisFPgteq`

```
\DTLisFPgteq{⟨num1⟩}{⟨num2⟩}
```

This checks if ⟨*num1*⟩ is greater than or equal to ⟨*num2*⟩, where both numbers are in standard fixed point format (i.e. no number group separators, no currency and a full stop as a decimal point).

`\DTLisFPeq`

```
\DTLisFPeq{⟨num1⟩}{⟨num2⟩}
```

This checks if ⟨*num1*⟩ is equal to ⟨*num2*⟩, where both numbers are in standard fixed point format (i.e. no number group separators, no currency and a full stop as a decimal point).

`\DTLisFPclosedbetween`

```
\DTLisFPclosedbetween{⟨num⟩}{⟨min⟩}{⟨max⟩}
```

This checks if ⟨*num*⟩ lies between ⟨*min*⟩ and ⟨*max*⟩ (end points included). All arguments must be numbers in standard fixed point format (i.e. no number group separators, no currency and a full stop as a decimal point).

`\DTLisFPopenbetween`

```
\DTLisFPopenbetween{⟨num⟩}{⟨min⟩}{⟨max⟩}
```

This checks if ⟨*num*⟩ lies between ⟨*min*⟩ and ⟨*max*⟩ (end points excluded). All arguments must be numbers in standard fixed point format (i.e. no number group separators, no currency and a full stop as a decimal point).

`\DTLisSubString`

```
\DTLisSubString{⟨string⟩}{⟨substring⟩}
```

This checks if ⟨*substring*⟩ is contained in ⟨*string*⟩. The remarks about `\DTLifSubString` also apply to `\DTLisSubString`. This command performs a case sensitive match.

`\DTLisPrefix`

```
\DTLisPrefix{⟨string⟩}{⟨prefix⟩}
```

This checks if ⟨*string*⟩ starts with ⟨*prefix*⟩. The remarks about `\DTLifStartsWith` also apply to `\DTLisPrefix`. This command performs a case sensitive match.

3 Fixed Point Arithmetic

The datatool package uses the fp package to perform fixed point arithmetic, however all numbers must be converted from the locale dependent format into the format required by the fp package. A numerical value (i.e. an integer, a real or currency) can be converted into a plain decimal number using

\DTLconverttodecimal

\DTLconverttodecimal{⟨*num*⟩}{⟨*cmd*⟩}

The decimal number will be stored in ⟨*cmd*⟩ which must be a control sequence. For example:

```
\DTLconverttodecimal{1,563.54}{\mynum}
```

will define \mynum to be 1563.54. The command \mynum can then be used in any of the arithmetic macros provided by the fp package. There are two commands provided to perform the reverse:

\DTLdecimaltolocale

\DTLdecimaltolocale{⟨*number*⟩}{⟨*cmd*⟩}

This converts a plain decimal number ⟨*number*⟩ (that uses a full stop as the decimal character and has no number group characters) into a locale dependent format. The resulting number is stored in ⟨*cmd*⟩, which must be a control sequence. For example:

```
\DTLdecimaltolocale{6795.3}{\mynum}
```

will define \mynum to be 6,795.3.

\DTLdecimaltocurrency

\DTLdecimaltocurrency{⟨*number*⟩}{⟨*cmd*⟩}

This will convert a plain decimal number ⟨*number*⟩ into a locale dependent currency format. For example:

```
\DTLdecimaltocurrency{267.5}{\price}\price
```

will produce: £267.50.

The currency symbol used by \DTLdecimaltocurrency is initially \$, but will use the currency last encountered. So, for example

```
\DTLifcurrency{\texteuro45.00}{}{}%
\DTLdecimaltocurrency{267.5}{\price}\price
```

will produce: €267.50. This is because the last currency symbol to be encountered was \texteuro. You can reset the currency symbol using the command:

\DTLsetdefaultcurrency

\DTLsetdefaultcurrency{⟨*symbol*⟩}

For example:

```
\DTLsetdefaultcurrency{\textyen}%
\DTLdecimaltocurrency{267.5}{\price}\price
```

will produce: ¥267.50

The datatool package provides convenience commands which use \DTLconverttodecimal, and then use the basic macros provided by the fp package. The resulting value is then converted back into the locale format using \DTLdecimaltolocale or \DTLdecimaltocurrency.

\DTLadd

\DTLadd{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

\DTLgadd

\DTLgadd{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

This sets the control sequence ⟨*cmd*⟩ to ⟨*num1*⟩+⟨*num2*⟩. \DLTadd sets ⟨*cmd*⟩ locally, while \DTLgadd sets ⟨*cmd*⟩ globally.

For example:

```
\DTLadd{\result}{3,562.65}{412.2}\result
```

will produce: 3,974.85. Since \DTLconverttodecimal can convert currency to a real number, you can also add prices. For example:

```
\DTLadd{\result}{\pounds3,562.65}{\pounds452.2}\result
```

produces: £4,014.85.

Note that datatool isn't aware of exchange rates! If you use different currency symbols, the last symbol will be used. For example

```
\DTLadd{\result}{\pounds3,562.65}{\euro452.2}\result
```

produces: €4,014.85.

Likewise, if one value is a number and the other is a currency, the type of the last value, ⟨*num2*⟩, will be used for the result. For example:

```
\DTLadd{\result}{3,562.65}{\$452.2}\result
```

produces: $4,014.85.

\DTLaddall

\DTLaddall{⟨*cmd*⟩}{⟨*number list*⟩}

\DTLgaddall

\DTLgaddall{⟨*cmd*⟩}{⟨*number list*⟩}

This sets the control sequence ⟨*cmd*⟩ to the sum of all the numbers in ⟨*number list*⟩. \DLTaddall sets ⟨*cmd*⟩ locally, while \DTLgaddall sets ⟨*cmd*⟩ globally. Example:

```
\DTLaddall{\total}{25.1,45.2,35.6}\total
```

produces: 105.9. Note that if any of the numbers in ⟨*number list*⟩ contain a comma, you must group the number. Example:

```
\DTLaddall{\total}{{1,525},{2,340},500}\total
```

produces: 4,365.

\DTLsub

\DTLsub{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

\DTLgsub

\DTLgsub{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

This sets the control sequence ⟨*cmd*⟩ to ⟨*num1*⟩−⟨*num2*⟩. \DLTsub sets ⟨*cmd*⟩ locally, while \DTLgsub sets ⟨*cmd*⟩ globally.

For example:

```
\DTLsub{\result}{3,562.65}{412.2}\result
```

will produce: 3,150.45. As with \DTLadd, ⟨*num1*⟩ and ⟨*num2*⟩ may be currency.

\DTLmul

\DTLmul{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

\DTLgmul

\DTLgmul{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

This sets the control sequence ⟨*cmd*⟩ to ⟨*num1*⟩×⟨*num2*⟩. \DLTmul sets ⟨*cmd*⟩ locally, while \DTLgmul sets ⟨*cmd*⟩ globally.

For example:

```
\DTLmul{\result}{568.95}{2}\result
```

will produce: 1,137.9. Again, ⟨*num1*⟩ or ⟨*num2*⟩ may be currency, but unlike \DTLadd and \DTLsub, currency overrides integer/real. For example:

```
\DTLmul{\result}{\pounds568.95}{2}\result
```

will produce: £1,137.90. Likewise,

```
\DTLmul{\result}{2}{\pounds568.95}\result
```

will produce: £1,137.90. Although it doesn't make sense to multiply two currencies, datatool will allow

```
\DTLmul{\result}{\$2}{\pounds568.95}\result
```

which will produce: £1,137.90.

\DTLdiv

\DTLdiv{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

\DTLgdiv

\DTLgdiv{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

This sets the control sequence ⟨*cmd*⟩ to ⟨*num1*⟩÷⟨*num2*⟩. \DLTdiv sets ⟨*cmd*⟩ locally, while \DTLgdiv sets ⟨*cmd*⟩ globally.

For example:

```
\DTLdiv{\result}{501}{2}\result
```

will produce: 250.5. Again, ⟨*num1*⟩ or ⟨*num2*⟩ may be currency, but the resulting type will be not be a currency if both ⟨*num1*⟩ and ⟨*num2*⟩ use the same currency symbol. For example:

```
\DTLdiv{\result}{\$501}{\$2}\result
```

will produce: 250.5. Whereas

```
\DTLdiv{\result}{\$501}{2}\result
```

will produce: $250.50.

\DTLabs

\DTLabs{⟨*cmd*⟩}{⟨*num*⟩}

\DTLgabs

\DTLgabs{⟨*cmd*⟩}{⟨*num*⟩}

This sets ⟨*cmd*⟩ to the absolute value of ⟨*num*⟩. \DLTabs sets ⟨*cmd*⟩ locally, while \DTLgabs sets ⟨*cmd*⟩ globally. Example:

```
\DTLabs{\result}{-\pounds2.50}\result
```

produces: £2.50.

\DTLneg

\DTLneg{⟨*cmd*⟩}{⟨*num*⟩}

\DTLgneg

\DTLgneg{⟨*cmd*⟩}{⟨*num*⟩}

This sets ⟨*cmd*⟩ to the negative of ⟨*num*⟩. \DLTneg sets ⟨*cmd*⟩ locally, while \DTLgneg sets ⟨*cmd*⟩ globally. Example:

```
\DTLneg{\result}{\pounds2.50}\result
```

produces: -£2.50.

\DTLsqrt

\DTLsqrt{⟨*cmd*⟩}{⟨*num*⟩}

\DTLgsqrt

\DTLgsqrt{⟨*cmd*⟩}{⟨*num*⟩}

This sets ⟨*cmd*⟩ to the sqrt root of ⟨*num*⟩. \DLTsqrt sets ⟨*cmd*⟩ locally, while \DTLgsqrt sets ⟨*cmd*⟩ globally. Example:

```
\DTLsqrt{\result}{2}\result
```

produces: 1.414213562373095042.

\DTLmin

\DTLmin{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

\DTLgmin

\DTLgmin{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

This sets the control sequence ⟨*cmd*⟩ to the minimum of ⟨*num1*⟩ and ⟨*num2*⟩. \DLTmin sets ⟨*cmd*⟩ locally, while \DTLgmin sets ⟨*cmd*⟩ globally. For example:

```
\DTLmin{\result}{256}{32}\result
```

produces: 32. Again, ⟨*num1*⟩ and ⟨*num2*⟩ may be currency. For example:

```
\DTLmin{\result}{256}{\pounds32}\result
```

produces: £32, whereas

```
\DTLmin{\result}{\pounds256}{32}\result
```

produces: 32. As mentioned above, datatool doesn't know about exchange rates, so be careful about mixing currencies. For example:

```
\DTLmin{\result}{\pounds5}{\$6}\result
```

produces: £5, which may not necessarily be true!

\DTLminall

\DTLminall{⟨*cmd*⟩}{⟨*number list*⟩}

\DTLgminall

\DTLgminall{⟨*cmd*⟩}{⟨*number list*⟩}

This sets the control sequence ⟨*cmd*⟩ to the minimum of all the numbers in ⟨*number list*⟩. \DLTminall sets ⟨*cmd*⟩ locally, while \DTLgminall sets ⟨*cmd*⟩ globally. Example:

```
\DTLminall{\theMin}{25.1,45.2,35.6}\theMin
```

produces: 25.1. Note that if any of the numbers in ⟨*number list*⟩ contain a comma, you must group the number. Example:

```
\DTLminall{\theMin}{{1,525},{2,340},500}\theMin
```

produces: 500.

\DTLmax

\DTLmax{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

\DTLgmax

\DTLgmax{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

This sets the control sequence ⟨*cmd*⟩ to the maximum of ⟨*num1*⟩ and ⟨*num2*⟩. \DLTmax sets ⟨*cmd*⟩ locally, while \DTLgmax sets ⟨*cmd*⟩ globally. For example:

```
\DTLmax{\result}{256}{32}\result
```

produces: 256. Again, ⟨*num1*⟩ and ⟨*num2*⟩ may be currency, but the same warnings for \DTLmin apply.

\DTLmaxall

\DTLmaxall{⟨*cmd*⟩}{⟨*number list*⟩}

\DTLgmaxall

\DTLgmaxall{⟨*cmd*⟩}{⟨*number list*⟩}

This sets the control sequence ⟨*cmd*⟩ to the maximum of all the numbers in ⟨*number list*⟩. \DLTmaxall sets ⟨*cmd*⟩ locally, while \DTLgmaxall sets ⟨*cmd*⟩ globally. Example:

```
\DTLmaxall{\theMax}{25.1,45.2,35.6}\theMax
```

produces: 45.2. Note that if any of the numbers in ⟨*number list*⟩ contain a comma, you must group the number. Example:

```
\DTLmaxall{\theMax}{{1,525},{2,340},500}\theMax
```

produces: 2,340.

\DTLmeanforall

\DTLmeanforall{⟨*cmd*⟩}{⟨*number list*⟩}

\DTLgmeanall

\DTLgmeanforall{⟨*cmd*⟩}{⟨*number list*⟩}

This sets the control sequence ⟨*cmd*⟩ to the arithmetic mean of all the numbers in ⟨*number list*⟩. \DLTmeanforall sets ⟨*cmd*⟩ locally, while \DTLgmeanforall sets ⟨*cmd*⟩ globally. Example:

```
\DTLmeanforall{\theMean}{25.1,45.2,35.6}\theMean
```

produces: 35.3. Note that if any of the numbers in ⟨*number list*⟩ contain a comma, you must group the number. Example:

```
\DTLmeanforall{\theMean}{{1,525},{2,340},500}\theMean
```

produces: 1,455.

\DTLvarianceforall

\DTLvarianceforall{⟨*cmd*⟩}{⟨*number list*⟩}

\DTLgvarianceforall

\DTLgvarianceforall{⟨*cmd*⟩}{⟨*number list*⟩}

This sets the control sequence ⟨*cmd*⟩ to the variance of all the numbers in ⟨*number list*⟩. \DLTvarianceforall sets ⟨*cmd*⟩ locally, while \DTLgvarianceforall sets ⟨*cmd*⟩ globally. Example:

```
\DTLvarianceforall{\theVar}{25.1,45.2,35.6}\theVar
```

produces: 67.38. Again note that if any of the numbers in ⟨*number list*⟩ contain a comma, you must group the number.

\DTLsdforall

\DTLsdforall{⟨*cmd*⟩}{⟨*number list*⟩}

\DTLgsdforall

\DTLgsdforall{⟨*cmd*⟩}{⟨*number list*⟩}

This sets the control sequence ⟨*cmd*⟩ to the standard deviation of all the numbers in ⟨*number list*⟩. \DLTsdforall sets ⟨*cmd*⟩ locally, while \DTLgsdforall sets ⟨*cmd*⟩ globally. Example:

```
\DTLsdforall{\theSD}{25.1,45.2,35.6}\theSD
```

produces: 8.208532146492453016. Note that if any of the numbers in ⟨*number list*⟩ contain a comma, you must group the number. Example:

```
\DTLsdforall{\theSD}{{1,525},{2,340},500}\theSD
```

produces: 752.805862534735216539.

\DTLround

\DTLround{⟨*cmd*⟩}{⟨*num*⟩}{⟨*num digits*⟩}

\DTLground

\DTLground{⟨*cmd*⟩}{⟨*num*⟩}{⟨*num digits*⟩}

This sets ⟨*cmd*⟩ to ⟨*num*⟩ rounded to ⟨*num digits*⟩ after the decimal character. \DLTround sets ⟨*cmd*⟩ locally, while \DTLground sets ⟨*cmd*⟩ globally. Example:

```
\DTLround{\result}{3.135276}{2}\result
```

produces: 3.14.

\DTLtrunc

\DTLtrunc{⟨*cmd*⟩}{⟨*num*⟩}{⟨*num digits*⟩}

\DTLgtrunc

\DTLgtrunc{⟨*cmd*⟩}{⟨*num*⟩}{⟨*num digits*⟩}

This sets ⟨*cmd*⟩ to ⟨*num*⟩ truncated to ⟨*num digits*⟩ after the decimal character. \DLTtrunc sets ⟨*cmd*⟩ locally, while \DTLgtrunc sets ⟨*cmd*⟩ globally. Example:

```
\DTLtrunc{\result}{3.135276}{2}\result
```

produces: 3.13.

\DTLclip

\DTLclip{⟨*cmd*⟩}{⟨*num*⟩}

\DTLgclip

\DTLgclip{⟨*cmd*⟩}{⟨*num*⟩}

This sets ⟨*cmd*⟩ to ⟨*num*⟩ with all unnecessary 0's removed. \DLTclip sets ⟨*cmd*⟩ locally, while \DTLgclip sets ⟨*cmd*⟩ globally.

4 Strings

Strings are considered to be anything non-numerical. The datatool package loads the substr package, so you can use the commands defined in that package to determine if one string is contained in another string. In addition, the datatool provides the following macros:

\DTLsubstitute

\DTLsubstitute{⟨*cmd*⟩}{⟨*original*⟩}{⟨*replacement*⟩}

This replaces the first occurrence of ⟨*original*⟩ in ⟨*cmd*⟩ with ⟨*replacement*⟩. Note that ⟨*cmd*⟩ must be the name of a command. For example:

```
\def\mystr{abcdce}\DTLsubstitute{\mystr}{c}{z}\mystr
```

produces: abzdce.

\DTLsubstituteall

\DTLsubstituteall{⟨*cmd*⟩}{⟨*original*⟩}{⟨*replacement*⟩}

This replaces all occurrences of ⟨*original*⟩ in ⟨*cmd*⟩ with ⟨*replacement*⟩, where again, ⟨*cmd*⟩ must be the name of a command. For example:

```
\def\mystr{abcdce}\DTLsubstituteall{\mystr}{c}{z}\mystr
```

produces: abzdze.

\DTLsplitstring

\DTLsplitstring{⟨*string*⟩}{⟨*split text*⟩}{⟨*before cmd*⟩}{⟨*after cmd*⟩}

This splits ⟨*string*⟩ at the first occurrence of ⟨*split text*⟩ and stores the before part in the command ⟨*before cmd*⟩ and the after part in the command ⟨*after cmd*⟩. For example:

```
\DTLsplitstring{abcdce}{c}{\beforepart}{\afterpart}%
Before part: ``\beforepart''. After part: ``\afterpart''
```

produces: Before part: “ab”. After part: “dce”. Note that for `\DTLsplitstring`, ⟨*string*⟩ is not expanded, so

```
\def\mystr{abcdce}%
\DTLsplitstring{\mystr}{c}{\beforepart}{\afterpart}%
Before part: ``\beforepart''. After part: ``\afterpart''
```

produces: Before part: “abcdce”. After part: “”. If you want the string expanded, you will need to use `\expandafter`:

```
\def\mystr{abcdce}%
\expandafter\DTLsplitstring\expandafter
{\mystr}{c}{\beforepart}{\afterpart}%
Before part: ``\beforepart''. After part: ``\afterpart''
```

which produces: Before part: “ab”. After part: “dce”.

\DTLinitials

\DTLinitials{⟨*string*⟩}

This converts ⟨*string*⟩ (typically a name) into initials. For example:

```
\DTLinitials{Mary Ann}
```

produces: M.A. (including the final full stop). Note that

```
\DTLinitials{Mary-Ann}
```

produces: M.-A. (including the final full stop). Be careful if the initial letter has an accent. The accented letter needs to be placed in a group, if you want the initial to also have an accent, otherwise the accent command will be ignored. For example:

```
\DTLinitials{{\'E}lise Adams}
```

produces: É.A., whereas

```
\DTLinitials{\'Elise Adams}
```

produces: E.A. In fact, any command which appears at the start of the name that is not enclosed in a group will be ignored. For example:

```
\DTLinitials{\MakeUppercase{m}ary ann}
```

produces: m.a., whereas

```
\DTLinitials{{\MakeUppercase{m}}ary ann}
```

produces: M.a., but note that

```
\DTLinitials{\MakeUppercase{mary ann}}
```

produces: mary ann.

`\DTLstoreinitials`

`\DTLstoreinitials{`⟨*string*⟩`}{`⟨*cmd*⟩`}`

This converts ⟨*string*⟩ into initials and stores the result in ⟨*cmd*⟩ which must be a command name. The remarks about `\DTLinitials` also relate to `\DTLstoreinitials`. For example

```
\DTLstoreinitials{Marie-{\'E}lise del~Rosario}{\theInitials}\theInitials
```

produces: M.-É.d.R.

Both the above commands rely on the following to format the initials:

`\DTLafterinitials`

`\DTLafterinitials`

This indicates what to do at the end of the initials. This simply does a full stop by default.

`\DTLbetweeninitials`

`\DTLbetweeninitials`

This indicates what to do between initials. This does a full stop by default.

`\DTLinitialhyphen`

`\DTLinitialhyphen`

This indicates what to do at a hyphen. This simply does a hyphen by default, but can be redefined to do nothing to prevent the hyphen appearing in the initials.

`\DTLafterinitialbeforehyphen`

`\DTLafterinitialbeforehyphen`

This indicates what to do between an initial and a hyphen. This simply does a full stop by default.

For example

```
\renewcommand*{\DTLafterinitialbeforehyphen}{}%
\DTLinitials{Marie-{\'E}lise del~Rosario}
```

produces: M-É.d.R. whereas

```
\renewcommand*{\DTLafterinitialbeforehyphen}{}%
\renewcommand*{\DTLafterinitials}{}%
\renewcommand*{\DTLbetweeninitials}{}%
\renewcommand*{\DTLinitialhyphen}{}%
\DTLinitials{Marie-{\'E}lise del~Rosario}
```

produces: MÉdR

5 Databases

The datatool package provides a means of creating and loading databases. Once a database has been created (or loaded), it is possible to iterate through each row of data, to make it easier to perform repetitive actions, such as mail merging.

⚠ Whilst 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 uses a completely different method of storing the data to previous versions.[1] 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.

5.1 Creating a New Database

`\DTLnewdb`

`\DTLnewdb{`⟨*db name*⟩`}`

This command creates a new empty database called ⟨*db name*⟩. You can test if a database is empty using:

`\DTLifdbempty`

`\DTLifdbempty{`⟨*db name*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

If the database called ⟨*db name*⟩ is empty, do ⟨*true part*⟩, otherwise do ⟨*false part*⟩.

`\DTLrowcount`

`\DTLrowcount{`⟨*db name*⟩`}`

This command displays the number of rows in the database called ⟨*db name*⟩.

`\DTLnewrow`

`\DTLnewrow{`⟨*db name*⟩`}`

This command starts a new row in the database called ⟨*db name*⟩. This new row becomes the current row when adding new entries.

For example, the following creates an empty database called `mydata`:

```
\DTLnewdb{mydata}
```

[1] Thanks to Morten Høgholm for the suggestion.

The following tests if the database is empty:

```
\DTLifdbempty{mydata}{empty}{not empty}!
```

This produces: empty!

The following adds an empty row to the database, this is the first row of the database:

```
\DTLnewrow{mydata}
```

Note that even though the only row in the database is currently empty, the database is no longer considered to be empty:

```
\DTLifdbempty{mydata}{empty}{not empty}!
```

This now produces: not empty! The row count is given by

```
\DTLrowcount{mydata}
```

which produces: 1.

`\DTLnewdbentry`

\DTLnewdbentry{⟨*db name*⟩}{⟨*key*⟩}{⟨*value*⟩}

This creates a new entry with the identifier ⟨*key*⟩ whose value is ⟨*value*⟩ and adds it to the last row of the database called ⟨*db name*⟩. For example:

```
\DTLnewdbentry{mydata}{Surname}{Smith}
\DTLnewdbentry{mydata}{FirstName}{John}
```

Adds an entry with identifier `Surname` and value `Smith` to the last row of the database named `mydata`, and then adds an entry with identifier `FirstName` and value `John`. Note that the key should not contain any fragile commands. It is generally best to only use non-active characters in the key.

⚠ Note that database entries can't contain paragraph breaks as many of the macros used by datatool are short commands. If you do need a paragraph break in an entry, you can instead use the command:

`\DTLpar`

\DTLpar

For example:

```
\DTLnewdbentry{mydata}{Description}{First paragraph.\DTLpar
Second paragraph.}
```

`\DTLaddentryforrow`

\DTLaddentryforrow{⟨*db*⟩}{⟨*assign list*⟩}{⟨*condition*⟩}{⟨*key*⟩}{⟨*value*⟩}

This adds the entry with the key given by ⟨*key*⟩ and value given by ⟨*value*⟩ to the first row in the database ⟨*db*⟩ which satisfies the condition given by ⟨*condition*⟩. The ⟨*assign list*⟩ argument is the same as for `\DTLforeach` (described in subsection 5.4) and may be used to set the values which are to be tested in ⟨*condition*⟩ (where, again, ⟨*condition*⟩ is the same as for `\DTLforeach`). For example:

```
\DTLaddentryforrow{mydata}{\firstname=FirstName,\surname=Surname}%
{\DTLiseq{\firstname}{John}\and\DTLiseq{\surname}{Smith}}%
{Score}{75}
```

\DTLsetheader

\DTLsetheader{⟨*db*⟩}{⟨*key*⟩}{⟨*header*⟩}

This assigns a header for a given key in the database named ⟨*db*⟩. This is used by \DTLdisplaydb and \DTLdisplaylongdb in the header row (see subsection 5.3). If you don't assign a header, the header will be given by the key. For example:

```
\DTLsetheader{mydata}{Price}{Price (\$)}
```

5.2 Loading a Database from an External ASCII File

Instead of using the commands described in subsection 5.1 to create a new database, you can load a database from an external ASCII file using:

\DTLloaddb

\DTLloaddb[⟨*options*⟩]{⟨*db name*⟩}{⟨*filename*⟩}

This creates a new database called ⟨*db name*⟩, and fills it with the entries given in the file ⟨*filename*⟩. The filename may have a header row at the start of the file, which provides the ⟨*key*⟩ when creating a new database entry using \DTLnewdbentry. The optional argument ⟨*options*⟩ is a key=value list of options. Available options are:

noheader This is a boolean value and indicates if the file does not contain a header. If no value is supplied, true is assumed (i.e. the file doesn't contain a header row). If this option is omitted, it is assumed that the file contains a header row.

keys This is a comma-separated list of keys to use, where the keys are listed in the same order as the columns. If the file has a header, these keys will override the values given in the header row. If the file has no header row and no keys are supplied in ⟨*options*⟩, then the keys will be given by \dtldefaultkey⟨*n*⟩, where ⟨*n*⟩ is the column number and \dtldefaultkey defaults to "Column". Note that the list of keys must be delimited by braces since they contain commas. For example:

\dtldefaultkey

```
\DTLloaddb[noheader,keys={Temperature,Time,T2G}]{data}{data.csv}
```

headers This is a comma-separated list of headers. If not supplied, the header will be the same as that given in the header row, or the key if there is no header row. Note that the list of headers must be delimited by braces since they contain commas. For example:

```
\DTLloaddb[noheader,keys={Temperature,Time,T2G},%
headers={\shortstack{Incubation\\Temperature},%
\shortstack{Incubation\\Time},%
\shortstack{Time to\\Growth}}]{data}{data.csv}
```

By default, the entries in the database must be separated by a comma, and optionally delimited by the double quote character ("). The separator can be changed to a tab separator using the command:

\DTLsettabseparator

\DTLsettabseparator

To set the separator to a character other than a tab, you need to use

\DTLsetseparator

```
\DTLsetseparator{⟨character⟩}
```

The delimiter can be changed using

\DTLsetdelimiter

```
\DTLsetdelimiter{⟨character⟩}
```

For example, suppose you have a file called `mydata.csv` which contains the following:

```
FirstName,Surname,Score
John,"Smith, Jr",68
Jane,Brown,75
Andy,Brown,42
Z\"oe,Adams,52
```

then

```
\DTLloaddb{mydata}{mydata.csv}
```

is equivalent to:

```
\DTLnewdb{mydata}
\DTLnewrow{mydata}%
\DTLnewdbentry{mydata}{FirstName}{John}%
\DTLnewdbentry{mydata}{Surname}{Smith, Jr}%
\DTLnewdbentry{mydata}{Score}{68}%
\DTLnewrow{mydata}%
\DTLnewdbentry{mydata}{FirstName}{Jane}%
\DTLnewdbentry{mydata}{Surname}{Brown}%
\DTLnewdbentry{mydata}{Score}{75}%
\DTLnewrow{mydata}%
\DTLnewdbentry{mydata}{FirstName}{Andy}%
\DTLnewdbentry{mydata}{Surname}{Brown}%
\DTLnewdbentry{mydata}{Score}{42}%
\DTLnewrow{mydata}%
\DTLnewdbentry{mydata}{FirstName}{Z\"oe}%
\DTLnewdbentry{mydata}{Score}{52}%
\DTLnewdbentry{mydata}{Surname}{Adams}%
```

Note that the entry `Smith, Jr` had to be delimited in `mydata.csv` using the double quote character since it contained a comma which is used as the separator.

The file used in the above example contained a LaTeX command, namely `\"`. When using `\DTLloaddb` all the special characters that appear in the command retain their LaTeX meaning when the file is loaded. It is likely however that the data file may have been created by another application that is not TeX-aware, such as a spreadsheet application. For example, suppose you have a file called, say, `products.csv` which looks like:

```
Product,Cost
Fruit & Veg,$1.25
Stationary,$0.80
```

This file contains two of TEX's special characters, namely & and $. In this case, if you try to load the file using \DTLloaddb, you will encounter errors. Instead you can use:

\DTLloadrawdb

\DTLloadrawdb[⟨*options*⟩]{⟨*db name*⟩}{⟨*filename*⟩}

This is the same as \DTLloaddb except that it maps nine of the ten special characters onto commands which produce that symbol. The only character that retains its active state is the backslash character, so you will still need to check the file for backslash characters. The mappings used are listed in Table 1. So using the file products.csv, as described above,

```
\DTLloadrawdb{mydata}{products.csv}
```

is equivalent to:

```
\DTLnewdb{mydata}
\DTLnewrow{mydata}%
\DTLnewdbentry{mydata}{Product}{Fruit \& Veg}%
\DTLnewdbentry{mydata}{Cost}{\$1.25}%
\DTLnewrow{mydata}%
\DTLnewdbentry{mydata}{Product}{Stationary}%
\DTLnewdbentry{mydata}{Cost}{\$0.80}%
```

Table 1: Special character mappings used by \DTLloadrawdb (note that the backslash retains its active state)

Character	Mapping
%	\%
$	\$
&	\&
#	\#
_	_
{	\{
}	\}
~	\textasciitilde
^	\textasciicircum

It may be that there are other characters that require mapping. For example, the file products.csv may instead look like:

```
Product,Cost
Fruit & Veg,£1.25
Stationary,£0.80
```

The pound character is not an internationally standard keyboard character, and does not generally achieve the desired effect when used in a LATEX document. It will therefore be necessary to convert this symbol to an appropriate control sequence. This can be done using the command:

\DTLrawmap

\DTLrawmap{⟨*string*⟩}{⟨*replacement*⟩}

For example:

```
\DTLrawmap{£}{\pounds}
```

will replace all occurrences[2] of £ with \pounds. Naturally, the mappings must be set *prior* to loading the data with \DTLloadrawdb.

⚠ Note that the warning in the previous section about no paragraph breaks in an entry also applies to entries loaded from a database. If you do need a paragraph break, use \DTLpar instead of \par, but remember that each row of data in an external data file must not have a line break.

5.3 Displaying the Contents of a Database

Once you have created a database, either loading it from an external file, as described in subsection 5.2, or using the commands described in subsection 5.1, you can display the entire database in a tabular or longtable environment.

\DTLdisplaydb

\DTLdisplaydb{⟨*db*⟩}

This displays the database given by ⟨*db*⟩ in a tabular environment. The first row displays the headers for the database in bold, the subsequent rows display the values for each key in each row of the database.

\DTLdisplaylongdb

\DTLdisplaylongdb[⟨*options*⟩]{⟨*db*⟩}

This is like \DTLdisplaydb except that it uses the longtable environment instead of the tabular environment. Note that if you use this command, you must load longtable, as it is not loaded by datatool. The optional argument ⟨*options*⟩ is a comma-separated list of key=value pairs. The following keys are available:

caption The caption for the longtable.

contcaption The continuation caption.

shortcaption The caption to be used in the list of figures.

label The label for this table.

foot The longtable's foot.

lastfoot The foot for the last page of the longtable.

For example, suppose I have a database called iris, then I can display the contents in a longtable using:

```
\DTLdisplaylongdb[%
caption={Iris Data},%
label={tab:iris},%
```

[2] when it is loaded into the LaTeX database, it does not modify the data file!

```
contcaption={Iris Data (continued)},%
foot={\em Continued overleaf},%
lastfoot={}%
]{iris}
```

I can then reference the table using `\ref{tab:iris}`.

See the longtable documentation for details on how to change the longtable settings, such as how to change the table so that it is left aligned instead of centred on the page.

Note that if you want more control over the way the data is displayed, for example, you want to filter rows or columns, you will need to use `\DTLforeach`, described in subsection 5.4.

Example 1 (Displaying the Contents of a Database)

Suppose I have a file called `t2g.csv` that contains the following:

```
40,120,40
40,90,60
35,180,20
55,190,40
```

This represents time to growth data, where the first column is the incubation temperature, the second column is the incubation time and the third column is the time to growth. This file has no header row, so when it is loaded, the noheaders option is required. Note that `\DTLdisplaydb` only puts the data in a tabular environment, so `\DTLdisplaydb` needs to be put in a table environment with a caption to make it a float.

First load the data base, setting the keys and headers:

```
\DTLloaddb[noheader,%
keys={Temperature,Time,T2G},%
headers={\shortstack{Incubation\\Temperature},%
\shortstack{Incubation\\Time},\shortstack{Time to\\Growth}}%
]{t2g}{t2g.csv}
```

Now display the data in a table:

```
\begin{table}[htbp]
\caption{Time to Growth Data}
\vspace{\baselineskip}
\centering
\DTLdisplaydb{t2g}
\end{table}
```

The result is shown in Table 2.

Each column in the database has an associated data type which indicates what type of data is in that column. This may be one of: string, integer, real number or currency. If a column contains more than one type, the data type is determined as follows:

- If the column contains at least one string, then the column data type is string.

Table 2: Time to Growth Data

Incubation Temperature	Incubation Time	Time to Growth
40	120	40
40	90	60
35	180	20
55	190	40

- If the column doesn't contain a string, but contains at least one currency, then the column data type is currency.
- If the column contains only real numbers and integers, the column data type is real number.
- The column data type is integer if the column only contains integers.

The column data type is updated whenever a new entry is added to the database. Note that the column data type is not adjusted when an entry is removed from the database.

The column alignments used by `\DTLdisplaydb` are given by:

`\dtlstringalign`

```
\dtlstringalign
```

The string alignment defaults to `l` (left aligned).

`\dtlintalign`

```
\dtlintalign
```

The integer alignment defaults to `r` (right aligned).

`\dtlrealalign`

```
\dtlrealalign
```

The alignment for real numbers defaults to `r` (right aligned).

`\dtlcurrencyalign`

```
\dtlcurrencyalign
```

The currency alignment defaults to `r` (right aligned).

You can redefine these to change the column alignments. For example, if you want columns containing strings to have the alignment `p{2in}`, then you can redefine `\dtlstringalign` as follows:

```
\renewcommand{\dtlstringalign}{p{2in}}
```

⚠ You can't use siunitx's `S` column alignment with either `\DTLdisplaydb` or `\DTLdisplaylongdb`. Instead, you will need to use `\DTLforeach`. The siunitx documentation provides an example.

`\dtlbeforecols` `\dtlbetweencols` `\dtlaftercols`

In addition to the `\dtl`⟨*type*⟩`align` commands above, you can also modify the tabular column styles by redefining `\dtlbeforecols`, `\dtlbetweencols` and `\dtlaftercols`. For example, to place a vertical line before the start of the first

column and after the last column, do:

```
\renewcommand{\dtlbeforecols}{|}
\renewcommand{\dtlaftercols}{|}
```

If you additionally want vertical lines between columns, do:

```
\renewcommand{\dtlbetweencols}{|}
```

Limited modifications can be made to the way the data is displayed with `\DTLdisplaydb` and `\DTLdisplaylongdb`. The commands controlling the formatting are described below. If a more complicated layout is required, you will need to use `\DTLforeach` described in subsection 5.4.

`\dtlheaderformat`

`\dtlheaderformat{`⟨*header*⟩`}`

This indicates how to format a column header, where the header is given by ⟨*header*⟩. This defaults to `\null\hfil\textbf{`⟨*header*⟩`}\hfil\null`.

`\dtlstringformat`

`\dtlstringformat{`⟨*text*⟩`}`

This specifies how to format each entry in the columns that contain strings. This defaults to just displaying ⟨*text*⟩.

`\dtlintformat`

`\dtlintformat{`⟨*text*⟩`}`

This specifies how to format each entry in the columns that contain only integers. This defaults to just displaying ⟨*text*⟩.

`\dtlrealformat`

`\dtlrealformat{`⟨*text*⟩`}`

This specifies how to format each entry in the columns that contain only real numbers or a mixture of real numbers and integers. This defaults to just displaying ⟨*text*⟩.

`\dtlcurrencyformat`

`\dtlcurrencyformat{`⟨*text*⟩`}`

This specifies how to format each entry in the columns that contain only currency or currency mixed with real numbers and/or integers. This defaults to just displaying ⟨*text*⟩.

`\dtldisplaystarttab`

`\dtldisplaystarttab`

This is a hook to add something at the beginning of the tabular environment. This defaults to nothing.

`\dtldisplayendtab`

`\dtldisplayendtab`

This is a hook to add something at the end of the tabular environment. This defaults to nothing.

`\dtldisplayafterhead`

```
\dtldisplayafterhead
```

This is a hook to add something after the header row, before the first row of data. This defaults to nothing.

`\dtldisplaystartrow`

```
\dtldisplaystartrow
```

This is a hook to add something at the start of each row, but not including the header row or the first row of data. This defaults to nothing.

Example 2 (Balance Sheet)

Suppose you have a file called `balance.csv` that contains the following:

```
Description,In,Out,Balance
Travel expenses,,230,-230
Conference fees,,400,-630
Grant,700,,70
Train fare,,70,0
```

The data can be loaded using:

```
\DTLloaddb[headers={Description,In (\pounds),Out (pounds),Balance
(pounds)}]{balance}{balance.csv}
```

Suppose I want negative numbers to be displayed in red. I can do this by redefining `\dtlrealformat` to check if the entry is negative. For example:

```
\begin{table}[htbp]
\caption{Balance Sheet}
\renewcommand*{\dtlrealformat}[1]{\DTLiflt{#1}{0}{\color{red}}{}#1}
\vspace{\baselineskip}
\centering
\DTLdisplaydb{balance}
\end{table}
```

This produces Table 3.

Table 3: Balance Sheet

Description	**In (£)**	**Out (£)**	**Balance (£)**
Travel expenses		230.00	-230.00
Conference fees		400.00	-630.00
Grant	700.00		70.00
Train Fare		70.00	0.00

5.4 Iterating Through a Database

Once you have created a database, either loading it from an external file, as described in subsection 5.2, or using the commands described in subsection 5.1,

you can then iterate through each row of the database and access elements in that row.

\DTLforeach

\DTLforeach[⟨*condition*⟩]{⟨*db name*⟩}{⟨*assign list*⟩}{⟨*text*⟩}

\DTLforeach*

\DTLforeach*[⟨*condition*⟩]{⟨*db name*⟩}{⟨*assign list*⟩}{⟨*text*⟩}

This will iterate through each row of the database called ⟨*db name*⟩, applying ⟨*text*⟩ to each row of the database where ⟨*condition*⟩ is met. The argument ⟨*assign list*⟩ is a comma separated list of ⟨*cmd*⟩=⟨*key*⟩ pairs. At the start of each row, each command ⟨*cmd*⟩ in ⟨*assign list*⟩ will be set to the value of the entry given by ⟨*key*⟩. These commands may then be used in ⟨*text*⟩.

⚠ Note that this assignment is done globally to ensure that `\DLTforeach` works correctly in a tabular environment. Since you may want to use the same set of commands in a later `\DTLforeach`, the commands are not checked to determine if they already exist. It is therefore important that you check you are not using an existing command whose value should not be changed.

The optional argument ⟨*condition*⟩ is a condition in the form allowed by `\ifthenelse`. This includes the commands provided by the ifthen package (such as `\not`, `\and`, `\or`), as well as the commands described in subsection 2.2. The default value of ⟨*condition*⟩ is `\boolean{true}`.

The starred version `\DTLforeach*` is a read-only version. If you want to modify the database using any of the commands described in subsection 5.6, you must use the unstarred version. The starred version is faster.

Example 3 (Student scores)

Suppose you have a data file called `studentscores.csv` that contains the following:

```
FirstName,Surname,StudentNo,Score
John,"Smith, Jr",102689,68
Jane,Brown,102647,75
Andy,Brown,103569,42
Z\"oe,Adams,105987,52
Roger,Brady,106872,58
Clare,Verdon,104356,45
```

and you load the data into a database called `scores` using:

```
\DTLloaddb{scores}{studentscores.csv}
```

you can then display the database in a table as follows:

```
\begin{table}[htbp]
\caption{Student scores}
\centering
\begin{tabular}{llr}
\bfseries First Name &
\bfseries Surname &
\bfseries Score (\%)%
\DTLforeach{scores}{%
```

```
\firstname=FirstName,\surname=Surname,\score=Score}{%
\\
\firstname & \surname & \score}
\end{tabular}
\end{table}
```

This produces Table 4. (Note that since I didn't need the student registration number, I didn't bother to assign a command to the key `StudentNo`.)

Table 4: Student scores

First Name	**Surname**	**Score (%)**
John	Smith, Jr	68
Jane	Brown	75
Andy	Brown	42
Zöe	Adams	52
Roger	Brady	58
Clare	Verdon	45

The macro `\DTLforeach` may be nested up to three times. Each level uses the corresponding counters: `DTLrowi`, `DTLrowii` and `DTLrowiii` which keep track of the current row.

⚠ Note that these counters are only incremented when ⟨*condition*⟩ is satisfied, therefore they will not have the correct value in ⟨*condition*⟩. These counters are incremented using `\refstepcounter` before the start of ⟨*text*⟩, so they may be referenced using `\label`, however remember that `\label` references the last counter to be incremented using `\refstepcounter` *in the current scope*. The `\label` should therefore be the first command in ⟨*text*⟩ to ensure that it references the current row counter.

`\DTLcurrentindex`

`\DTLcurrentindex`

At the start of each iteration in `\DTLforeach`, `\DTLcurrentindex` is set to the arabic value of the current row counter. Note that this is only set after the condition is tested, so it should only be used in the body of `\DTLforeach` not in the condition. It is also only set locally, so if you use it in a tabular environment, it can only be used before the first instance of `\\` or `&` in the current iteration.

Within the body of `\DTLforeach` (i.e. within ⟨*text*⟩) the following conditionals may be used:

`\DTLiffirstrow`

`\DTLiffirstrow{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

If the current row is the first row, then do ⟨*true part*⟩, otherwise do ⟨*false part*⟩.

`\DTLifoddrow`

`\DTLifoddrow{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

If the current row number is an odd number, then do ⟨*true part*⟩, otherwise do ⟨*false part*⟩.

\DTLsavelastrowcount

```
\DTLsavelastrowcount{⟨cmd⟩}
```

This command will store the value of the row counter used in the last occurrence of \DTLforeach in the control sequence ⟨*cmd*⟩.

\DTLforeachkeyinrow

```
\DTLforeachkeyinrow{⟨cmd⟩}{⟨text⟩}
```

This iterates through each key in the current row, (globally) assigns ⟨*cmd*⟩ to the value of that key, and does ⟨*text*⟩ (⟨*cmd*⟩ must be a control sequence and may be used in ⟨*text*⟩). This command may only be used in the body of \DTLforeach. At each iteration, \DTLforeachkeyinrow sets \dtlkey to the current key, \dtlcol to the current column index, \dtltype to the data type for the current column, and \dtlheader to the header for the current column. Note that \dtltype corresponds to the column type but if the entries in the column have mixed types, it may not correspond to the type of the current entry.

\dtlforeachkey

```
\dtlforeachkey(⟨key cs⟩,⟨col cs⟩,⟨type cs⟩,⟨header cs⟩)\in{⟨db⟩}\do{⟨body⟩}
```

This iterates through all the keys in the database given by ⟨*db*⟩. In each iteration, ⟨*key cs*⟩ is set to the key, ⟨*col cs*⟩ is set to the column index, ⟨*type cs*⟩ is set to the data type, ⟨*header cs*⟩ is set to the header for that column, and then ⟨*body*⟩ is done. Note that ⟨*key cs*⟩, ⟨*col cs*⟩, ⟨*type cs*⟩ and ⟨*header cs*⟩ must all be control sequences. No check is performed to determine if that control sequence already exists, and the control sequences are defined globally (since it's likely that \dtlforeachkey may be used within a tabular environment), so you need to make sure you don't override an existing command of the same name. The data type may have the following values: 0 (string), 1 (integer), 2 (real number), 3 (currency) or it will be empty if the column has no entries.

\dtlforcolumn

```
\dtlforcolumn{⟨cs⟩}{⟨db⟩}{⟨key⟩}{⟨body⟩}
```

This iterates through the column given by ⟨*key*⟩ in the database given by ⟨*db*⟩ and applies ⟨*body*⟩. In each iteration, ⟨*cs*⟩ (which must be a control sequence) is set to the current element in the column and may be used in ⟨*body*⟩.

\dtlbreak

```
\dtlbreak
```

You can break out of most of the loops provided by datatool using \dtlbreak. Note, however, that it doesn't break the loop until the end of the current iteration. There is no provision for a next or continue style command.

Additional loop commands provided by datatool are described in subsection 10.7.

Example 4 (Student Scores—Labelling)

In the previous example, the student scores, stored in the database scores were placed in a table. In this example the table will be modified slightly to number each student according to the row. Suppose I also want to identify which

row Jane Brown is in, and reference it in the text. The easiest way to do this is to construct a label on each row which uniquely identifies that student. The label can't simply be constructed from the surname, as there are two students with the same surname. In order to create a unique label, I can either construct a label from both the surname and the first name, or I can use the student's registration number, or I can use the student's score, since all the scores are unique. The former method will cause a problem since one of the names (Zöe) contains an accent command. Although the registration numbers are all unique, they are not particularly memorable, so I shall instead use the scores.

```
\begin{table}[htbp]
\caption{Student scores}
\centering
\begin{tabular}{cllc}
\bfseries Row &
\bfseries First Name &
\bfseries Surname &
\bfseries Score (\%)%
\DTLforeach*{scores}{%
\firstname=FirstName,\surname=Surname,\score=Score}{%
\label{row:\score}\\\theDTLrowi &
\firstname & \surname & \score}%
\end{tabular}
\end{table}

Jane Brown scored the highest (75\%), her score can be seen on
row~\ref{row:75}.
```

This produces Table 5 and the following text: Jane Brown scored the highest (75%), her score can be seen on row 2.

Notes:

- the `\label` command is placed before `\\` to ensure that it is in the same scope as the command `\refstepcounter{DTLrowi}`.
- To avoid unwanted spaces the end of line characters are commented out with the percent (`%`) symbol.

Table 5: Student scores

Row	**First Name**	**Surname**	**Score (%)**
1	John	Smith, Jr	68
2	Jane	Brown	75
3	Andy	Brown	42
4	Zöe	Adams	52
5	Roger	Brady	58
6	Clare	Verdon	45

Example 5 (Filtering Rows)

As mentioned earlier, the optional argument ⟨*condition*⟩ of `\DTLforeach` provides a means to exclude certain rows. This example uses the database defined in example 3, but only displays the information for students whose marks are above 60. At the end of the table, `\DTLsavelastrowcount` is used to store the number of rows in the table. (Note that `\DTLsavelastrowcount` is outside of `\DTLforeach`.)

```
\begin{table}[htbp]
\caption{Top student scores}
\centering
\begin{tabular}{llr}
\bfseries First Name &
\bfseries Surname &
\bfseries Score (\%)%
\DTLforeach*[\DTLisgt{\score}{60}]{scores}{%
\firstname=FirstName,\surname=Surname,\score=Score}{%
\\
\firstname & \surname & \score}
\end{tabular}

\DTLsavelastrowcount{\n}%
\n\ students scored above 60\%.
\end{table}
```

This produces Table 6. Note that in this example, I could have specified the condition as `\score>60` since all the scores are integers, however, as it's possible that an entry may feasibly have a decimal score I have used `\DTLisgt` instead.

Table 6: Top student scores

First Name	**Surname**	**Score (%)**
John	Smith, Jr	68
Jane	Brown	75

2 students scored above 60%.

Suppose now, I only want to display the scores for students whose surname begins with 'B'. I can do this as follows:

```
\begin{table}[htbp]
\caption{Student scores (B)}
\centering
\begin{tabular}{llr}
\bfseries First Name &
\bfseries Surname &
\bfseries Score (\%)%
\DTLforeach*[\DTLisopenbetween{\surname}{B}{C}]{scores}{%
\firstname=FirstName,\surname=Surname,\score=Score}{%
\\
\firstname & \surname & \score}
\end{tabular}
\end{table}
```

This produces Table 7.

Table 7: Student scores (B)

First Name	Surname	Score (%)
Jane	Brown	75
Andy	Brown	42
Roger	Brady	58

Example 6 (Breaking Out of a Loop)

Suppose I only want to display the first three rows of a database. I could do:[3]

```
\DTLforeach*[\value{DTLrowi}<3]{scores}%
{\firstname=FirstName,\surname=Surname,\score=Score}{%
\\\firstname & \surname & \score
}
```

However, this isn't very efficient, as it still has to iterate through the entire database, checking if the condition is met. If the database has over 100 entries, this will slow the time taken to create the table. It would therefore be much more efficient to break out of the loop when row count exceeds 3:

```
\begin{table}[htbp]
\caption{First Three Rows}
\centering
\vskip\baselineskip
\begin{tabular}{llr}
\bfseries First Name & \bfseries Surname & \bfseries Score (\%)%
\DTLforeach*{scores}%
{\firstname=FirstName,\surname=Surname,\score=Score}{%
\ifthenelse{\DTLcurrentindex=3}{\dtlbreak}{}%
\\\firstname & \surname & \score
}%
\end{tabular}
\end{table}
```

This produces Table 8. Note that the loop is not broken until the end of the current iteration, so even though `\dtlbreak` occurs at the start of the third row, the loop isn't finished until the third row is completed. (Recall that `\DTLcurrentindex` must be used before the first instance of `\\` or `&`.) Alternatively, you can use `DTLrowi` instead:

```
\DTLforeach{scores}%
{\firstname=FirstName,\surname=Surname,\score=Score}{%
\\\firstname & \surname & \score
\ifthenelse{\value{DTLrowi}=3}{\dtlbreak}{}%
}%
```

---

[3] Recall that `DTLrowi` is incremented after the condition is tested, so it will be out by 1 when the condition is tested which is why `<3` is used instead of `<4`.

Table 8: First Three Rows

First Name	**Surname**	**Score (%)**
John	Smith, Jr	68
Jane	Brown	75
Andy	Brown	42

Example 7 (Stripy Tables)

This example uses the same database as in the previous examples. It requires the colortbl package, which provides the command `\rowcolor`. The command `\DTLifoddrow` is used to produce a striped table.

```
\begin{table}[htbp]
\caption{A stripy table}\label{tab:stripy}
\centering
\begin{tabular}{llc}
\bfseries First Name &
\bfseries Surname &
\bfseries Score (\%)%
\DTLforeach*{scores}{%
\firstname=FirstName,\surname=Surname,\score=Score}{%
\\\DTLifoddrow{\rowcolor{blue}}{\rowcolor{green}}%
\firstname & \surname & \score}%
\end{tabular}
\end{table}
```

This produces Table 9.

Table 9: A stripy table

First Name	**Surname**	**Score (%)**
John	Smith, Jr	68
Jane	Brown	75
Andy	Brown	42
Zöe	Adams	52
Roger	Brady	58
Clare	Verdon	45

Example 8 (Two Database Rows per Tabular Row)

In order to save space, you may want two database rows per tabular row, when displaying a database in a tabular environment. This can be accomplished using `\DTLifoddrow`. For example

```
\begin{table}[htbp]
\caption{Two database rows per tabular row}
\centering
```

```
\begin{tabular}{llcllc}
\bfseries First Name &
\bfseries Surname &
\bfseries Score (\%) &
\bfseries First Name &
\bfseries Surname &
\bfseries Score (\%)%
\DTLforeach*{scores}{\firstname=FirstName,\surname=Surname,\score=Score}{%
\DTLifoddrow{\\}{&}%
\firstname & \surname & \score}%
\end{tabular}
\end{table}
```

produces Table 10

Table 10: Two database rows per tabular row

First Name	**Surname**	**Score (%)**	**First Name**	**Surname**	**Score (%)**
John	Smith, Jr	68	Jane	Brown	75
Andy	Brown	42	Zöe	Adams	52
Roger	Brady	58	Clare	Verdon	45

Example 9 (Iterating Through Keys in a Row)

Suppose you have lots of columns in your database, and you want to display them all without having to set a variable for each column. You can leave the assignment list in `\DTLforeach` blank, and iterate through the keys using `\DTLforeachkeyinrow`. For example:

```
\begin{table}[htbp]
\caption{Student Scores (Iterating Through Keys)}
\vskip\baselineskip
\centering
\begin{tabular}{llll}
\bfseries First Name & \bfseries Surname &
\bfseries Registration No. &
\bfseries Score (\%)%
\DTLforeach*{scores}{}{%
\\\gdef\doamp{\gdef\doamp{&}}%
\DTLforeachkeyinrow{\thisValue}{\doamp\thisValue}}%
\end{tabular}
\end{table}
```

This produces Table 11.

Note that the `&` must be between columns, so I have defined a command called `\doamp` that on first use redefines itself to do `&`. So, for each row, at the start of the key iteration, `\doamp` does nothing, and on subsequent iterations it does `&`. This ensures that the correct number of `&`s are used. Since each cell in the `tabular` environment is scoped, `\gdef` is needed instead of `\def`.

In the above, I needed to know how many columns are in the database, and the order that the headings should appear. If you are unsure, you can use

Table 11: Student Scores (Iterating Through Keys)

First Name	**Surname**	**Registration No.**	**Score (%)**
John	Smith, Jr	102689	68
Jane	Brown	102647	75
Andy	Brown	103569	42
Zöe	Adams	105987	52
Roger	Brady	106872	58
Clare	Verdon	104356	45

`\dtlforeachkey` to determine the number of columns and to display the header row. For example:

```
\begin{table}[htbp]
\caption{Student Scores}
\vskip\baselineskip
\centering
% Work out the column alignments.
\def\colalign{}%
\dtlforeachkey(\theKey,\theCol,\theType,\theHead)\in{scores}\do
{\edef\colalign{\colalign l}}%
% Begin the tabular environment.
\edef\dobegintabular{\noexpand\begin{tabular}{\colalign}}%
\dobegintabular
% Do the header row.
\gdef\doamp{\gdef\doamp{&}}%
\dtlforeachkey(\theKey,\theCol,\theType,\theHead)\in{scores}\do
{\doamp\bfseries \theHead}%
% Iterate through the data.
\DTLforeach*{scores}{}{%
\\\gdef\doamp{\gdef\doamp{&}}%
\DTLforeachkeyinrow{\thisValue}{\doamp\thisValue}}%
\end{tabular}
\end{table}
```

Table 12: Student Scores (Using `\dtlforeachkey` and `\DTLforeachkeyinrow`)

FirstName	**Surname**	**StudentNo**	**Score**
John	Smith, Jr	102689	68
Jane	Brown	102647	75
Andy	Brown	103569	42
Zöe	Adams	105987	52
Roger	Brady	106872	58
Clare	Verdon	104356	45

Notes:

- In order to determine the column alignment for the tabular environment, I first define `\colalign` to nothing, and then I iterate through the keys appending `l` to `\colalign`. Since `\colalign` only contains alphabetical

characters, I can just use \edef for this. I could modify this to check the data type and, say, use l (left alignment) for columns containing strings and c (centred) for the other columns:

```
\dtlforeachkey(\theKey,\theCol,\theType,\theHead)\in{scores}\do
{\ifnum\theType=0\relax
   \edef\colalign{\colalign l}% column contains strings
 \else
   \edef\colalign{\colalign c}% column contains numerical values
 \fi
}%
```

- To ensure \colalign gets correct expanded when passed to the tabular environment I temporarily define \dobegintabular to the code required to start the tabular environment:

  ```
  \edef\dobegintabular{\noexpand\begin{tabular}{\colalign}}%
  ```

  This sets \dobegintabular to \begin{tabular}{llll}. After defining \dobegintabular, I then need to use it.

- As before, I use \doamp to put the ampersands between columns.

- Recall that I can set the headers using \DTLsetheader or using the headers key when loading the data from an external file. For example:

  ```
  \DTLsetheaders{scores}{FirstName}{First Name}
  \DTLsetheaders{scores}{Score}{Score (\%)}
  ```

Recall that \DTLforeachkeyinrow sets \dtlkey to the current key. This can be used to filter out columns. Alternatively, if you know the column index, you can test \dtlcol instead. The following code modifies the above example so that it filters out the column whose key is StudentNo:

```
\begin{table}[htbp]
\caption{Student Scores (Filtering Out a Column)}
\vskip\baselineskip
\centering
\def\colalign{}%
\dtlforeachkey(\theKey,\theCol,\theType,\theHead)\in{scores}\do
{\DTLifeq{\theKey}{StudentNo}{}{\edef\colalign{\colalign l}}}%
\edef\dobegintabular{\noexpand\begin{tabular}{\colalign}}%
\dobegintabular
\gdef\doamp{\gdef\doamp{&}}%
\dtlforeachkey(\theKey,\theCol,\theType,\theHead)\in{scores}\do
{\DTLifeq{\theKey}{StudentNo}{}{\doamp\bfseries \theHead}}%
\DTLforeach*{scores}{}{%
\\\gdef\doamp{\gdef\doamp{&}}%
\DTLforeachkeyinrow{\thisValue}{%
  \DTLifeq{\dtlkey}{StudentNo}{}{\doamp\thisValue}}}%
\end{tabular}
\end{table}
```

The result is shown in Table 13.

Table 13: Student Scores (Filtering Out a Column)

FirstName	**Surname**	**Score**
John	Smith, Jr	68
Jane	Brown	75
Andy	Brown	42
Zöe	Adams	52
Roger	Brady	58
Clare	Verdon	45

Example 10 (Nested `\DTLforeach`)

In this example I have a CSV file called `index.csv` which contains:

```
File,Temperature,NaCl,pH
exp25a.csv,25,4.7,0.5
exp25b.csv,25,4.8,1.5
exp30a.csv,30,5.12,4.5
```

The first column of this file contains the name of another CSV file which has the results of a time to growth experiment performed at the given incubation temperature, salt concentration and pH. The file `exp25a.csv` contains the following:

```
Time,Log Count
0,3.75
23,3.9
45,4.0
```

The file `exp25b.csv` contains the following:

```
Time,Log Count
0,3.6
60,3.8
120,4.0
```

The file `exp30a.csv` contains the following:

```
Time,Log Count
0,3.73
23,3.67
60,4.9
```

Suppose I now want to iterate through `index.csv`, load the given file, and create a table for that data. I can do this using nested `\DTLforeach` as follows:

```
 % load index data file
\DTLloaddb{index}{index.csv}

 % iterate through index database
\DTLforeach{index}{\theFile=File,\theTemp=Temperature,%
\theNaCl=NaCl,\thepH=pH}{%
```

```
 % load results file into database of the same name
\DTLloaddb{\theFile}{\theFile}%
 % Create a table
\begin{table}[htbp]
\caption{Temperature = \theTemp, NaCl = \theNaCl,
pH = \thepH}\label{tab:\theFile}
\centering
\begin{tabular}{rl}
\bfseries Time & \bfseries Log Count
\DTLforeach{\theFile}{\theTime=Time,\theLogCount=Log Count}{%
\\\theTime & \theLogCount}%
\end{tabular}
\end{table}
}
```

This creates Table 14 to Table 16. (Note that each table is given a label that is based on the database name, to ensure that it is unique.)

Table 14: Temperature = 25, NaCl = 4.7, pH = 0.5

Time	**Log Count**
0	3.75
23	3.9
45	4.0

Table 15: Temperature = 25, NaCl = 4.8, pH = 1.5

Time	**Log Count**
0	3.6
60	3.8
120	4.0

Table 16: Temperature = 30, NaCl = 5.12, pH = 4.5

Time	**Log Count**
0	3.73
23	3.67
60	4.9

Example 11 (Dynamically Allocating Field Name)

(This example was suggested by Bill Hobbs.) Suppose you have a directory containing members of multiple clubs. The CSV file (say, `clubs.csv`) may look something like:

```
First Name,Surname,Rockin,Single
John,"Smith, Jr",member,
Jane,Brown,,friend
```

```
Andy,Brown,friend,member
Z\"oe,Adams,member,member
Roger,Brady,friend,friend
Clare,Verdon,member,
```

(Blank entries indicate that the person is not a member of that club.) The data can be loaded as follows:

```
\DTLloaddb{clubs}{clubs.csv}
```

Suppose at the beginning of your document you have specified which club you are interested in (`Rockin` or `Single`) and store it in `\DIdent`:

```
\newcommand{\DIdent}{Rockin}
```

You can now display the members for this particular club as follows:

```
\begin{table}[htbp]
\caption{Club Membership}
\vskip\baselineskip
\centering
\begin{tabular}{lll}
\bfseries First Name & \bfseries Surname & \bfseries Status
\DTLforeach*[\not\DTLiseq{\status}{}]{clubs}
{\firstname=First Name,\surname=Surname,\status=\DIdent}{%
\\\firstname & \surname & \status
}%
\end{tabular}
\end{table}
```

The result is shown in Table 17.

Table 17: Club Membership

First Name	**Surname**	**Status**
John	Smith, Jr	member
Andy	Brown	friend
Zöe	Adams	member
Roger	Brady	friend
Clare	Verdon	member

5.5 Null Values

If a database is created using `\DTLnewdb`, `\DTLnewrow` and `\DTLnewdbentry` (rather than loading it from an ASCII file), it is possible for some of the entries to have null values when a value is not assigned to a given key for a given row. (Note that a null value is not the same as an empty value.)

When you iterate through the database using `\DTLforeach` (described in subsection 5.4), if an entry is missing for a given row, the associated command given in the ⟨*values*⟩ argument will be set to a null value. This value depends on the data type associated with the given key.

\DTLstringnull

\DTLstringnull

This is the null value for a string.

\DTLnumbernull

\DTLnumbernull

This is the null value for a number.

\DTLifnull

\DTLifnull{⟨*cmd*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

This checks if ⟨*cmd*⟩ is null where ⟨*cmd*⟩ is a command name, if it is, then ⟨*true part*⟩ is done, otherwise ⟨*false part*⟩ is done. This macro is illustrated in example 12 below.

Example 12 (Null Values)

Consider the following (which creates a database called emailDB):

```
\DTLnewdb{emailDB}
\DTLnewrow{emailDB}
\DTLnewdbentry{emailDB}{Surname}{Jones}
\DTLnewdbentry{emailDB}{FirstName}{Mary}
\DTLnewdbentry{emailDB}{Email1}{mj@my.uni.ac.uk}
\DTLnewdbentry{emailDB}{Email2}{mj@somewhere.com}
\DTLnewrow{emailDB}
\DTLnewdbentry{emailDB}{Surname}{Smith}
\DTLnewdbentry{emailDB}{FirstName}{Adam}
\DTLnewdbentry{emailDB}{Email1}{as@my.uni.ac.uk}
\DTLnewdbentry{emailDB}{RegNum}{12345}
```

In the above example, the first row of the database contains an entry with the key Email2, but the second row doesn't. Whereas the second row contains an entry with the key RegNum, but the first row doesn't.

The following code puts the information in a tabular environment:

```
\begin{tabular}{lllll}
\bfseries First Name &
\bfseries Surname &
\bfseries Email 1 &
\bfseries Email 2 &
\bfseries Reg Num%
\DTLforeach{emailDB}{\firstname=FirstName,\surname=Surname,%
\emailI=Email1,\emailII=Email2,\regnum=RegNum}{%
\\\firstname & \surname & \emailI & \emailII & \regnum}%
\end{tabular}
```

This produces the following:

First Name	**Surname**	**Email 1**	**Email 2**	**Reg Num**
Mary	Jones	mj@my.uni.ac.uk	mj@somewhere.com	0
Adam	Smith	as@my.uni.ac.uk	NULL	12345

Note that on the first row of data, the registration number appears as 0, while on the next row, the second email address appears as NULL. The datatool package has identified the key RegNum for this database as a numerical key, since all elements in the database with that key are numerical, whereas it has identified the key Email2 as a string, since there is at least one element in this database with that key that is a string. Null numerical values are set to \DTLnumbernull (0), and null strings are set to \DTLstringnull (NULL).

The following code checks each value to determine whether it is null using \DTLifnull. If it is, the text *Missing* is inserted, otherwise the value itself is used:

```
\begin{tabular}{lllll}
\bfseries First Name &
\bfseries Surname &
\bfseries Email 1 &
\bfseries Email 2 &
\bfseries Reg Num%
\DTLforeach{emailDB}{\firstname=FirstName,\surname=Surname,%
\emailI=Email1,\emailII=Email2,\regnum=RegNum}{%
\\\DTLifnull{\firstname}{\emph{Missing}}{\firstname} &
\DTLifnull{\surname}{\emph{Missing}}{\surname} &
\DTLifnull{\emailI}{\emph{Missing}}{\emailI} &
\DTLifnull{\emailII}{\emph{Missing}}{\emailII} &
\DTLifnull{\regnum}{\emph{Missing}}{\regnum}}%
\end{tabular}
```

This produces the following:

First Name	**Surname**	**Email 1**	**Email 2**	**Reg Num**
Mary	Jones	mj@my.uni.ac.uk	mj@somewhere.com	*Missing*
Adam	Smith	as@my.uni.ac.uk	*Missing*	12345

If you want to do this, you may find it easier to define a convenience command that will display some appropriate text if an entry is missing, for example:

```
\newcommand*{\checkmissing}[1]{\DTLifnull{#1}{---}{#1}}
```

Then instead of typing, say,

```
\DTLifnull{\regnum}{---}{\regnum}
```

you can instead type:

```
\checkmissing{\regnum}
```

Now suppose that instead of defining the database using \DTLnewdb, \DTLnewrow and \DTLnewdbentry, you have a file with the contents:

```
Surname,FirstName,RegNum,Email1,Email2
Jones,Mary,,mj@my.uni.ac.uk,mj@somewhere.com
Smith,Adam,12345,as@my.uni.ac.uk,
```

and you load the data from this file using \DTLloaddb (defined in subsection 5.2). Now the database has no null values, but has an empty value for the key RegNum

on the first row of the database, and an empty value for the key `Email2` on the second row of the database. Now, the following code

```
\begin{tabular}{lllll}
\bfseries First Name &
\bfseries Surname &
\bfseries Email 1 &
\bfseries Email 2 &
\bfseries Reg Number%
\DTLforeach{emailDB}{\firstname=FirstName,\surname=Surname,%
\emailI=Email1,\emailII=Email2,\regnum=RegNum}{%
\\\DTLifnull{\firstname}{\emph{Missing}}{\firstname} &
\DTLifnull{\surname}{\emph{Missing}}{\surname} &
\DTLifnull{\emailI}{\emph{Missing}}{\emailI} &
\DTLifnull{\emailII}{\emph{Missing}}{\emailII} &
\DTLifnull{\regnum}{\emph{Missing}}{\regnum}}%
\end{tabular}
```

produces:

First Name	**Surname**	**Email 1**	**Email 2**	**Reg Number**
Mary	Jones	mj@my.uni.ac.uk	mj@somewhere.com	
Adam	Smith	as@my.uni.ac.uk		12345

5.6 Editing Database Rows

In the body of the `\DTLforeach` loop,[4] you can use the following to edit the current row:

`\DTLappendtorow`

`\DTLappendtorow{`⟨*key*⟩`}{`⟨*value*⟩`}`

This appends a new entry with the given ⟨*key*⟩ and ⟨*value*⟩ to the current row.

`\DTLreplaceentryforrow`

`\DTLreplaceentryforrow{`⟨*key*⟩`}{`⟨*value*⟩`}`

This replaces the entry for ⟨*key*⟩ with ⟨*value*⟩.

`\DTLremoveentryfromrow`

`\DTLremoveentryfromrow{`⟨*key*⟩`}`

This removes the entry for ⟨*key*⟩ from the current row.

`\DTLremovecurrentrow`

`\DTLremovecurrentrow`

This removes the current row from the database.

Example 13 (Editing Database Rows)

In this example I have a CSV file called `marks.csv` that contains student marks for three assignments:

[4] Only the unstarred version of `\DTLforeach`; the starred version is read-only.

```
Surname,FirstName,StudentNo,Assignment 1,Assignment 2,Assignment 3
"Smith, Jr",John,102689,68,57,72
"Brown",Jane,102647,75,84,80
"Brown",Andy,103569,42,52,54
"Adams",Z\"oe,105987,52,48,57
"Brady",Roger,106872,58,60,62
"Verdon",Clare,104356,45,50,48
```

First load this into a database called `marks`:

```
\DTLloaddb{marks}{marks.csv}
```

Suppose now I want to compute the average mark for each student, and append this to the database. I can do this as follows:

```
\DTLforeach{marks}{%
\assignI=Assignment 1,%
\assignII=Assignment 2,%
\assignIII=Assignment 3}{%
\DTLmeanforall{\theMean}{\assignI,\assignII,\assignIII}%
\DTLappendtorow{Average}{\theMean}}
```

For each row in the `marks` database, I now have an extra key called `Average` that contains the average mark over all three assignments for a given student. I can now put this data into a table:

```
\begin{table}[htbp]
\caption{Student marks}
\centering
\begin{tabular}{llcccc}
\bfseries Surname & \bfseries First Name &
\bfseries Assign 1 &
\bfseries Assign 2 &
\bfseries Assign 3 &
\bfseries Average Mark%
\DTLforeach{marks}{\surname=Surname,\firstname=FirstName,\average
=Average,\assignI=Assignment 1,\assignII=Assignment 2,\assignIII
=Assignment 3}{\\\surname
& \firstname & \assignI & \assignII & \assignIII &
\DTLround{\average}{\average}{2}\average}\relax
\end{tabular}
\end{table}
```

This produces Table 18.

Note that if I only wanted the averages for the table and nothing else, I could simply have computed the average in each row of the table and displayed it without adding the information to the database, however I am going to reuse this information in example 32, so adding it to the database means that I don't need to recompute the mean.

Table 18: Student marks

Surname	**First Name**	**Assign 1**	**Assign 2**	**Assign 3**	**Average Mark**
Smith, Jr	John	68	57	72	65.67
Brown	Jane	75	84	80	79.67
Brown	Andy	42	52	54	49.33
Adams	Zöe	52	48	57	52.33
Brady	Roger	58	60	62	60
Verdon	Clare	45	50	48	47.67

5.7 Arithmetical Computations on Database Entries

The commands used in section 3 can be used on database entries. You can, of course, directly use the commands provided by the fp package if you know that the values are in the correct format (i.e. no currency symbols, no number group separators and a full stop as the decimal point) but if this is not the case, then you should use the commands described in section 3. If you want to use a command provided by the fp package, that does not have a wrapper function in datatool, then you will need to convert the value using `\DTLconverttodecimal`, and convert it back using either `\DTLdecimaltolocale` or `\DTLdecimaltocurrency`.

Example 14 (Arithmetical Computations)

In this example, I am going to produce a table similar to Table 4, except that I want to add an extra row at the end which contains the average score.

```
\begin{table}[htbp]
\caption{Student scores}\label{tab:mean}
\centering
\def\total{0}%
\begin{tabular}{llr}
\bfseries First Name &
\bfseries Surname &
\bfseries Score (\%)%
\DTLforeach{scores}{%
\firstname=FirstName,\surname=Surname,\score=Score}{%
\\
\firstname & \surname &
\DTLgadd{\total}{\score}{\total}%
\score
}\\
\multicolumn{2}{l}{\bfseries Average Score} &
\DTLsavelastrowcount{\n}%
\DTLdiv{\average}{\total}{\n}%
\DTLround{\average}{\average}{2}%
\average
\end{tabular}
\end{table}
```

This produces Table 19. **Notes:**

- I had to use `\DTLgadd` rather than `\DTLadd` since it occurs within a tabular environment which puts each entry in a local scope.

- I used `\DTLsavelastrowcount` to store the number of rows produced by `\DTLforeach` in the control sequence `\n`.
- I used `\DTLround` to round the average score to 2 decimal places.

Table 19: Student scores

First Name	**Surname**	**Score (%)**
John	Smith, Jr	68
Jane	Brown	75
Andy	Brown	42
Zöe	Adams	52
Roger	Brady	58
Clare	Verdon	45
Average Score		56.67

`\DTLsumforkeys`

`\DTLsumforkeys[`⟨*condition*⟩`][`⟨*assign list*⟩`]{`⟨*db list*⟩`}{`⟨*key list*⟩`}{`⟨*cmd*⟩`}`

This command sums all the entries over all the databases listed in the comma separated list of database names ⟨*db list*⟩ for each key in ⟨*key list*⟩ where the condition given by ⟨*condition*⟩ is true. The second optional argument ⟨*assign list*⟩ is the same as the assignment list used by `\DTLforeach`, so that you can use the information in ⟨*condition*⟩. The result is stored in ⟨*cmd*⟩ which must be a control sequence. For example:

```
\DTLsumforkeys{scores}{Score}{\total}
```

sets `\total` to the sum of all the scores in the database called `scores`.

`\DTLsumcolumn`

`\DTLsumcolumn{`⟨*db*⟩`}{`⟨*key*⟩`}{`⟨*cmd*⟩`}`

This is a faster version of `\DTLsumforkeys` that only sums the entries in a single column (specified by ⟨*key*⟩) for a single database (specified by ⟨*db*⟩) and doesn't provide any filtering. The result is stored in ⟨*cmd*⟩ which must be a control sequence.

`\DTLmeanforkeys`

`\DTLmeanforkeys[`⟨*condition*⟩`][`⟨*assign list*⟩`]{`⟨*db list*⟩`}{`⟨*key list*⟩`}{`⟨*cmd*⟩`}`

This command computes the arithmetic mean of all the entries over all the databases listed in ⟨*db list*⟩ for all keys in ⟨*key list*⟩ where the condition given by ⟨*condition*⟩ is true. The second optional argument ⟨*assign list*⟩ is the same as the assignment list used by `\DTLforeach`, so that you can use the information in ⟨*condition*⟩. The result is stored in ⟨*cmd*⟩ which must be a control sequence. For example:

```
\DTLmeanforkeys{scores}{Score}{\average}
```

sets \average to the mean of all the scores in the database called scores.

\DTLmeanforcolumn

\DTLmeanforcolumn{⟨*db*⟩}{⟨*key*⟩}{⟨*cmd*⟩}

This is a faster version of \DTLmeanforkeys that only computes the mean for a single column (specified by ⟨*key*⟩) for a single database (specified by ⟨*db*⟩) and doesn't provide any filtering. The result is stored in ⟨*cmd*⟩ which must be a control sequence.

\DTLvarianceforkeys

\DTLvarianceforkeys[⟨*condition*⟩][⟨*assign list*⟩]{⟨*db list*⟩}{⟨*key list*⟩}{⟨*cmd*⟩}

This command computes the variance of all the entries over all the databases listed in ⟨*db list*⟩ for all keys in ⟨*key list*⟩ where the condition given by ⟨*condition*⟩ is true. The second optional argument ⟨*assign list*⟩ is the same as the assignment list used by \DTLforeach, so that you can use the information in ⟨*condition*⟩. The result is stored in ⟨*cmd*⟩ which must be a control sequence.

\DTLvarianceforcolumn

\DTLvarianceforcolumn{⟨*db*⟩}{⟨*key*⟩}{⟨*cmd*⟩}

This is a faster version of \DTLvarianceforkeys that only computes the variance for a single column (specified by ⟨*key*⟩) for a single database (specified by ⟨*db*⟩) and doesn't provide any filtering. The result is stored in ⟨*cmd*⟩ which must be a control sequence.

\DTLsdforkeys

\DTLsdforkeys[⟨*condition*⟩][⟨*assign list*⟩]{⟨*db list*⟩}{⟨*key list*⟩}{⟨*cmd*⟩}

This command computes the standard deviation of all the entries over all the databases listed in ⟨*db list*⟩ for all keys in ⟨*key list*⟩ where the condition given by ⟨*condition*⟩ is true. The second optional argument ⟨*assign list*⟩ is the same as the assignment list used by \DTLforeach, so that you can use the information in ⟨*condition*⟩. The result is stored in ⟨*cmd*⟩ which must be a control sequence.

\DTLsdforcolumn

\DTLsdforcolumn{⟨*db*⟩}{⟨*key*⟩}{⟨*cmd*⟩}

This is a faster version of \DTLsdforkeys that only computes the standard deviation for a single column (specified by ⟨*key*⟩) for a single database (specified by ⟨*db*⟩) and doesn't provide any filtering. The result is stored in ⟨*cmd*⟩ which must be a control sequence.

\DTLminforkeys

\DTLminforkeys[⟨*condition*⟩][⟨*assign list*⟩]{⟨*db list*⟩}{⟨*key list*⟩}{⟨*cmd*⟩}

This command determines the minimum value over all entries for all keys in ⟨*key list*⟩ over all the databases listed in ⟨*db list*⟩ where ⟨*condition*⟩ is true. The second optional argument ⟨*assign list*⟩ is the same as the assignment list used by \DTLforeach, so that you can use the information in ⟨*condition*⟩. The result is stored in ⟨*cmd*⟩, which must be a control sequence. For example

```
\DTLminforkeys{scores}{Score}{\theMin}
```

sets `\theMin` to the minimum score in the database.

`\DTLminforcolumn`

`\DTLminforcolumn{⟨db⟩}{⟨key⟩}{⟨cmd⟩}`

This is a faster version of `\DTLminforkeys` that only computes the minimum for a single column (specified by ⟨*key*⟩) for a single database (specified by ⟨*db*⟩) and doesn't provide any filtering. The result is stored in ⟨*cmd*⟩ which must be a control sequence.

`\DTLmaxforkeys`

`\DTLmaxforkeys[⟨condition⟩][⟨assign list⟩]{⟨db list⟩}{⟨key list⟩}{⟨cmd⟩}`

This command determines the maximum value over all entries for all keys in ⟨*key list*⟩ over all the databases listed in ⟨*db list*⟩ where ⟨*condition*⟩ is true. The second optional argument ⟨*assign list*⟩ is the same as the assignment list used by `\DTLforeach`, so that you can use the information in ⟨*condition*⟩. The result is stored in ⟨*cmd*⟩, which must be a control sequence. For example

```
\DTLminforkeys{scores}{Score}{\theMax}
```

sets `\theMax` to the minimum score in the database.

`\DTLmaxforcolumn`

`\DTLmaxforcolumn{⟨db⟩}{⟨key⟩}{⟨cmd⟩}`

This is a faster version of `\DTLmaxforkeys` that only computes the maximum for a single column (specified by ⟨*key*⟩) for a single database (specified by ⟨*db*⟩) and doesn't provide any filtering. The result is stored in ⟨*cmd*⟩ which must be a control sequence.

`\DTLcomputebounds`

`\DTLcomputebounds{⟨db list⟩}{⟨x key⟩}{⟨y key⟩}{⟨minX cmd⟩}{⟨minY cmd⟩}{⟨maxX cmd⟩}{⟨maxY cmd⟩}`

Computes the maximum and minimum x and y values over all the databases listed in ⟨*db list*⟩ where the x value is given by ⟨*x key*⟩ and the y value is given by ⟨*y key*⟩. The results are stored in ⟨*minX cmd*⟩, ⟨*minY cmd*⟩, ⟨*maxX cmd*⟩ and ⟨*maxY cmd*⟩.

Example 15 (Mail Merging)

This example uses the database given in example 3 and uses `\DTLmeanforkeys` to determine the average score. A letter is then created for each student to inform them of their score and the class average.

```
\documentclass{letter}

\usepackage{datatool}

\begin{document}
 % load database
\DTLloaddb{scores}{studentscores.csv}
```

```
 % compute arithmetic mean for key ‘Score’
\DTLmeanforkeys{scores}{Score}{\average}
 % Round the average to 2 decimal places
\DTLround{\average}{\average}{2}
 % Save the highest score in \maxscore
\DTLmaxforkeys{scores}{Score}{\maxscore}

\DTLforeach{scores}{\firstname=FirstName,\surname=Surname,%
\score=Score}{%
\begin{letter}{}
\opening{Dear \firstname\ \surname}

\DTLifnumgt{\score}{60}{Congratulations you}{You} achieved a score
of \score\% which was \DTLifnumgt{\score}{\average}{above}{below}
the average of \average\%. \DTLifnumeq{\score}{\maxscore}{You
achieved the highest score}{The top score was \maxscore}.

\closing{Yours Sincerely}
\end{letter}
}
\end{document}
```

5.8 Sorting a Database

`\DTLsort`

`\DTLsort[`⟨*replacement key list*⟩`]{`⟨*sort criteria*⟩`}{`⟨*db name*⟩`}`

`\DTLsort*`

`\DTLsort*[`⟨*replacement key list*⟩`]{`⟨*sort criteria*⟩`}{`⟨*db name*⟩`}`

This will sort the database called ⟨*db name*⟩ according to the criteria given by ⟨*sort criteria*⟩, which must be a comma separated list of keys and optionally =⟨*order*⟩, where ⟨*order*⟩ is either `ascending` or `descending`. If the order is omitted, `ascending` is assumed. The database keeps track of the data type for a given key, and uses this to determine whether an alphabetical or numerical sort is required. (String comparisons are made using the command `\dtlcompare` or `\dtlicompare` described in subsection 10.3.)

The optional argument ⟨*replacement key list*⟩ is a list of keys to use if the current key given in ⟨*sort criteria*⟩ is null for a given entry. Null keys are unlikely to occur if you have loaded the database from an external ASCII file, but may occur if the database is created using `\DTLnewdb`, `\DTLnewrow` and `\DTLnewdbentry`. For example:

```
\DTLsort[Editor,Organization]{Author}{mydata}
```

will sort according to the `Author` key, but if that key is missing for a given row of the database, the `Editor` key will be used, and if the `Editor` key is missing, it will use the `Organization` key. Note that this is not the same as:

```
\DTLsort{Author,Editor,Organization}{mydata}
```

which will first compare the `Author` keys, but if the author names are the same, it will then compare the `Editor` keys, and if the editor names are also the same, it will then compare the `Organization` keys.

The unstarred version uses a case sensitive comparison for strings, whereas the starred version ignores the case when comparing strings. Note that the case sensitive comparison orders uppercase characters before lowercase characters, so the letter B is considered to be lower than the letter a.

Example 16 (Sorting a Database)

This example uses the database called `scores` defined in example 3. First, I am going to sort the database according to the student scores in descending order (highest to lowest) and display the database in a table

```
\begin{table}[htbp]
\caption{Student scores (sorted by score)}
\centering
\DTLsort{Score=descending}{scores}%
\begin{tabular}{llr}
\bfseries First Name &
\bfseries Surname &
\bfseries Score (\%)%
\DTLforeach{scores}{%
\firstname=FirstName,\surname=Surname,\score=Score}{%
\\
\firstname & \surname & \score}
\end{tabular}
\end{table}
```

This produces Table 20.

Table 20: Student scores (sorted by score)

First Name	**Surname**	**Score (%)**
Jane	Brown	75
John	Smith, Jr	68
Roger	Brady	58
Zöe	Adams	52
Clare	Verdon	45
Andy	Brown	42

Now I am going to sort the database according to surname and then first name, and display it in a table. Note that since I want to sort in ascending order, I can omit the `=ascending` part of the sort criteria. I have also decided to reverse the first and second columns, so that the surname is in the first column.

```
\begin{table}[htbp]
\caption{Student scores (sorted by name)}
\centering
\DTLsort{Surname,FirstName}{scores}%
\begin{tabular}{llr}
\bfseries Surname &
\bfseries First Name &
```

```
\bfseries Score (\%)%
\DTLforeach{scores}{%
\firstname=FirstName,\surname=Surname,\score=Score}{%
\\
\surname & \firstname & \score}
\end{tabular}
\end{table}
```

This produces Table 21.

Table 21: Student scores (sorted by name)

Surname	**First Name**	**Score (%)**
Adams	Zöe	52
Brady	Roger	58
Brown	Andy	42
Brown	Jane	75
Smith, Jr	John	68
Verdon	Clare	45

Now suppose I add two new students to the database:

```
\DTLnewrow{scores}%
\DTLnewdbentry{scores}{Surname}{van der Mere}%
\DTLnewdbentry{scores}{FirstName}{Henk}%
\DTLnewdbentry{scores}{Score}{71}%
\DTLnewrow{scores}%
\DTLnewdbentry{scores}{Surname}{de la Mere}%
\DTLnewdbentry{scores}{FirstName}{Jos}%
\DTLnewdbentry{scores}{Score}{58}%
```

and again I try sorting the database, and displaying the contents as a table:

```
\begin{table}[htbp]
\caption{Student scores (case sensitive sort)}
\centering
\DTLsort{Surname,FirstName}{scores}%
\begin{tabular}{llr}
\bfseries Surname &
\bfseries First Name &
\bfseries Score (\%)%
\DTLforeach{scores}{%
\firstname=FirstName,\surname=Surname,\score=Score}{%
\\
\surname & \firstname & \score}
\end{tabular}
\end{table}
```

This produces Table 22. Notice that the surnames aren't correctly ordered. This is because a case-sensitive sort was used. Changing `\DTLsort` to `\DTLsort*` in the above code produces Table 23.

Table 22: Student scores (case sensitive sort)

Surname	**First Name**	**Score (%)**
Adams	Zöe	52
Brady	Roger	58
Brown	Andy	42
Brown	Jane	75
Smith, Jr	John	68
Verdon	Clare	45
de la Mere	Jos	58
van der Mere	Henk	71

Table 23: Student scores (case ignored when sorting)

Surname	**First Name**	**Score (%)**
Adams	Zöe	52
Brady	Roger	58
Brown	Andy	42
Brown	Jane	75
de la Mere	Jos	58
Smith, Jr	John	68
van der Mere	Henk	71
Verdon	Clare	45

Example 17 (Influencing the sort order)

Consider the data displayed in Table 23, suppose that you want the names "van der Mere" and "de la Mere" sorted according to the actual surname "Mere" rather than by the "von part". There are two ways you can do this: firstly, you could store the von part in a separate field, and then sort by surname, then von part, then first name, or you could define a command called, say, `\switchargs`, as follows:

```
\newcommand*{\switchargs}[2]{#2#1}
```

then store the data as:

```
FirstName,Surname,StudentNo,Score
John,"Smith, Jr",102689,68
Jane,Brown,102647,75
Andy,Brown,103569,42
Z\"oe,Adams,105987,52
Roger,Brady,106872,58
Clare,Verdon,104356,45
Henk,\switchargs{Mere}{van der },106789,71
Jos,\switchargs{Mere}{de la },104256,58
```

Now sort the data, and put it in table (this is the same code as in the previous example:

```
\begin{table}[htbp]
\caption{Student scores (influencing the sort order)}
\centering
```

```
\DTLsort*{Surname,FirstName}{scores}%
\begin{tabular}{llr}
\bfseries Surname &
\bfseries First Name &
\bfseries Score (\%)%
\DTLforeach{scores}{%
\firstname=FirstName,\surname=Surname,\score=Score}{%
\\
\surname & \firstname & \score}
\end{tabular}
\end{table}
```

This produces Table 24.

Table 24: Student scores (influencing the sort order)

Surname	**First Name**	**Score (%)**
Adams	Zöe	52
Brady	Roger	58
Brown	Andy	42
Brown	Jane	75
de la Mere	Jos	58
van der Mere	Henk	71
Smith, Jr	John	68
Verdon	Clare	45

5.9 Saving a Database to an External File

`\DTLsavedb`

`\DTLsavedb{`⟨*db name*⟩`}{`⟨*filename*⟩`}`

This writes the database called ⟨*db name*⟩ to a file called ⟨*filename*⟩. The separator and delimiter characters used are as given by `\DTLsetseparator` (or `\DTLsettabseparator`) and `\DTLsetdelimiter`. For example:

```
\DTLsettabdelimiter
\DTLsavedb{scores}{scores.txt}
```

will create a file called `scores.txt` and will save the data in a tab separated format. (The delimiters will only be used if a given entry contains the separator character.)

`\DTLsavetexdb`

`\DTLsavetexdb{`⟨*db name*⟩`}{`⟨*filename*⟩`}`

This writes the database called ⟨*db name*⟩ to a LaTeX file called ⟨*filename*⟩, where the database is stored as a combination of `\DTLnewdb`, `\DTLnewrow` and `\DTLnewdbentry` commands.

6 Pie Charts (datapie package)

The datapie package is not loaded by the datatool package, so you need to explicitly load datapie if you want to use any of the commands defined in this section. You will also need to have the pgf/tikz packages installed. The datapie package may be given the following options:

color Colour option (default).

monochrome Monochrome option.

rotateinner Rotate inner labels so that they are aligned with the pie chart radial axis.

norotateinner Don't rotate inner labels (default).

rotateouter Rotate outer labels so that they are aligned with the pie chart radial axis.

norotateouter Don't rotate outer labels (default).

Numerical information contained in a database created by the datatool package can be converted into a pie chart using

`\DTLpiechart`

```
\DTLpiechart[⟨condition⟩]{⟨settings list⟩}{⟨db name⟩}{⟨values⟩}
```

where ⟨*db name*⟩ is the name of the database, and ⟨*condition*⟩ has the same form as the optional argument to `\DTLforeach` described in subsection 5.4. If ⟨*condition*⟩ is false, that information is omitted from the construction of the pie chart. The argument ⟨*values*⟩ is a comma separated list of ⟨*cmd*⟩=⟨*key*⟩ pairs, the same as that required by the penultimate argument of `\DTLforeach`. The ⟨*settings list*⟩ is a comma separated list of ⟨*setting*⟩=⟨*value*⟩ pairs, where ⟨*setting*⟩ can be any of the following:

variable This specifies the control sequence to use that contains the value used to construct the pie chart. The control sequence must be one of the control sequences to appear in the assignment list ⟨*values*⟩. This setting is required.

start This is the starting angle of the first segment. The value is 0 by default.

radius This is the radius of the pie chart. The default value is 2cm.

innerratio The distance from the centre of the pie chart to the point where the inner labels are placed is given by this value multiplied by the ratio. The default value is 0.5.

outerratio The distance from the centre of the pie chart to the point where the outer labels are placed is given by this value multiplied by the ratio. The default value is 1.25.

cutawayratio The distance from the centre of the pie chart to the point of cutaway segments is given by this value multiplied by the ratio. The default value is 0.2.

inneroffset This is the absolute distance from the centre of the pie chart to the point where the inner labels are placed. You should use only one or other of innerratio and inneroffset, not both. If you also want to specify the radius, you must use ratio before inneroffset. If omitted, the inner offset is obtained from the ratio multiplied by the innerratio value.

outeroffset This is the absolute distance from the centre of the pie chart to the point where the outer labels are placed. You should use only one or other of outerratio and outeroffset, not both. If you also want to specify the radius, you must use ratio before outeroffset. If omitted, the outer offset is obtained from the ratio multiplied by the outerratio value.

cutawayoffset This is the absolute distance from the centre of the pie chart to the point of the cutaway segments. You should use only one or other of cutawayratio and cutawayoffset, not both. If you also want to specify the radius, you must use ratio before cutawayoffset. If omitted, the cutaway offset is obtained from the ratio multiplied by the cutawayratio value.

cutaway This is a list of cutaway segments. This should be a comma separated list of individual numbers, or number ranges (separated by a dash). For example `cutaway={1,3}` will separate the first and third segments from the rest of the pie chart, offset by the value of the cutawayoffset setting, whereas `cutaway={1-3}` will separate the first three segments from the rest of the pie chart. If omitted, the pie chart will be whole.

innerlabel The value of this is positioned in the middle of each segment at a distance of inneroffset from the centre of the pie chart. The default is the same as the value of variable.

outerlabel The value of this is positioned at a distance of outeroffset from the centre of the pie chart. The default is empty.

rotateinner This is a boolean setting, so it can only take the values `true` and `false`. If the value is omitted `true` is assumed. If true, the inner labels are rotated along the spokes of the pie chart, otherwise the inner labels are not rotated. There are analogous package options rotateinner and norotateinner.

rotateouter This is a boolean setting, so it can only take the values `true` and `false`. If the value is omitted `true` is assumed. If true, the outer labels are rotated along the spokes of the pie chart, otherwise the outer labels are not rotated. There are analogous package options rotateouter and norotateouter.

Example 18 (A Pie Chart)

This example loads data from a file called `fruit.csv` which contains the following:

```
Name,Quantity
"Apples",30
"Pears",25
"Lemons,Limes",40.5
"Peaches",34.5
"Cherries",20
```

First load the data:

```
\DTLloaddb{fruit}{fruit.csv}
```

Now create a pie chart in a figure:

```
\begin{figure}[htbp]
\centering
\DTLpiechart{variable=\quantity}{fruit}{\name=Name,\quantity=Quantity}
\caption{A pie chart}
\end{figure}
```

This creates Figure 1.

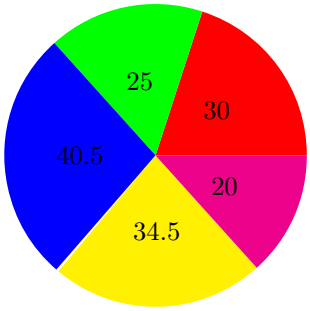

Figure 1: A pie chart

There are no outer labels by default, but they can be set using the outerlabel setting. The following sets the outer label to the value of the `Name` key:

```
\begin{figure}[htbp]
\centering
\DTLpiechart{variable=\quantity,outerlabel=\name}{fruit}{%
\name=Name,\quantity=Quantity}
\caption{A pie chart (outer labels set)}
\end{figure}
```

This creates Figure 2.

You may prefer the labels to be rotated. The following switches on the rotation for the inner and outer labels:

```
\begin{figure}[htbp]
\centering
\DTLpiechart{variable=\quantity,outerlabel=\name,%
rotateinner,rotateouter}{fruit}{%
\name=Name,\quantity=Quantity}
\caption{A pie chart (rotation enabled)}
\end{figure}
```

This creates Figure 3.

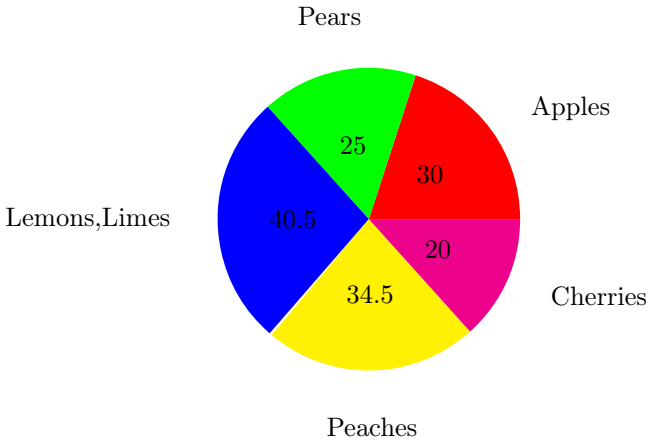

Figure 2: A pie chart (outer labels set)

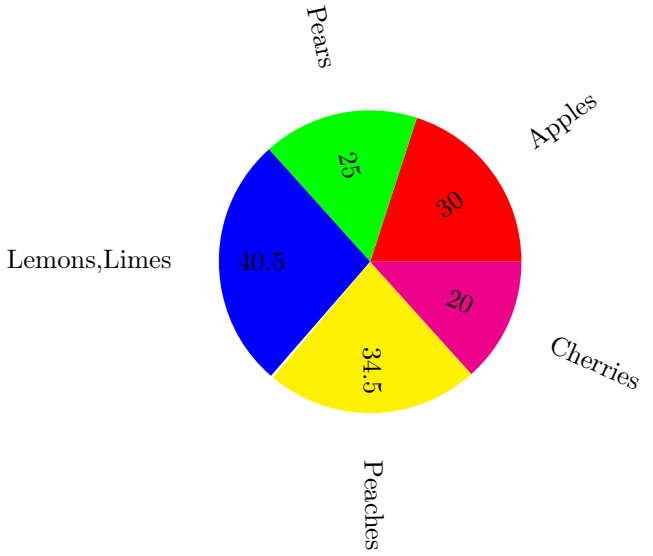

Figure 3: A pie chart (rotation enabled)

Example 19 (Separating Segments from the Pie Chart)

You may want to separate one or more segments from the pie chart, perhaps to emphasize them. You can do this using the cutaway setting. The following separates the first and third segments from the pie chart:

```
\begin{figure}[htbp]
\centering
\DTLpiechart{variable=\quantity,outerlabel=\name,%
cutaway={1,3}}{fruit}{%
\name=Name,\quantity=Quantity}
\caption{A pie chart with cutaway segments}
\end{figure}
```

This produces Figure 4.

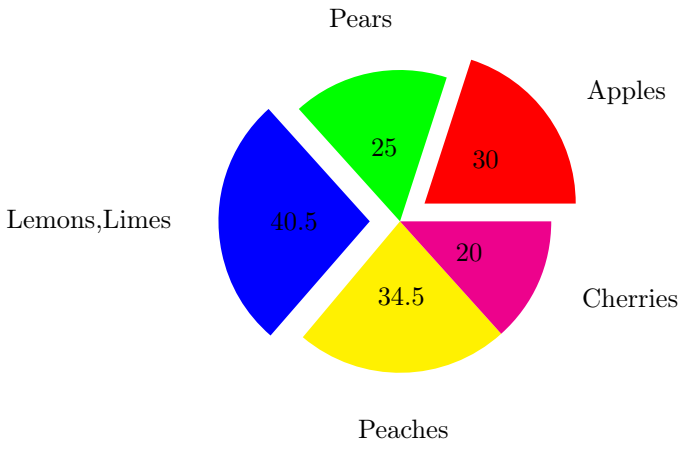

Figure 4: A pie chart with cutaway segments

Alternatively I can specify a range of segments. The following separates the first two segments:

```
\begin{figure}[htbp]
\centering
\DTLpiechart{variable=\quantity,outerlabel=\name,%
cutaway={1-2}}{fruit}{%
\name=Name,\quantity=Quantity}
\caption{A pie chart with cutaway segments (\texttt{cutaway=\{1-2\}})}
\end{figure}
```

This produces Figure 5.

Notice the difference between Figure 5 and Figure 6 which was produced using:

```
\begin{figure}[htbp]
\centering
\DTLpiechart{variable=\quantity,outerlabel=\name,%
cutaway={1,2}}{fruit}{%
\name=Name,\quantity=Quantity}
\caption{A pie chart with cutaway segments (\texttt{cutaway=\{1,2\}})}
\end{figure}
```

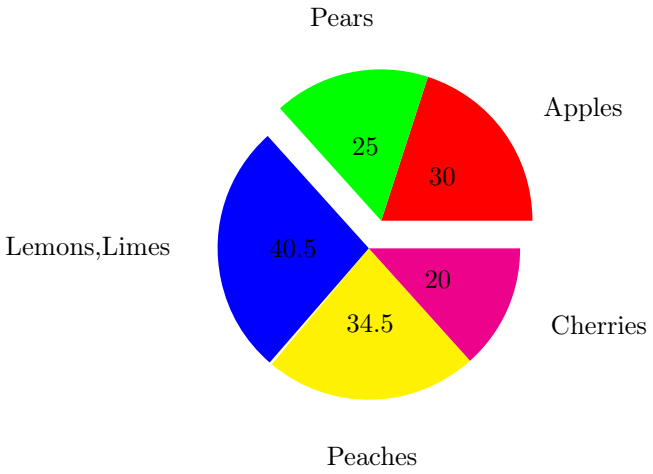

Figure 5: A pie chart with cutaway segments (`cutaway={1-2}`)

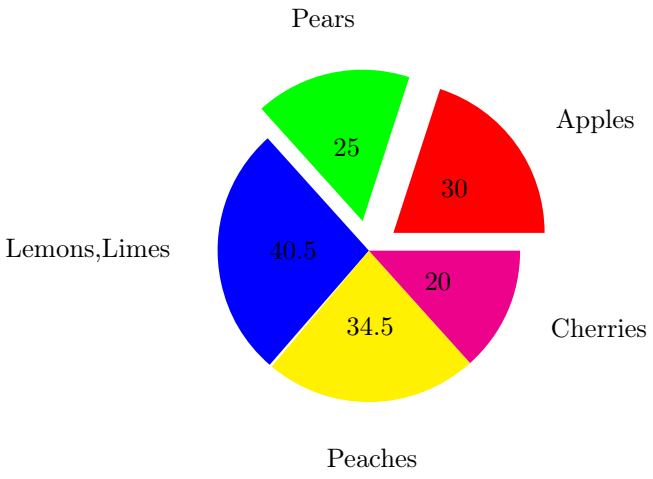

Figure 6: A pie chart with cutaway segments (`cutaway={1,2}`)

6.1 Pie Chart Variables

\DTLpievariable

```
\DTLpievariable
```

This command is set to the variable given by the variable setting in the ⟨*settings list*⟩ argument of \DTLpiechart. The innerlabel is set to \DTLpievariable by default.

\DTLpiepercent

```
\DTLpiepercent
```

This command is set to the percentage value of \DTLpievariable. The percentage value is rounded to ⟨n⟩ digits, where ⟨n⟩ is the value of the LaTeX counter DTLpieroundvar.

Example 20 (Changing the Inner and Outer Labels)

This example uses the database defined in example 18. The inner label is now set to the percentage value, rather than the actual value, and the outer label is set to the name with the actual value in parentheses.

```
\begin{figure}[htbp]
\centering
\DTLpiechart{variable=\quantity,%
innerlabel={\DTLpiepercent\%},%
outerlabel={\name\ (\DTLpievariable)}}{fruit}{%
\name=Name,\quantity=Quantity}
\caption{A pie chart (changing the labels)}
\end{figure}
```

This produces Figure 7.

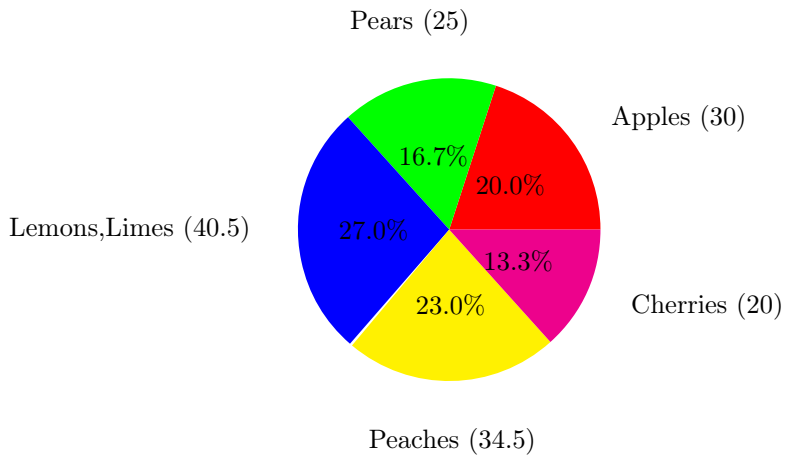

Figure 7: A pie chart (changing the labels)

6.2 Pie Chart Label Formatting

\DTLdisplayinnerlabel

\DTLdisplayinnerlabel{⟨*text*⟩}

This governs how the inner label is formatted, where ⟨*text*⟩ is the text of the inner label. The default is to just do ⟨*text*⟩.

\DTLdisplayouterlabel

\DTLdisplayouterlabel{⟨*text*⟩}

This governs how the outer label is formatted, where ⟨*text*⟩ is the text of the outer label. The default is to just do ⟨*text*⟩.

Example 21 (Changing the Inner and Outer Label Format)

This example extends example 20. The inner and outer labels are now both typeset in a sans-serif font:

```
\begin{figure}[htbp]
\centering
\renewcommand*{\DTLdisplayinnerlabel}[1]{\textsf{#1}}
\renewcommand*{\DTLdisplayouterlabel}[1]{\textsf{#1}}
\DTLpiechart{variable=\quantity,%
innerlabel={\DTLpiepercent\%},%
outerlabel={\name\ (\DTLpievariable)}}{fruit}{%
\name=Name,\quantity=Quantity}
\caption{A pie chart (changing the label format)}
\end{figure}
```

This produces Figure 8.

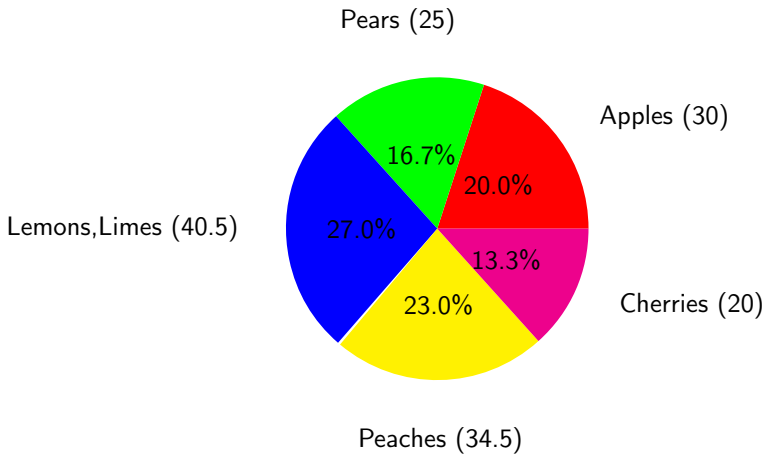

Figure 8: A pie chart (changing the label format)

6.3 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 will need to use

\DTLsetpiesegmentcolor

```
\DTLsetpiesegmentcolor{⟨n⟩}{⟨color⟩}
```

The first argument ⟨*n*⟩ is the segment index (starting from 1), and the second argument ⟨*color*⟩ is a colour specifier as used in commands such as `\color`.

It is a good idea to set the colours so that each segment colour is somehow relevant to whatever the segment represents. For example, in the previous examples of pie charts depicting fruit, some of default colours were inappropriate. Whilst red is appropriate for apples and green is appropriate for pears, blue doesn't really correspond to lemons or limes.

\DTLdopiesegmentcolor

```
\DTLdopiesegmentcolor⟨n⟩
```

This sets the current text colour to that of the ⟨*n*⟩th segment.

\DTLdocurrentpiesegmentcolor

```
\DTLdocurrentpiesegmentcolor
```

This sets the current text colour to that of the current pie segment. This command may only be used within a pie chart, or within the body of `\DTLforeach`.

\DTLpieoutlinecolor

```
\DTLpieoutlinecolor
```

This sets the outline colour for the pie chart. The default is black.

\DTLpieoutlinewidth

```
\DTLpieoutlinewidth
```

This is a length that governs the line width of the outline. The default value is 0pt, but can be changed using `\setlength`. The outline is only drawn if `\DTLpieoutlinewidth` is greater than 0pt.

Example 22 (Pie Segment Colours)

This example extends example 21. It sets the outline thickness to 2pt, and the outer label is now set in the same colour as the fill colour of the segment to which it belongs. The third segment (lemons and limes) is set to yellow and the fourth segment (peaches) is set to pink. In addition, a legend is created using `\DTLforeach`.

```
\begin{figure}[htbp]
\centering
\setlength{\DTLpieoutlinewidth}{2pt}
\DTLsetpiesegmentcolor{3}{yellow}
\DTLsetpiesegmentcolor{4}{pink}
\renewcommand*{\DTLdisplayinnerlabel}[1]{\textsf{#1}}
\renewcommand*{\DTLdisplayouterlabel}[1]{%
```

```
\DTLdocurrentpiesegmentcolor
\textsf{\shortstack{#1}}}
\DTLpiechart{variable=\quantity,%
innerlabel={\DTLpiepercent\%},%
outerlabel={\name\\(\DTLpievariable)}}{fruit}{%
\name=Name,\quantity=Quantity}
\begin{tabular}[b]{ll}
\DTLforeach{fruit}{\name=Name}{\DTLiffirstrow{}{\\}%
\DTLdocurrentpiesegmentcolor\rule{10pt}{10pt} &
\name
}
\end{tabular}
\caption{A pie chart (using segment colours and outline)}
\end{figure}
```

This produces Figure 9. (The format of the outer label has been changed to use `\shortstack` to prevent the outer labels from taking up so much horizontal space. The outerlabel setting has also been modified to use \\ after the name to move the percentage value onto the next row.)

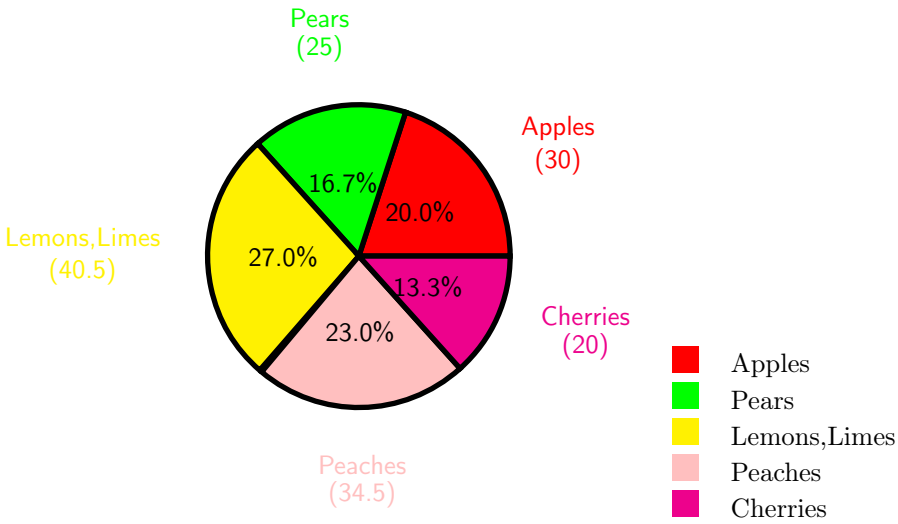

Figure 9: A pie chart (using segment colours and outline)

6.4 Adding Extra Commands Before and After the Pie Chart

The pie charts created using `\DTLpiechart` are placed inside a tikzpicture environment (defined by the tikz package).

`\DTLpieatbegintikz`

```
\DTLpieatbegintikz
```

The macro `\DTLpieatbegintikz` is called at the start of the tikzpicture environment, allowing you to change the tikzpicture settings. By default

\DTLpieatbegintikz does nothing, but you can redefine it to, say, scale the pie chart (but be careful not to distort the chart).

\DTLpieatendtikz

```
\DTLpieatendtikz
```

The macro \DTLpieatendtikz is called at the end of the tikzpicture environment, allowing you add additional graphics to the pie chart. This does nothing by default.

Example 23 (Adding Information to the Pie Chart)

This example modifies example 18. It redefines \DTLpieatendtikz to add an annotated arrow.

```
\begin{figure}[htbp]
\centering
\renewcommand*{\DTLpieatendtikz}{%
\draw[<-] (45:1.5cm) -- (40:2.5cm)node[right]{Apples};}
\DTLpiechart{variable=\quantity}{fruit}{%
\name=Name,\quantity=Quantity}
\caption{An annotated pie chart}
\end{figure}
```

This produces Figure 10. (Note that the centre of the pie chart is the origin of the TikZ picture.)

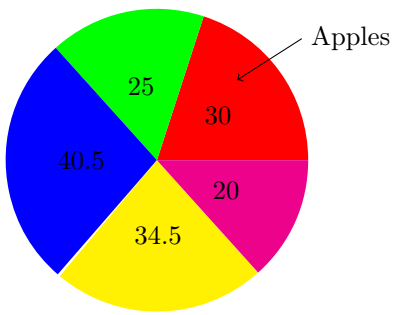

Figure 10: An annotated pie chart

---

7 Scatter and Line Plots (dataplot package)

The dataplot package provides commands for creating scatter or line plots from databases. It uses the pgf/TikZ plot handler library to create the plots. See the pgf manual for more detail on pgf streams and plot handles. The dataplot package

is not loaded by datatool so if you want to use it you need to load it explicitly using \usepackage{dataplot}.

\DTLplot

\DTLplot[⟨*condition*⟩]{⟨*db list*⟩}{⟨*settings*⟩}

This command creates a plot (inside a tikzpicture environment) of all the data given in the databases listed in ⟨*db list*⟩, which should be a comma separated list of database names. The optional argument ⟨*condition*⟩ is the same as that for \DTLforeach. The ⟨*settings*⟩ argument is a comma separated list of ⟨*setting*⟩=⟨*value*⟩ pairs. There are two settings that must be specified x and y. The other settings are optional. Note that any value that contains a comma, must be enclosed in braces. For example colors={red,cyan,blue}. Note where any setting requires a number, or list of numbers (such as bounds) the number must be supplied in standard decimal notation (i.e. no currency, no number groups, and a full stop as the decimal point). Available settings are as follows:

x The database key that specifies the x co-ordinates. This setting is required.

y The database key that specifies the y co-ordinates. This setting is required.

markcolors A comma separated list of colour names for the markers. An empty value will use the current colour.

linecolors A comma separated list of colour names for the plot lines. An empty value will use the current colour.

colors A comma separated list of colour names for the lines and markers.

marks A comma separated list of code to generate plot marks. (This should typically be a list of \pgfuseplotmark commands, see the pgf manual for further details.) You may use \relax as an element of the list to suppress markers for the corresponding plot. For example: marks={\pgfuseplotmark{o},\relax} will use an open circle marker for the first database, and no markers for the second database listed in ⟨*db list*⟩.

lines A comma separated list of line style settings. (This should typically be a list of \pgfsetdash commands, see the pgf manual for further details on how to set the line style.) An empty value will use the current line style. You may use \relax as an element of the list to suppress line for the corresponding plot. For example: lines={\relax,\pgfsetdash{}{0pt}} will have no lines for the first database, and a solid line for the second database listed in ⟨*db list*⟩.

width The width of the plot. This must be a length. The plot width does not include outer tick marks or labels.

height The height of the plot. This must be a length. The plot height does not include outer tick marks or labels.

style This setting governs whether to use lines or markers in the plot, and may take one of the following values: both (lines and markers), lines (only lines) or markers (only markers). The default is markers.

axes This setting governs whether to display the axes, and may take one of the following values: `both`, `x`, `y` or `none`. If no value is specified, `both` is assumed.

box This setting governs whether or not to surround the plot in a box. It is a boolean setting, taking only the values `true` and `false`. If no value is specified, `true` is assumed.

xtics This setting governs whether or not to display the x tick marks. It is a boolean setting, taking only the values `true` and `false`. If no value is specified `true` is assumed. If the axes setting is set to `both` or `x`, this value will automatically be set to `true`, otherwise it will be set to `false`.

ytics This setting governs whether or not to display the y ticks. It is a boolean setting, taking only the values `true` and `false`. If no value is specified `true` is assumed. If the axes setting is set to `both` or `y`, this value will automatically be set to `true`, otherwise it will be set to `false`.

xminortics This setting governs whether or not to display the x minor tick marks. It is a boolean setting, taking only the values `true` and `false`. If no value is specified `true` is assumed. This setting also sets the x major tick marks on if the value is `true`.

yminortics This setting governs whether or not to display the y minor tick marks. It is a boolean setting, taking only the values `true` and `false`. If no value is specified `true` is assumed. This setting also sets the y major tick marks on if the value is `true`.

xticdir This sets the x tick direction, and may only take the values `in` or `out`.

yticdir This sets the y tick direction, and may only take the values `in` or `out`.

ticdir This sets the x and y tick direction, and may only take the values `in` or `out`.

bounds The value must be in the form $\langle min\ x\rangle$`,`$\langle min\ y\rangle$`,`$\langle max\ x\rangle$`,`$\langle max\ y\rangle$. This sets the graph bounds to the given values. If omitted the bounds are computed from the maximum and minimum values of the data. For example

```
\DTLplot{data1,data2}{x=Height,y=Weight,bounds={0,0,10,20}}
```

Note that the bounds setting overrides the minx, maxx, miny and maxy settings.

minx The value is the minimum value of the x axis.

miny The value is the minimum value of the y axis.

maxx The value is the maximum value of the x axis.

maxy The value is the maximum value of the y axis.

xticpoints The value must be a comma separated list of decimal numbers indicating where to put the x tick marks. If omitted, the x tick marks are placed at equal intervals along the x axis such that each interval is not less than the length given by `\DTLmintickgap`. This setting overrides xticgap.

xticgap This value specifies the gap between the x tick marks.

yticpoints The value must be a comma separated list of decimal numbers indicating where to put the y tick marks. If omitted, the y tick marks are placed at equal intervals along the y axis such that each interval is not less than the length given by `\DTLmintickgap`. This setting overrides `yticgap`.

yticgap This value specifies the gap between the y tick marks.

grid This is a boolean value that specifies whether or not to display the grid. If no value is given, `true` is assumed. The minor grid lines are only displayed if the minor tick marks are set.

xticlabels The value must be a comma separated list of labels for each x tick mark. If omitted, the labels are the value of the x tick position, rounded $\langle n \rangle$ digits after the decimal point, where $\langle n \rangle$ is given by the value of the counter `DTLplotroundXvar`.

yticlabels The value must be a comma separated list of labels for each y tick mark. If omitted, the labels are the value of the y tick position, rounded $\langle n \rangle$ digits after the decimal point, where $\langle n \rangle$ is given by the value of the counter `DTLplotroundYvar`.

xlabel The value is the label for the x axis. If omitted, the axis has no label.

ylabel The value is the label for the y axis. If omitted, the axis has no label.

legend This setting governs whether or not to display the legend, and where it should be displayed. It may take one of the following values `none` (don't display the legend), `north`, `northeast`, `east`, `southeast`, `south`, `southwest`, `west` or `northwest`. If the value is omitted, `northeast` is assumed.

legendlabels The value must be a comma separated list of labels for the legend. If omitted, the database names are used.

Example 24 (A Basic Graph)

Suppose you have a file called `groupa.csv` that contains the following:

```
Height,Weight
1.55,45.4
1.54,48.0
1.56,58.0
1.56,50.2
1.57,46.0
1.58,48.3
1.59,56.5
1.59,58.1
1.60,60.9
1.62,56.3
```

First load this into a database called `groupa`:

```
\DTLloaddb{groupa}{groupa.csv}
```

The data can now be converted into a scatter plot as follows:

```
\begin{figure}[htbp]
\centering
\DTLplot{groupa}{x=Height,y=Weight}
\caption{A scatter plot}
\end{figure}
```

This produces Figure 11.

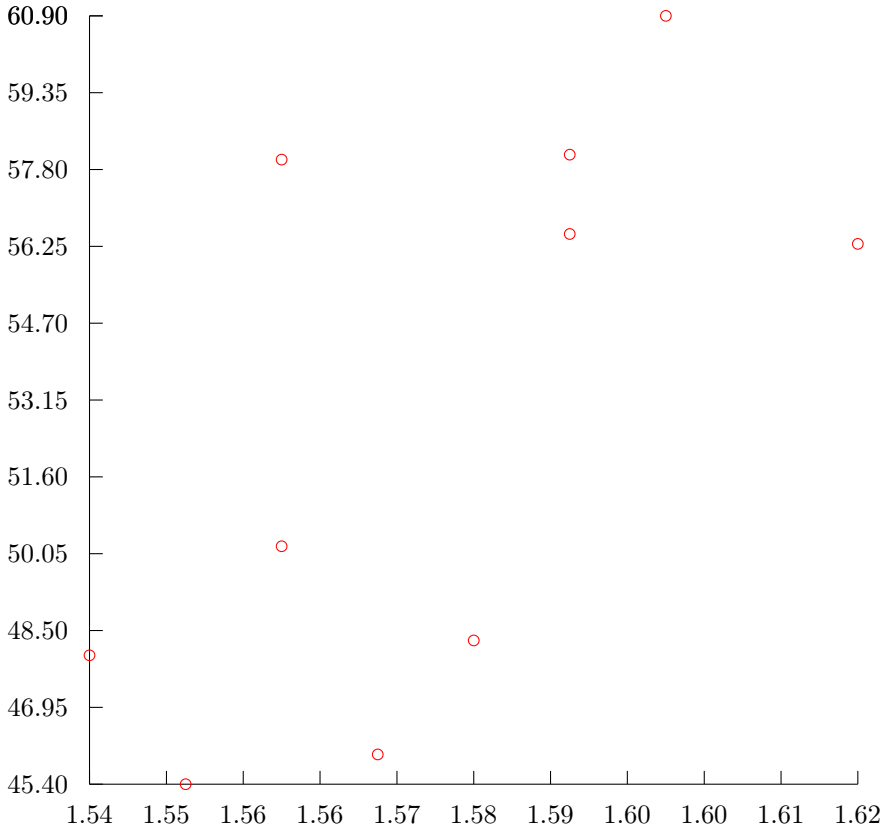

Figure 11: A scatter plot

Alternatively, you can use the style setting to change it into a line plot:

```
\begin{figure}[htbp]
\centering
\DTLplot{groupa}{x=Height,y=Weight,style=lines}
\caption{A line plot}
\end{figure}
```

This produces Figure 12.

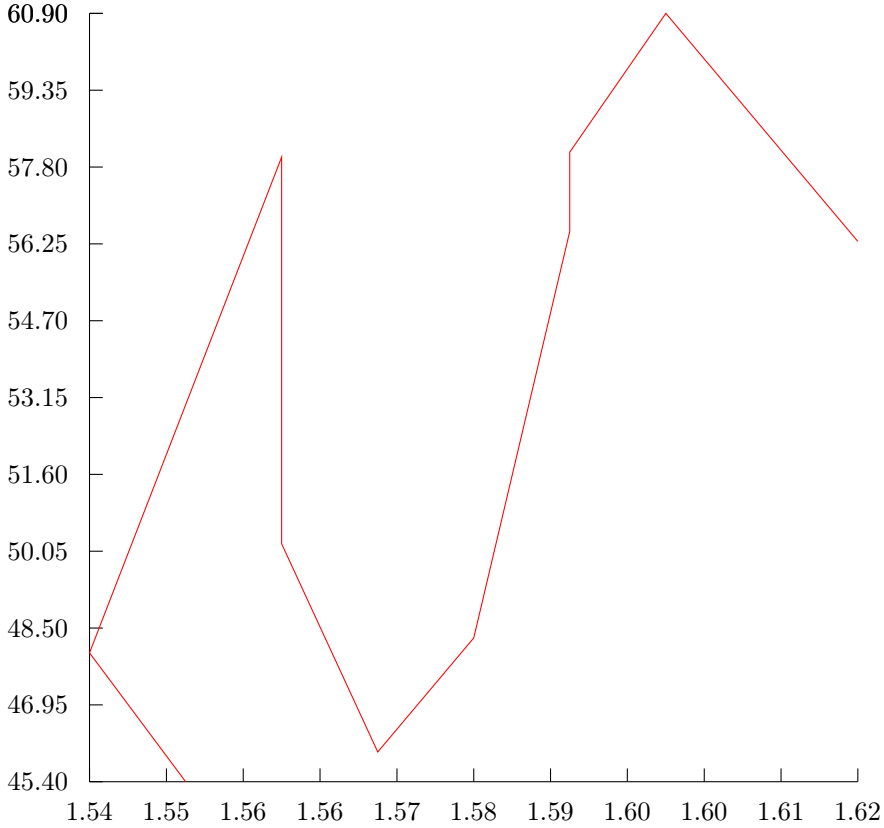

Figure 12: A line plot

Example 25 (Plotting Multiple Data Sets)

In this example, I shall use the database called `groupa` defined in example 24, and another database called `groupb` which is loaded from the file `groupb.csv` which contains the following:

```
Height,Weight
1.54,48.4
1.54,42.0
1.55,64.0
1.56,58.2
1.56,49.0
1.57,40.3
1.58,51.5
1.58,63.1
1.59,74.9
1.59,59.3
```

First load this into a database called `groupb`:

```
\DTLloaddb{groupb}{groupb.csv}
```

I can now plot both groups in the same graph, but I want a smaller graph than Figure 11 and Figure 12, so I am going to set the plot width and height to 3in:

```
\begin{figure}[htbp]
\centering
\DTLplot{groupa,groupb}{x=Height,y=Weight,width=3in,height=3in}
\caption{A scatter plot}
\end{figure}
```

This produces Figure 13.

Now let's add a legend using the legend setting, with the legend labels `Group A` and `Group B`, and set the x tick intervals using xticpoints setting. I am also going to set the x axis label to `Height (m)` and the y axis label to `Weight (kg)`, and place a box around the plot.

```
\begin{figure}[htbp]
\centering
\DTLplot{groupa,groupb}{x=Height,y=Weight,
width=3in,height=3in,legend,legendlabels={Group A,Group B},
xlabel={Height (m)},ylabel={Weight (kg)},box,
xticpoints={1.54,1.55,1.56,1.57,1.58,1.59,1.60,1.61,1.62}}
\caption{A scatter plot}
\end{figure}
```

This produces Figure 14.

7.1 Adding Information to the Plot

The datatool package provides two hooks used at the beginning and end of the tikzpicture environment:

`\DTLplotatbegintikz`

```
\DTLplotatbegintikz
```

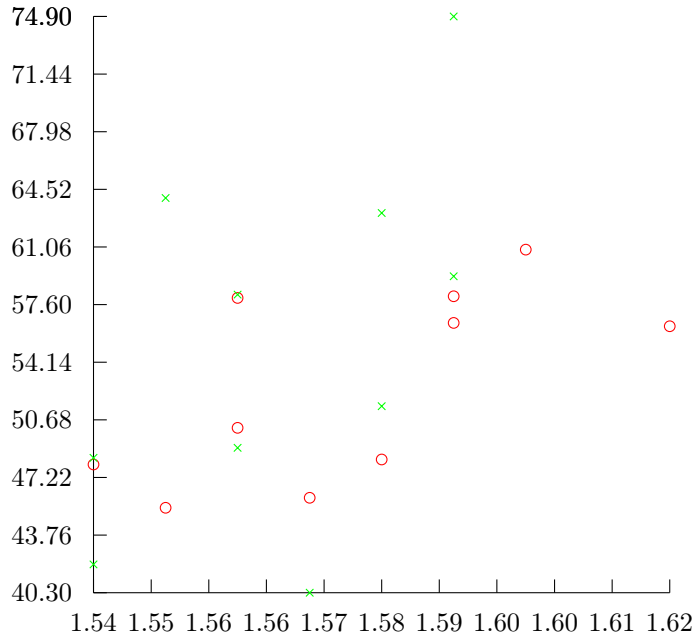

Figure 13: A scatter plot

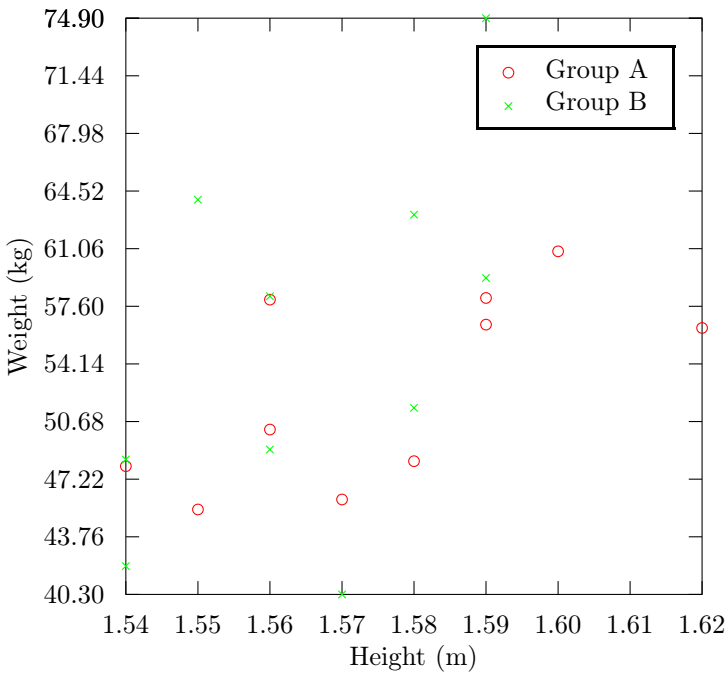

Figure 14: A scatter plot

and

\DTLplotatendtikz

```
\DTLplotatendtikz
```

They are both defined to do nothing by default, but can be redefined to add commands to the image. The unit vectors are set prior to using these hooks, so you can use the same co-ordinates as those in the data sets.

\DTLaddtoplotlegend

```
\DTLaddtoplotlegend{⟨marker⟩}{⟨line style⟩}{⟨text⟩}
```

This adds a new row to the plot legend where ⟨*marker*⟩ is code to produce the marker, ⟨*line style*⟩ is code to set the line style and ⟨*text*⟩ is a textual label. You can use `\relax` to suppress the marker or line. For example:

```
\DTLaddtoplotlegend{\pgfuseplotmark{x}}{\relax}{Some Data}
```

Note that the legend is plotted before `\DTLplotatendtikz`, so if you want to add information to the legend you will need to do the in `\DTLplotatstarttikz`.

Example 26 (Adding Information to a Plot)

Returning to the plots created in example 25, suppose I now want to annotate the plot, say I want to draw your notice to a particular point, say the point (1.58,48.3), then I can redefine `\DTLplotatendtikz` to draw an annotated arrow to that point:

```
\renewcommand*{\DTLplotatendtikz}{%
\draw[<-,line width=1pt] (1.58,48.3) -- (1.6,43)
 node[below]{interesting point};
}
```

So Figure 14 now looks like Figure 15. (Obviously, `\DTLplotatendtikz` needs to be redefined before using `\DTLplot`.)

7.2 Global Plot Settings

7.2.1 Lengths

This section describes the lengths that govern the appearance of the plot created using `\DTLplot`. These lengths can be changed using `\setlength`.

\DTLplotwidth

```
\DTLplotwidth
```

This length governs the length of the x axis. Note that the plot width does not include any outer tick marks or labels. The default value is 4in.

\DTLplotheight

```
\DTLplotheight
```

This length governs the length of the y axis. Note that the plot height does not include any outer tick marks or labels. The default value is 4in

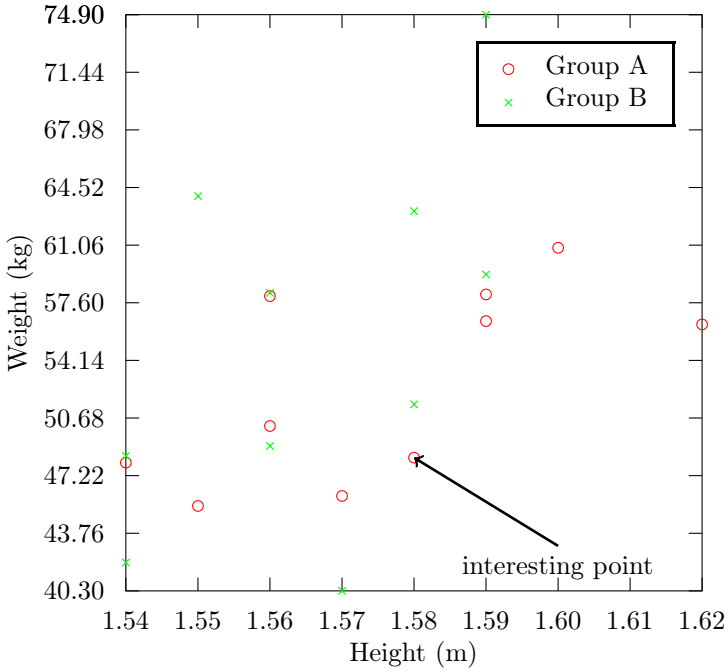

Figure 15: A scatter plot

`\DTLticklength`

```
\DTLticklength
```

This governs the length of the tick marks. The default value is 5pt.

`\DTLminorticklength`

```
\DTLminorticklength
```

This governs the length of the minor tick marks. The default value is 2pt.

`\DTLticklabeloffset`

```
\DTLticklabeloffset
```

This governs the distance from the axis to the tick labels. The default value is 8pt.

`\DTLmintickgap`

```
\DTLmintickgap
```

This is the minimum distance allowed between tick marks. If the plot width or height is less than this distance there will only be tick marks at either end of the axis. The default value is 20pt.

`\DTLlegendxoffset`

```
\DTLlegendxoffset
```

This is the horizontal distance from the border of the plot to the outer border of the legend. The default value is 10pt.

\DTLlegendyoffset

```
\DTLlegendyoffset
```

This is the vertical distance from the border of the plot to the outer border of the legend. The default value is 10pt.

7.2.2 Counters

These counters govern the appearance of plots created using `\DTLplot`. The value of the counters can be changed using `\setcounter`.

```
DTLplotroundXvar
```

Unless you specify your own tick labels, the x tick labels will be given by the tick points rounded to $\langle n \rangle$ digits after the decimal point, where $\langle n \rangle$ is the value of the counter `DTLplotroundXvar`.

```
DTLplotroundYvar
```

Unless you specify your own tick labels, the y tick labels will be given by the tick points rounded to $\langle n \rangle$ digits after the decimal point, where $\langle n \rangle$ is the value of the counter `DTLplotroundYvar`.

7.2.3 Macros

These macros govern the appearance of plots created using `\DTLplot`. They can be changed using `\renewcommand`.

\DTLplotmarks

```
\DTLplotmarks
```

This must be a comma separated list of `pgf` code to create the plot marks. `\DTLplot` cycles through this list for each database listed. The `pgf` package provides convenient commands for generating plots using `\pgfuseplotmark`. See the `pgf` manual for more details.

\DTLplotmarkcolors

```
\DTLplotmarkcolors
```

This must be a comma separated list of defined colours to apply to the plot marks. `\DTLplot` cycles through this list for each database listed. If this macro is set to empty, the current colour will be used instead.

\DTLplotlines

```
\DTLplotlines
```

This must be a comma separated list of `pgf` code to set the style of the plot lines. `\DTLplot` cycles through this list for each database listed. Dash patterns can be set using `\pgfsetdash`, see the `pgf` manual for more details. If `\DTLplotlines` is set to empty the current line style will be used instead.

\DTLplotlinecolors

```
\DTLplotlinecolors
```

This must be a comma separated list of defined colours to apply to the plot lines. \DTLplot cycles through this list for each database listed. If this macro is set to empty, the current colour will be used instead. The default is the same as \DTLplotmarkcolors.

\DTLXAxisStyle

```
\DTLXAxisStyle
```

This governs the style of the x axis. It is passed as the optional argument to the TikZ \draw command. By default it is just - which is a solid line style with no start or end arrows. The x axis line starts from the bottom left corner of the plot and extends to the bottom right corner of the plot. So if you want the x axis to have an arrow head at the right end, you can do:

```
\renewcommand*{\DTLXAxisStyle}{->}
```

\DTLYAxisStyle

```
\DTLYAxisStyle
```

This governs the style of the y axis. It is analogous to \DTLXAxisStyle described above.

\DTLmajorgridstyle

```
\DTLmajorgridstyle
```

This specifies the format of the major grid lines. It may be set to any TikZ setting that you can pass to the optional argument of \draw. The default value is color=gray,- which indicates a grey solid line.

\DTLminorgridstyle

```
\DTLminorgridstyle
```

This specifies the format of the minor grid lines. It may be set to any TikZ setting that you can pass to the optional argument of \draw. The default value is color=gray,loosely dotted which indicates a grey dotted line.

\DTLformatlegend

\DTLformatlegend{⟨*legend*⟩}

This formats the entire legend, which is passed as the argument. The default is to set the legend with a white background, a black frame.

7.3 Adding to a Plot Stream

\DTLplotstream

\DTLplotstream[⟨*condition*⟩]{⟨*db name*⟩}{⟨*x key*⟩}{⟨*y key*⟩}

This adds points to a stream from the database called ⟨*db name*⟩ where the x co-ordinates are given by the key ⟨*x key*⟩ and the y co-ordinates are given by the key ⟨*y key*⟩. (\DTLconverttodecimal is used to convert locale dependent values to a standard decimal that is recognised by the pgf package.) The optional argument ⟨*condition*⟩ is the same as that for \DTLforeach.

Example 27 (Adding to a Plot Stream)

Suppose you have a CSV file called `data.csv` containing the following:

```
x,y
0,0
1,1
2,0.5
1.5,0.3
```

First load the file into a database called `data`:

```
\DTLloaddb{data}{data.csv}
```

Now create a figure containing this data:

```
\begin{figure}[tbhp]
\centering
\begin{tikzpicture}
\pgfplothandlermark{\pgfuseplotmark{o}}
\pgfplotstreamstart
\DTLplotstream{data}{x}{y}%
\pgfplotstreamend
\pgfusepath{stroke}
\end{tikzpicture}
\caption{Adding to a plot stream}
\end{figure}
```

This produces Figure 16.

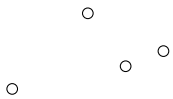

Figure 16: Adding to a plot stream

Example 28 (Plotting Multiple Keys in the Same Database)

Suppose I have conducted two time to growth experiments. For each experiment, I have recorded the log count at set times, and I have recorded this information in the same data file called, say, `growth.csv` which contains the following:

```
Time,Experiment 1,Experiment 2
0,3.73,3.6
23,3.67,3.7
60,4.9,3.8
```

I can load the data into a database using:

```
\DTLloaddb{growth}{growth.csv}
```

However, I'd like to plot both results on the same graph. Since they are contained in the same database, I can't use the method I used in example 25. Instead I can use a combination of \DTLplot and \DTLplotstream:

```
\begin{figure}[tbhp]
\centering
 % computer bounds
\DTLminforkeys{growth}{Time}{\minX}
\DTLminforkeys{growth}{Experiment 1,Experiment 2}{\minY}
\DTLmaxforkeys{growth}{Time}{\maxX}
\DTLmaxforkeys{growth}{Experiment 1,Experiment 2}{\maxY}
 % round x tick labels
\setcounter{DTLplotroundXvar}{0}
 % redefine \DTLplotatbegintikz to plot the data for Experiment 1
\renewcommand*{\DTLplotatbegintikz}{%
 % set plot mark
\pgfplothandlermark{\color{green}\pgfuseplotmark{x}}
 % start plot stream
\pgfplotstreamstart
 % add data from Experiment 1 to plot stream
\DTLplotstream{growth}{Time}{Experiment 1}%
 % end plot stream
\pgfplotstreamend
 % stroke path
\pgfusepath{stroke}
 % add information to legend (no line is require so use \relax)
\DTLaddtoplotlegend{\color{green}%
\pgfuseplotmark{x}}{\relax}{Experiment 1}
}
 % now plot the data for Experiment 2
\DTLplot{growth}{x=Time,y=Experiment 2,legend,
width=3in,height=3in,bounds={\minX,\minY,\maxX,\maxY},
xlabel={Time},ylabel={Log Count},
legendlabels={Experiment 2}}
\caption{Time to growth data}
\end{figure}
```

This produces Figure 17. Notes:

- I redefined \DTLplotatbegintikz in order to add the new plot to the legend, since \DTLplotatendtikz is used after the legend is plotted. The x and y unit vectors are set before \DTLplotatbegintikz so I don't need to worry about the co-ordinates.
- I set the counter DTLplotroundXvar to zero otherwise the x axis would have looked too cluttered.
- I have used \DTLminforkeys and \DTLmaxforkeys to determine the bounds since \DTLplot won't take the data for Experiment 1 into account when computing the bounds.

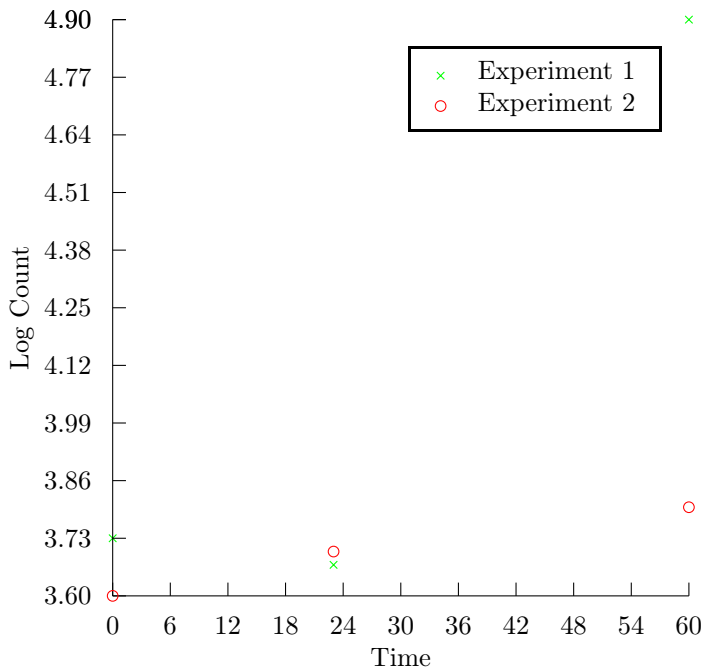

Figure 17: Time to growth data

8 Bar Charts (databar package)

The databar package provides commands for creating bar charts. It is not loaded by the datatool package, so if you want to use it you will need to load it explicitly using `\usepackage{databar}`. You must also have the pgf package installed.

Bar charts can either be vertical or horizontal, the default is vertical. In this section the x axis refers to the horizontal axis when plotting a vertical bar chart and to the vertical axis when plotting a horizontal bar chart. The x axis units are in increments of one bar. The y axis refers to the vertical axis when plotting a vertical bar chart and to the horizontal axis when plotting a horizontal bar chart. The y axis uses the same co-ordinates as the data. The bars may have an upper and lower label. In a vertical bar chart, the lower label is placed below the x axis and the upper label is placed above the top of the bar. In a horizontal bar chart, the lower label is placed to the left of the x axis and the upper label is placed to the right of the end of the bar. (This is actually a misnomer as it is possible for the "upper" label to be below the "lower" label if a bar has a negative value, however the bars are considered to be anchored on the x axis, and the other end of the bar is considered to be the "upper" end, regardless of its direction.)

The databar package options are as follows:

color Created coloured bar charts (default).

gray Created grey scale bar charts.

vertical Created vertical bar charts (default).

horizontal Created horizontal bar charts.

\DTLbarchart

\DTLbarchart[⟨*condition*⟩]{⟨*db name*⟩}{⟨*settings*⟩}{⟨*values*⟩}

\DTLmultibarchart

\DTLmultibarchart[⟨*condition*⟩]{⟨*db name*⟩}{⟨*settings*⟩}{⟨*values*⟩}

These commands both create a bar chart from the information in the database ⟨*db name*⟩, where ⟨*condition*⟩ is the same as the optional argument for \DTLforeach described in subsection 5.4, and ⟨*values*⟩ is the same as the penultimate argument of \DTLforeach. The ⟨*settings*⟩ argument is a ⟨*setting*⟩=⟨*value*⟩ list of settings. The first command, \DLTbarchart, will draw a bar chart for a given column of data in the database, whereas the second command, \DTLmultibarchart, will draw a bar chart that is divided into groups of bars where each bar within a group represents data from several columns of a given row in the database.

The variable setting is required for \DTLbarchart and the variables, the other settings are optional (though some may only be used for one of \DTLbarchart and \DLTmultibarchart), and are as follows:

variable This specifies the control sequence to use that contains the value used to construct the bar chart. The control sequence must be one of the control sequences to appear in the assignment list ⟨*values*⟩. This setting is required for \DTLbarchart, and is unavailable for \DTLmultibarchart.

variables This specifies a list of control sequences to use which contain the values used to construct the bar chart. Each control sequence must be one of the control sequences to appear in the assignment list ⟨*values*⟩. This setting is required for \DTLmultibarchart, and is unavailable for \DTLbarchart.

max This specifies the maximum value on the y axis. (This should be a standard decimal value.)

length This specifies the overall length of the y axis, and must be a dimension.

maxdepth This must be a zero or negative number. It specifies the maximum depth of the y axis. (This should be a standard decimal value.)

axes This setting specifies which axes to display. This may take one of the following values: both, x, y or none.

barlabel This setting specifies the lower bar label. When used with \DTLmultibarchart it indicates the group label.

multibarlabels This setting should contain a comma separated list of labels for each bar within a group for \DTLmultibarchart. This setting is not available for \DTLbarchart.

upperbarlabel This setting specifies the upper bar label. This setting is not available for \DTLmultibarchart.

uppermultibarlabels This setting must be a comma separated list of upper bar labels for each bar within a group. This setting is not available for \DTLbarchart.

yticpoints This must be a comma separated list of tick locations for the y axis. (These should be standard decimal values.) This setting overrides yticgap.

yticgap This specifies the gap between the y tick marks. (This should be a standard decimal value.)

yticlabels This must be a comma separated list of tick labels for the y axis.

ylabel This specifies the label for the y axis.

groupgap This specifies the gap between groups when using `\DTLmultibarchart`. This value is given as a multiple of the bar width. The default value is 1, which indicates a gap of one bar width. This setting is not available for `\DTLbarchart`.

verticalbars This is a boolean setting, so it can only take the values `true` (do a vertical bar chart) or `false` (do a horizontal bar chart). If the value is omitted, `true` is assumed.

Example 29 (A Basic Bar Chart)

Recall example 18 defined a database called `fruit`. This example will be using that database to plot a bar chart. The following plots a basic vertical bar chart:

```
\begin{figure}[htbp]
\centering
\DTLbarchart{variable=\theQuantity}{fruit}{\theQuantity=Quantity}
\caption{A basic bar chart}
\end{figure}
```

This produces Figure 18.

8.1 Changing the Appearance of a Bar Chart

`\DTLbarchartlength`

`\DTLbarchartlength`

This specifies the total length of the y axis. You must use `\setlength` to change this value. The default value is 3in.

`\DTLbarwidth`

`\DTLbarwidth`

This specifies the width of each bar. You must use `\setlength` to change this value. The default value is 1cm.

`\DTLbarlabeloffset`

`\DTLbarlabeloffset`

This specifies the distance from the x axis to the lower bar label. You must use `\setlength` to change this value. The default value is 10pt.

`DTLbarroundvar`

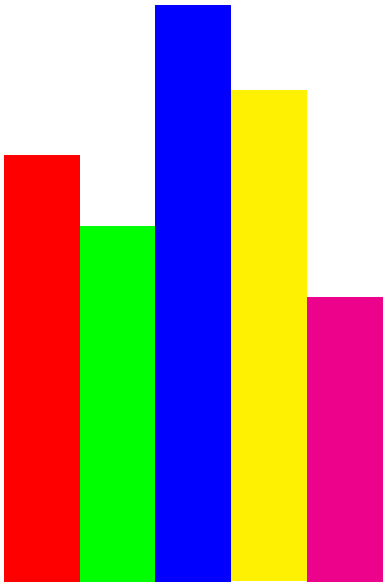

Figure 18: A basic bar chart

The y tick labels are rounded to $\langle n \rangle$ digits after the decimal point, where $\langle n \rangle$ is given by the value of the counter `DTLbarroundvar`. You must use `\setcounter` to change this value.

`\DTLsetbarcolor`

```
\DTLsetbarcolor{⟨n⟩}{⟨color⟩}
```

This sets the $\langle n \rangle$th bar colour to $\langle color \rangle$. Only the first eight bars have a colour defined by default. If you need more than eight bars, you will need to define more bar colours. It is recommended that you set the colour of each bar to correspond with whatever the bar represents.

`\DTLdobarcolor`

```
\DTLdobarcolor{⟨n⟩}
```

This sets the current colour to the colour of the $\langle n \rangle$th bar.

`\DTLbaroutlinecolor`

```
\DTLbaroutlinecolor
```

This macro contains the colour of the bar outlines. This defaults to `black`.

`\DTLbaroutlinewidth`

```
\DTLbaroutlinewidth
```

This length specifies the line width for the bar outlines. If it is 0pt, the outline is not drawn. The default value is 0pt.

`\DTLbaratbegintikz`

```
\DTLbaratbegintikz
```

This specifies any additional commands to add to the start of the plot. It defaults to nothing, and is called after the unit vectors are set.

\DTLbaratendtikz

```
\DTLbaratendtikz
```

This specifies any additional commands to add to the end of the plot. It defaults to nothing.

\DTLeverybarhook

```
\DTLeverybarhook
```

\DTLstartpt
\DTLmidpt
\DTLendpt

The specifies code to apply at every bar. Within the definition of `\DTLeverybarhook` you can use the commands `\DTLstartpt` (the start of the bar), `\DTLmidpt` (the mid point of the bar) and `\DTLendpt` (the end of the bar). For example (using the earlier `fruit` database):

```
\renewcommand*{\DTLeverybarhook}{%
\pgftext[at=\DTLmidpt]{\insertName\space(\insertValue)}%
}
\DTLbarchart{variable=\insertValue,axes=both,
ylabel=Quantity,max=50,verticalbars=false
}%
{fruit}{\insertValue=Value,\insertName=Name}
```

This puts the name followed by the quantity in brackets in the middle of the bar.

\ifDTLverticalbars

```
\ifDTLverticalbars
```

This conditional governs whether the chart uses vertical or horizontal bars.

\DTLbarXlabelalign

```
\DTLbarXlabelalign
```

This specifies the text alignment of the lower bar labels. This defaults to `left,rotate=-90` if you use the vertical package option or the verticalbars setting, and defaults to `right` if you use the horizontal package option or the verticalbars=`false` setting.

\DTLbarYticklabelalign

```
\DTLbarYlabelalign
```

This specifies the text alignment of the y axis labels. This defaults to `right` for vertical bar charts and `center` for horizontal bar charts.

\DTLbardisplayYticklabel

```
\DTLbardisplayYticklabel{⟨text⟩}
```

This specifies how to display the y tick label. The argument is the tick label.

\DTLdisplaylowerbarlabel

```
\DTLdisplaylowerbarlabel{⟨text⟩}
```

This specifies how to display the lower bar label for `\DTLbarchart` and the lower bar group label for `\DTLmultibarchart`. The argument is the label.

\DTLdisplaylowermultibarlabel

\DTLdisplaylowermultibarlabel{⟨*text*⟩}

This specifies how to display the lower bar label for \DTLmultibarchart. The argument is the label. This command is ignored by \DTLbarchart.

\DTLdisplayupperbarlabel

\DTLdisplayupperbarlabel{⟨*text*⟩}

This specifies how to display the upper bar label for \DTLbarchart and the upper bar group label for \DTLmultibarchart. The argument is the label.

\DTLdisplayuppermultibarlabel

\DTLdisplayuppermultibarlabel{⟨*text*⟩}

This specifies how to display the upper bar label for \DTLmultibarchart. The argument is the label. This command is ignored by \DTLbarchart.

Example 30 (A Labelled Bar Chart)

This example extends example 29 so that the chart is a bit more informative (which is after all the whole point of a chart). This chart now has a label below each bar, as well as a label above the bar. The lower label uses the value of the Name key, and the upper label uses the quantity. I have also set the outline width so each bar has a border.

```
\begin{figure}[htbp]
\setlength{\DTLbaroutlinewidth}{1pt}
\centering
\DTLbarchart{variable=\theQuantity,barlabel=\theName,%
upperbarlabel=\theQuantity}{fruit}{%
\theQuantity=Quantity,\theName=Name}
\caption{A bar chart}
\end{figure}
```

This produces Figure 19.

Example 31 (Profit/Loss Bar Chart)

Suppose I have a file called profits.csv that looks like:

```
Year,Profit
2000,\pounds2,535
2001,\pounds3,752
2002,-\pounds1,520
2003,\pounds1,270
```

First I can load this file into a database called profits:

```
\DTLloaddb{profits}{profits.csv}
```

Now I can plot the data as a bar chart:

```
\begin{figure}[htbp]
```

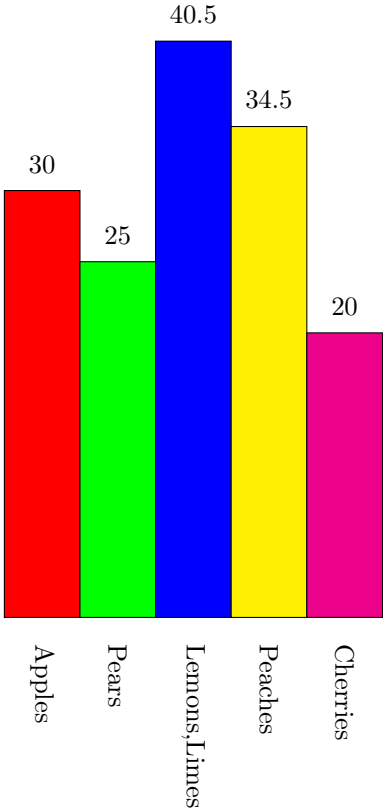

Figure 19: A bar chart

```
\centering
 % Set the width of each bar to 10pt
\setlength{\DTLbarwidth}{10pt}
 % Set the outline width to 1pt
\setlength{\DTLbaroutlinewidth}{1pt}
 % Round the $y$ tick labels to integers
\setcounter{DTLbarroundvar}{0}
 % Adjust the tick label offset
\setlength{\DTLticklabeloffset}{20pt}
 % Change the y tick label alignment
\renewcommand*{\DTLbarYticklabelalign}{left}
 % Rotate the y tick labels
\renewcommand*{\DTLbardisplayYticklabel}[1]{\rotatebox{-45}{#1}}
 % Set the bar colours depending on the value of \theProfit
\DTLforeach{profits}{\theProfit=Profit}{%
\ifthenelse{\DTLislt{\theProfit}{0}}
{\DTLsetbarcolor{\DTLcurrentindex}{red}}
{\DTLsetbarcolor{\DTLcurrentindex}{blue}}}
 % Do the bar chart
\DTLbarchart{variable=\theProfit,upperbarlabel=\theYear,
ylabel={Profit/Loss (\pounds)},verticalbars=false,
maxdepth=-2000,max=4000}{profits}
{\theProfit=Profit,\theYear=Year}
\caption{Profits for 2000--2003}
\end{figure}
```

This produces Figure 20. Notes:

1. This example uses `\rotatebox`, so the graphics or graphicx package is required.
2. The y tick labels are too wide to fit horizontally so they have been rotated to avoid overlapping with their neighbour.
3. Rotating the y tick labels puts them too close to the y axis, so `\DTLticklabeloffset` is made larger to compensate.
4. Remember not to use `\year` as an assignment command as this command already exists!
5. Before the bar chart is created I have iterated through the database, setting the bar colour to red or blue depending on the value of `\theProfit`.

Both `\DTLbarchart` and `\DTLmultibarchart` set the following macros, which may be used in `\DTLbaratbegintikz` and `\DTLbaratendtikz`:

`\DTLbarchartwidth`

`\DTLbarchartwidth`

This is the overall width of the bar chart. In the case of `\DTLbarchart` this is just the number of bars. In the case of `\DTLmultibarchart` it is computed as:

$$m \times n + (m - 1) \times g$$

where m is the number of bar groups (i.e. the number of rows of data), n is the number of bars within a group (i.e. the number of commands listed in the variables) setting and g is the group gap (as specified by the groupgap setting).

\DTLnegextent

```
\DTLnegextent
```

This is set to the negative extent of the bar chart. (This value may either be zero or negative, and corresponds to the maxdepth setting.)

\DTLbarmax

```
\DTLbarmax
```

This is set to the maximum extent of the bar chart. (This value corresponds to the max setting.)

Figure 20: Profits for 2000–2003

Example 32 (A Multi-Bar Chart)

This example uses the `marks` database described in example 13. Recall that this database stores student marks for three assignments. The keys for the assignment marks are `Assignment 1`, `Assignment 2` and `Assignment 3`, respectively. I can convert this data into a bar chart using the following:

```
\begin{figure}[htbp]
\centering
\DTLmultibarchart{variables={\assignI,\assignII,\assignIII},
barwidth=10pt,uppermultibarlabels={\assignI,\assignII,\assignIII},
barlabel={\firstname\ \surname}}{marks}{%
\surname=Surname,\firstname=FirstName,\assignI=Assignment 1,%
\assignII=Assignment 2,\assignIII=Assignment 3}
\caption{Student marks}
\end{figure}
```

This produces Figure 21. Notes:

1. I used `variables={\assignI,\assignII,\assignIII}` to set the variable to use for each bar within a group. This means that there will be three bars in each group.

2. I have set the bar width to 10pt, otherwise the chart will be too wide.

3. I used `uppermultibarlabels={\assignI,\assignII,\assignIII}` to set the upper labels for each bar within a group. This will print the assignment mark above the relevant bar.

4. I used `barlabel={\firstname\ \surname}` to place the student's name below the group corresponding to that student.

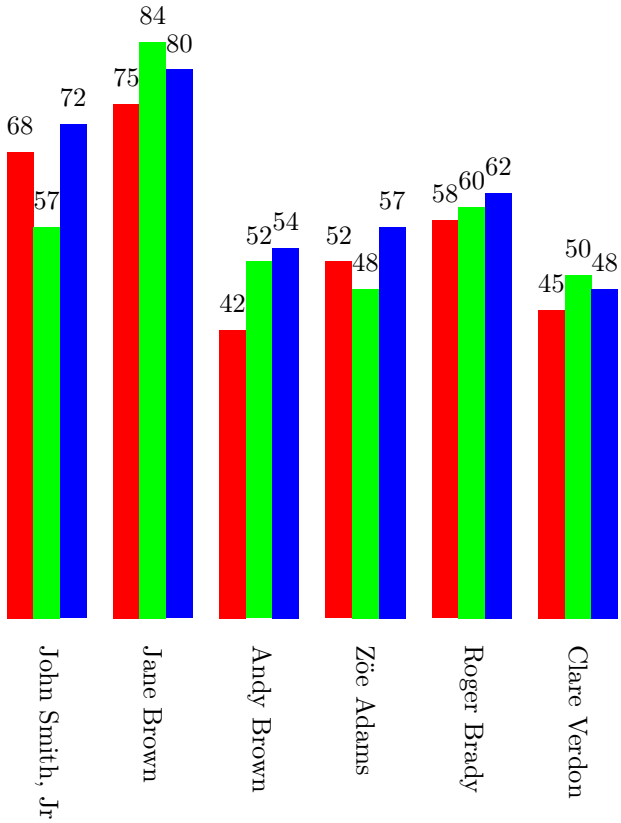

Figure 21: Student marks

Recall that example 13 computed the average score over for each student, and saved it with the key `Average`. This information can be added to the bar chart. It might also be useful to compute the average over all students and add this information to the chart. This is done as follows:

```
\begin{figure}[htbp]
\centering
% compute the overall mean
\DTLmeanforkeys{marks}{Average}{\overallmean}
% round it to 2 decimal places
\DTLround{\overallmean}{\overallmean}{2}
% draw a grey dotted line indicating the overall mean
% covering the entire width of the bar chart
\renewcommand*{\DTLbaratendtikz}{%
  \draw[lightgray,loosely dotted] (0,\overallmean) --
    (\DTLbarchartwidth,\overallmean)
    node[right,black]{Average (\overallmean)};}
% Set the lower bar labels to draw a brace across the current
% group, along with the student's name and average score
\renewcommand*{\DTLdisplaylowerbarlabel}[1]{%
\tikz[baseline=(current bounding box.center)]{
\draw[snake=brace,rotate=-90](0,0) -- (\DTLbargroupwidth,0);}
\DTLround{\theMean}{\theMean}{2}%
\shortstack{#1\\(Average: \theMean)}}
% draw the bar chart
\DTLmultibarchart{variables={\assignI,\assignII,\assignIII},
barwidth=10pt,uppermultibarlabels={\assignI,\assignII,\assignIII},
barlabel={\firstname\ \surname}}{marks}
{\surname=Surname,\firstname=FirstName,\assignI=Assignment 1,%
\assignII=Assignment 2,\assignIII=Assignment 3,\theMean=Average}
\caption{Student marks}
\end{figure}
```

which produces Figure 22. Notes:

1. I've used the TikZ snake library to create a brace, so I need to put

   ```
   \usetikzlibrary{snakes}
   ```

   in the preamble. See the pgf manual for more details on how to use this library.

2. I used `\DTLbargroupwidth` to indicate the width of each bar group.

3. I used `\DTLbarchartwidth` to indicate the width of the entire bar chart

---

9 Converting a BibTeX database into a datatool database (databib package)

The databib package provides the means of converting a BibTeX database into a datatool database. The database can then be sorted using `\DTLsort`, described

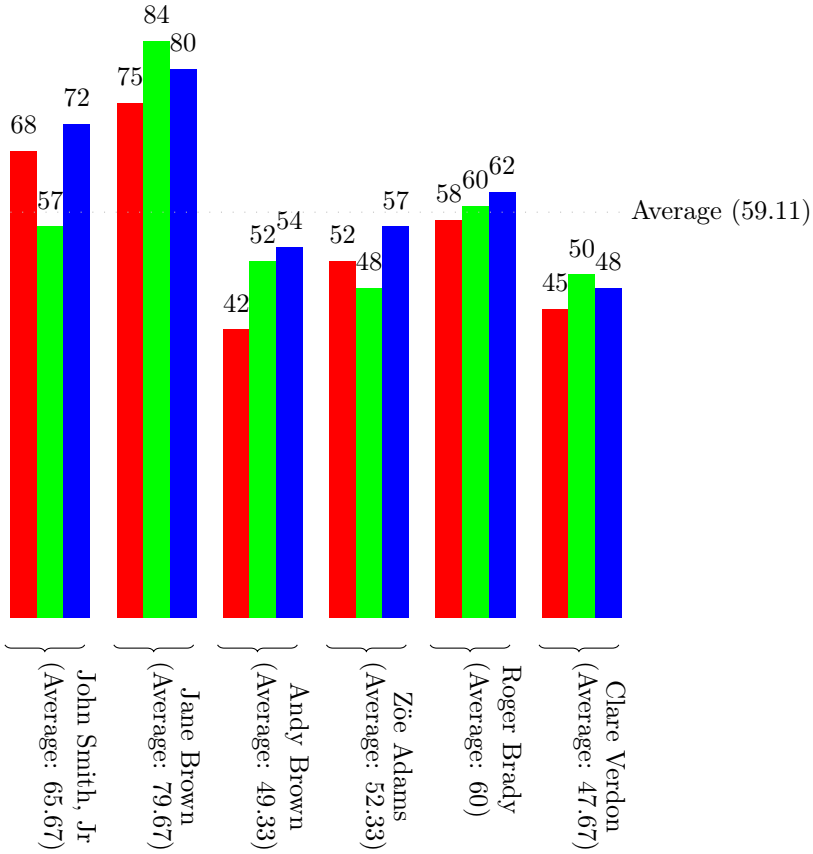

Figure 22: Student marks

in subsection 5.8. For example, you may want to sort the bibliography in reverse chronological order. Once you have sorted the bibliography, you can display it using `\DTLbibliography`, described in subsection 9.3, or you can iterate through the database using `\DTLforeachbib`, described in subsection 9.5.

Note that the databib package is not automatically loaded by datatool, so if you want to use it, you must load it using `\usepackage{databib}`.

⚠ The purpose of this package is to provide a means for authors to format their own bibliography style where there is no bibliography style file available that produces the desired results. The `\DTLsort` macro uses a much less efficient sorting algorithm than BibTeX, and loading the bibliography as a datatool database is much slower than loading a standard `bbl` file. If you have a large database, and you are worried that LaTeX may have become stuck, try using the verbose option to datatool or use the command `\dtlverbosetrue`. This will print informative messages to the console and transcript file, to let you know what's going on.

9.1 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 LaTeX. When you run BibTeX, you need to specify the name of the document's auxiliary file (without the `aux` extension). 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 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:

1. `latex mydoc`

   This writes the citation information to the auxiliary file. The bibliography currently doesn't exists, so it isn't displayed. Citations will appear in the document as ?? since the internal cross-references don't exist yet.

2. `bibtex mydoc`

   This reads the auxiliary file, and creates a file with the extension `bbl` which typically contains the typeset bibliography.

3. `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. `latex mydoc`

   The cross-referencing information can be read from the auxiliary file.

9.1.1 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_name⟩ = "⟨value⟩",
    ⋮
    ⟨field_name⟩ = "⟨value⟩"
}
```

Note that curly braces `{` and `}` may be used instead of `"` and `"`.

The entry type, given by ⟨*entry_type*⟩ above, indicates the type of document. This may be one of: `article`, `book`, `booklet`, `inbook`, `incollection`, `inproceedings`[5], `manual`, `mastersthesis`, `misc`, `phdthesis`, `proceedings`, `techreport` or `unpublished`.

The ⟨*cite_key*⟩ 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, 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`.

Author and editor names must be entered in one of the following ways:

1. ⟨*First names*⟩ ⟨*von part*⟩ ⟨*Surname*⟩, ⟨*Jr part*⟩

   The ⟨*von part*⟩ is optional and is identified by the name(s) starting with lowercase letters. 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 {\uppercase{d}e La} Cruz"
   ```

[5] Note that `conference` is a synonym for `inproceedings`.

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 `{\uppercase{d}e La}` will trick BibTeX into thinking that it starts with a lower case 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 Samarin"
```

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. You should always 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. For further information, see [1].

9.2 Loading a databib database

The databib package always requires the `databib.bst` bibliography style file (which is supplied with this bundle). 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`

`\DTLloadbbl[`⟨*bbl name*⟩`]{`⟨*db name*⟩`}{`⟨*bib list*⟩`}`

This command performs several functions:

1. it writes the following line in the auxiliary file:

   `\bibstyle{databib}`

   which tells BibTeX to use the `databib.bst` BibTeX style file,

2. it writes `\bibdata{`⟨*bib list*⟩`}` to the auxiliary file, which tells BibTeX which `bib` files to use,

3. it creates a datatool database called ⟨*db name*⟩,

4. it loads the file ⟨*bbl name*⟩ if it exists. (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 run your document through LaTeX (or PDFLaTeX) and then run BibTeX on the auxiliary file, as described in subsection 9.1. 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 LaTeX your document, this file will be read, and the information will be added to ⟨*db name*⟩.

⚠ Note that `\DTLloadbbl` doesn't generate any text. Once you have loaded the data, you can display the bibliography uses `\DTLbibliography` (described below) or you can iterate through it using `\DTLforeachbibentry` described in subsection 9.5.

Note that the `databib.bst` BibTeX style file provides the following additional fields: `isbn`, `doi`, `pubmed`, `url` and `abstract`. However these fields 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 subsubsection 9.4.1).

9.3 Displaying a databib database

A databib database which has been loaded using `\DTLloadbbl` (described in subsection 9.2) can be displayed using:

`\DTLbibliography`

`\DTLbibliography[`⟨*conditions*⟩`]{`⟨*db name*⟩`}`

where ⟨*db name*⟩ is the name of the database.

Within the optional argument ⟨*condition*⟩, you may use any of the commands that may be used within the optional argument of `\DTLforeach` *In addition*, you may use the following commands:

\DTLbibfieldexists

\DTLbibfieldexists{⟨*field label*⟩}

This tests whether the field with the given label exists for the current entry. The field label may be one of: Address, Author, BookTitle, Chapter, Edition, Editor, HowPublished, Institution, Journal, Key, Month, Note, Number, Organization, Pages, Publisher, School, Series, Title, Type, Volume, Year, ISBN, DOI, PubMed, Abstract or Url.

For example, suppose you have loaded a databib database called mybib using \DTLloadbbl (described in subsection 9.2) then the following bibliography will only include those entries which have a Year field:

```
\DTLbibliography[\DTLbibfieldexists{Year}]{mybib}
```

\DTLbibfieldiseq

\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

\DTLbibfieldcontains{⟨*field label*⟩}{⟨*sub string*⟩}

This tests whether the value of the field given by ⟨*field label*⟩ contains ⟨*sub string*⟩. 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

\DTLbibfieldislt{⟨*field label*⟩}{⟨*value*⟩}

This tests whether the value of the field given by ⟨*field label*⟩ is 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

\DTLbibfieldisle{⟨*field label*⟩}{⟨*value*⟩}

This tests whether the value of the field given by ⟨*field label*⟩ is less than or equal to ⟨*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 or equal to 1983:

```
\DTLbibliography[\DTLbibfieldisle{Year}{1983}]{mybib}
```

\DTLbibfieldisgt

\DTLbibfieldisgt{⟨*field label*⟩}{⟨*value*⟩}

This tests whether the value of the field given by ⟨*field label*⟩ is greater 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 greater than 1983:

```
\DTLbibliography[\DTLbibfieldisgt{Year}{1983}]{mybib}
```

\DTLbibfieldisge

\DTLbibfieldisge{⟨*field label*⟩}{⟨*value*⟩}

This tests whether the value of the field given by ⟨*field label*⟩ is greater than or equal to ⟨*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 greater than or equal to 1983:

```
\DTLbibliography[\DTLbibfieldisge{Year}{1983}]{mybib}
```

Note that `\DTLbibliography` uses `\DTLforeachbibentry` (described in subsection 9.5) so you may also use test the value of the counter `DTLbibrow` within ⟨*conditions*⟩. You may also use the boolean commands defined by the ifthen package, such as `\not`.

Example 33 (Creating a list of 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 2000. I have a file called `nlct.bib` which contains all my publications which I keep in the directory `$HOME/texmf/bibtex/bib/`. I could look through this file, work out the labels for all the publications whose year field is greater or equal to 2000, 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 create the following document:

```
\documentclass{article}
\usepackage{databib}
\begin{document}
\nocite{*}
\DTLloadbbl{mybib}{nlct}
\DTLsort{Year=descending,Month=descending}{mybib}
\DTLbibliography[\DTLbibfieldisge{Year}{2000}]{mybib}
\end{document}
```

Suppose I save this file as `mypubs.tex`, then I need to do:

```
latex mypubs
bibtex mypubs
latex mypubs
```

Notes:

1. `\nocite{*}` is used to add all the citations in the bibliography file (`nlct.bib` in this case) to the databib database.

2. `\DTLloadbbl{mybib}{nlct}` does the following:

   (a) 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`.

   (b) writes the line

   ```
   \bibdata{nlct}
   ```

   to the auxiliary file. This tells BibTeX that the bibliography data is stored in the file `nlct.bib`. Since I have placed this file in TeX's search path, BibTeX will be able to find it.

   (c) creates a datatool database called `mybib`.

   (d) if the `bbl` file (`mypubs.bbl` in this example) exists, it loads this file (which adds the bibliography data to the database), otherwise it does nothing further.

3. In my BibTeX database (`nlct.bib` in this example), I have remembered to use the BibTeX month macros: `jan`, `feb` etc. This means that the months are stored in the database in the form `\DTLmonthname{`⟨*nn*⟩`}`, where ⟨*nn*⟩ is a two digit number from 01 to 12. `\DTLsort` ignores command names when it compares strings, which means I can not only sort by year, but also by month[6].

4. Once I have loaded and sorted my database, I can then display it using `\DTLbibliography`. This uses the style given by the databib style package option, or the `\DTLbibliographystyle` command, both of which are described in subsection 9.4.

5. I have filtered the bibliography using the optional argument `[\DTLbibfieldisge{Year}{2000}]`, which checks if the year field of the current entry is greater than or equal to 2000. (Note that if an entry has no year field, the condition evaluates to false, and the entry will be omitted from the bibliography.)

6. If the bibliography database is large, sorting and creating the bibliography may take a while. Using databib is much slower than using a standard BibTeX style file.

Example 34 (Creating a list of my 10 most recent publications)

Suppose now my boss has asked me to produce a list of my ten most recent publications (in reverse chronological order). As in the previous example, I have a

[6] as long as I haven't put anything before the month name in the bibliography file, e.g. `month = 2 # apr` will sort by 2 03, instead of 03

file called `nlct.bib` which contains all my publications. I can create the required document as follows:

```
\documentclass{article}
\usepackage{databib}
\begin{document}
\nocite{*}
\DTLloadbbl{mybib}{nlct}
\DTLsort{Year=descending,Month=descending}{mybib}
\DTLbibliography[\value{DTLbibrow}<11]{mybib}
\end{document}
```

9.4 Changing the bibliography style

The style of the bibliography produced using `\DTLbibliography` depends on the `style` package option, or can be set using

`\DTLbibliographystyle`

`\DTLbibliographystyle{`⟨*style*⟩`}`

Note that this is *not* the same as `\bibliographystyle`, as the `databib` package uses its custom `databib.bst` bibliography style file.

Example:

```
\usepackage[style=plain]{databib}
```

This sets 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 BibTeX 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` (described in subsection 5.8).

9.4.1 Modifying an existing style

This section describes some of the commands which are used to format the bibliography. You can choose whichever predefined style best fits your required style, and then modify the commands described in this section. A description of the remaining commands not listed in this section can be found in subsection 14.4, subsection 14.5 and subsection 14.6.

`\DTLformatauthor`

`\DTLformatauthor{`⟨*von part*⟩`}{`⟨*surname*⟩`}{`⟨*jr part*⟩`}{`⟨*forenames*⟩`}`

`\DTLformateditor`

`\DTLformateditor{`⟨*von part*⟩`}{`⟨*surname*⟩`}{`⟨*jr part*⟩`}{`⟨*forenames*⟩`}`

These commands are used to format an author/editor's name, respectively. The list of authors and editors are stored in the `databib` database as a comma separated list of `{`⟨*von part*⟩`}{`⟨*surname*⟩`}{`⟨*jr part*⟩`}{`⟨*forenames*⟩`}` data. This ensures that when you sort on the Author or Editor field, the names will be sorted by the first author or editor's surname.

Within \DTLformatauthor and \DTLformateditor, you may use the following commands:

\DTLformatforenames

\DTLformatforenames{⟨*forenames*⟩}

This is used by the plain style to display the author's forenames[7].

\DTLformatabbrvforenames

\DTLformatabbrvforenames{⟨*forenames*⟩}

This is used by the abbrv style to display the author's initials (which are determined from ⟨*forenames*⟩). Note that if any of the authors has a name starting with an accent, the accented letter must be grouped in order for this command to work. For example:

```
author = "{\'E}lise {\"E}awyn Edwards",
```

The initials are formed using \DTLstoreinitials described in section 4, so if you want to change the way the initials are displayed (e.g. put a space between them) you will need to redefine the commands used by \DTLstoreinitials (such as \DTLbetweeninitials).

\DTLformatsurname

\DTLformatsurname{⟨*surname*⟩}

This displays its argument by default[8].

\DTLformatvon

\DTLformatvon{⟨*von part*⟩}

If the ⟨*von part*⟩ is empty, this command does nothing, otherwise it displays its argument followed by a non-breakable space.

\DTLformatjr

\DTLformatjr{⟨*jr part*⟩}

If the ⟨*jr part*⟩ is empty, this command displays nothing, otherwise it displays a comma followed by its argument[9].

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 by redefining \DTLformatauthor as follows:

```
\renewcommand*{\DTLformatauthor}[4]{%
\textsc{\DTLformatvon{#1}%
\DTLformatsurname{#2}\DTLformatjr{#3}},
\DTLformatforenames{#4}%
}
```

DTLmaxauthors

[7] It also checks whether ⟨*forenames*⟩ ends with a full stop using \DTLcheckendsperiod to prevent a sentence ending full stop from following an abbreviation full stop

[8] It also checks whether the surname ends with a full stop using \DTLcheckendsperiod

[9] again, it also checks ⟨*jr part*⟩ to determine if it ends with a full stop

The counter DTLmaxauthors is used to determine the maximum number of authors to display for a given entry. If the entry's author list contains more than that number of authors, \etalname is used, the definition of which is given in subsection 14.4. The default value of DTLmaxauthors is 10.

DTLmaxeditors

The DTLmaxeditors counter is analogous to the DTLmaxauthors counter. It is used to determine the maximum number of editor names to display. The default value of DTLmaxeditors is 10.

\DTLandlast
\DTLandnotlast
\DTLtwoand

Within a list of author or editor names, \DTLandlast is used between the last two names, otherwise \DTLandnotlast is used between names. However, if there are only two author or editor names, \DTLtwoand is used instead of \DTLandlast.

\DTLendbibitem

The command \DTLendbibitem is a hook provided to add additional information at the end of each bibliography item. This does nothing by default, but if you want to display the additional fields provided by the databib.bst style file, you can redefine \DTLendbibitem so that it displays a particular field, if it is defined. Within this command, you may use the commands \DTLbibfield, \DTLifbibfieldexist and \DTLifanybibfieldexist, which are described in subsection 9.5. 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}}{}}
```

(Note that \DTLpar needs to be used instead of \par.)

Example 35 (Compact bibliography)

Suppose I don't have much space in my document, and I need to produce a compact bibliography. Firstly, I can use the bibliography style abbrv, either through the package option:

```
\usepackage[style=abbrv]{databib}
```

or using:

```
\DTLbibliographystyle{abbrv}
```

Once I have set the style, I can further modify it thus:

```
\renewcommand*{\andname}{\&}
\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[10] and display the bibliography:

```
% create a database called mybib from the information given
% in mybib1.bib and mybib2.bib
\DTLloadbbl{mybib}{mybib1,mybib2}
% display the bibliography
\DTLbibliography{mybib}
```

Example 36 (Highlighting a given author)

Suppose my boss wants me to produce a list of all my publications (which I have stored in the file `nlct.bib`, as in example 33). Most of my publications have multiple co-authors, but suppose my boss would like me to highlight my name 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.)

```
\renewcommand*{\DTLformatauthor}[4]{%
{\DTLifstringeq{#2}{Talbot}{\bfseries }{}%
\DTLformatforenames{#4}
\DTLformatvon{#1}%
\DTLformatsurname{#2}%
\DTLformatjr{#3}}}
```

Notes:

1. I have used `\DTLifstringeq` (described in subsection 2.1) to perform the string comparison.

2. 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` or `\DTLisPrefix`, which are described in subsection 2.1 and subsection 2.2, respectively. Using the first approach I can do:

   ```
   \renewcommand*{\DTLformatauthor}[4]{%
   {\DTLifstringeq{#2}{Talbot}{\DTLifStartsWith{#4}{N}{\bfseries }{}}{}%
   \DTLformatforenames{#4}
   \DTLformatvon{#1}%
   \DTLformatsurname{#2}%
   \DTLformatjr{#3}}}
   ```

[10] I can load the bibliography earlier, but obviously the bibliography should only be displayed after the bibliography styles have been set, otherwise they will have no effect

Using the second approach I can do:

```
\renewcommand*{\DTLformatauthor}[4]{%
{\ifthenelse{\DTLiseq{#2}{Talbot}\and
\DTLisPrefix{#4}{N}}{\bfseries }{}%
\DTLformatforenames{#4}
\DTLformatvon{#1}%
\DTLformatsurname{#2}%
\DTLformatjr{#3}}}
```

3. I have used a group to localise the effect of `\bfseries`.

9.5 Iterating through a databib database

`\DTLbibliography` (described in subsection 9.3) 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:

`\DTLforeachbib`

`\DTLforeachbib[`⟨*condition*⟩`]{`⟨*db name*⟩`}{`⟨*text*⟩`}`

`\DTLforeachbib*`

`\DTLforeachbib*[`⟨*condition*⟩`]{`⟨*db name*⟩`}{`⟨*text*⟩`}`

This iterates through the databib database called ⟨*db name*⟩ and does ⟨*text*⟩ if ⟨*condition*⟩ is met. As with `\DTLforeach`, the starred version is read-only.

For each row of the database, the following commands are set:

`\DBIBcitekey`

- `\DBIBcitekey` This is the unique label which identifies the current entry (as used in the argument of `\cite` and `\nocite`).

`\DBIBentrytype`

- `\DBIBentrytype` This is the current entry type, 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`

`\DTLbibfield{`⟨*field label*⟩`}`

where ⟨*field label*⟩ may be one of: `Address`, `Author`, `BookTitle`, `Chapter`, `Edition`, `Editor`, `HowPublished`, `Institution`, `Journal`, `Key`, `Month`, `Note`, `Number`, `Organization`, `Pages`, `Publisher`, `School`, `Series`, `Title`, `Type`, `Volume`, `Year`, `ISBN`, `DOI`, `PubMed`, `Abstract` or `Url`.

You can determine if a field exists for a given entry using

`\DTLifbibfieldexists`

`\DTLifbibfieldexists{`⟨*field label*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

If the field given by ⟨*field label*⟩ exists for the current bibliography entry, it does ⟨*true part*⟩, otherwise it does ⟨*false part*⟩.

\DTLifbibanyfieldexists

\DTLifanybibfieldexists{⟨*field label list*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

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

\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.

\DTLcomputewidestbibentry

\DTLcomputewidestbibentry{⟨*conditions*⟩}{⟨*db name*⟩}{⟨*bib label*⟩}{⟨*cmd*⟩}

This computes the widest bibliography entry over all entries satisfying ⟨*conditions*⟩ in the database ⟨*db name*⟩, where the label is given by ⟨*bib label*⟩, and the result is stored in ⟨*cmd*⟩, which may then be used in the argument of the thebibliography environment.

The counter DTLbibrow keeps track of the current bibliography entry. This is reset at the start of each \DTLforeachbib and is incremented if ⟨*conditions*⟩ is met.

Within the optional argument ⟨*condition*⟩, you may use any of the commands that may be used within the optional argument of \DTLbibliography, described in subsection 9.3.

Example 37 (Separate List of Journals and Conference Papers)

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, all my publications are stored in the BibTeX database nlct.bib. The following creates the required document:

```
\documentclass{article}
\usepackage{databib}
\begin{document}
\nocite{*}
\DTLloadbbl{mybib}{nlct}

\renewcommand*{\refname}{Journal Papers}
\DTLcomputewidestbibentry{\equal{\DBIBentrytype}{article}}
{mybib}{J\theDTLbibrow}{\widest}

\begin{thebibliography}{\widest}
```

```
\DTLforeachbibentry[\equal{\DBIBentrytype}{article}]{mybib}{%
\bibitem[J\theDTLbibrow]{\DBIBcitekey} \DTLformatbibentry}
\end{thebibliography}

\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}

\end{document}
```

9.6 Multiple Bibliographies

It is possible to have more than one bibliography in a document, but it then becomes necessary to have a separate auxiliary file for each bibliography, and each auxiliary file must then be passed to BibTeX. In order to do this, you need to use

`\DTLmultibibs`

\DTLmultibibs{⟨*name list*⟩}

where ⟨*name list*⟩ is a comma separated list of names, ⟨*name*⟩. For each ⟨*name*⟩, this command creates an auxiliary file called ⟨*name*⟩`.aux` (note that this command may only be used in the preamble).

When you want to cite an entry for a given bibliography named in `\DTLmultibibs`, you must use:

`\DTLcite`

\DTLcite[⟨*text*⟩]{⟨*mbib*⟩}{⟨*cite key list*⟩}

This is analogous to `\cite[`⟨*text*⟩`]{`⟨*cite key list*⟩`}`, but writes the `\citation` command to ⟨*mbib*⟩`.aux` instead of to the document's main auxiliary file. It also ensures that the cross-referencing labels are based on ⟨*mbib*⟩, 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 `\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

`\DTLnocite`

\DTLnocite{⟨*mbib*⟩}{⟨*cite key list*⟩}

which is analogous to `\nocite{`⟨*cite key list*⟩`}`, where again the citation information is written to ⟨*mbib*⟩`.aux` instead of the document's main auxiliary file.

Note that for both `\DTLcite` and `\DTLnocite` the ⟨*mbib*⟩ part must be one of the names listed in `\DTLmultibibs`.

`\DTLloadmbbl`

\DTLloadmbbl{⟨*mbib*⟩}{⟨*db name*⟩}{⟨*bib list*⟩}

This is analogous to \DTLloadbbl{⟨*db name*⟩}{⟨*bib list*⟩} described in subsection 9.2. (Again ⟨*mbib*⟩ must be one of the names listed in \DTLmultibibs.) This creates a new datatool database called ⟨*db name*⟩ and loads the bibliography information from ⟨*mbib*⟩.bbl (if it exists).

\DTLmbibliography

\DTLmbibliography[⟨*condition*⟩]{⟨*mbib*⟩}{⟨*db name*⟩}

This is analogous to \DTLbibliography[⟨*condition*⟩]{⟨*db name*⟩}, but is required when displaying a bibliography in which elements have been cited using \DTLcite and \DTLnocite.

Example 38 (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 each of my research topics, with a mini-bibliography at the end of each section. As in the earlier examples, all my publications are stored in the file nlct.bib which is somewhere on TEX's path. 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.

```
\documentclass{article}
\usepackage{databib}
\DTLmultibibs{kernel,food}
\begin{document}
\section{Kernel methods}
In this section I'm going to describe some research work into
kernel methods, and in the process I'm going to cite some related
papers \DTLcite{kernel}{Cawley2007a,Cawley2006a}.

\DTLloadmbbl{kernel}{kernelDB}{nlct}
\DTLmbibliography{kernel}{kernelDB}

\section{Food research}

In this section I'm going to describe some research work
in the area of food safety, and in the process, I'm going
to cite some related papers \DTLcite{food}{Peck1999,Barker1999a}

\DTLloadmbbl{food}{foodDB}{nlct}
\DTLmbibliography{food}{foodDB}

\cite{*}
\renewcommand{\refname}{Complete List of Publications}
\DTLloadbbl{fullDB}{nlct}
\DTLbibliography{fullDB}
\end{document}
```

Notes:

1. This will create the files kernel.aux and food.aux. These will have to be passed to BIBTEX, in addition to the documents main auxiliary file. So, if

my document is called `researchwork.tex`, then I need to do:

```
latex researchwork
bibtex researchwork
bibtex kernel
bibtex food
latex researchwork
latex researchwork
```

2. `\cite{*}` is used to add all the entries in the bib file to the main bibliography database. As before, `\DTLloadbbl` and `\DTLbibliography` are used to load and display the main bibliography.

⚠ Don't try to directly input the `.bbl` file using `\input` (or `\include`) instead of using `\DTLloadbbl` or `\DTLloadmbbl` as these commands store the name of the required database and initialise the database before loading the `.bbl` file. Similarly, don't just copy the contents of the `.bbl` file into your document without first defining the database using `\DTLnewdb` and setting `\DTLBIBdbname` to the name of the database.

10 datatool.sty

10.1 Package Declaration

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{datatool}[2009/07/13 v2.02 (NLCT)]
```

Load required packages:

```
\RequirePackage{xkeyval}
\RequirePackage{ifthen}
\RequirePackage{xfor}
\RequirePackage{fp}
\RequirePackage{substr}
\RequirePackage{etex}
```

10.2 Package Options

`\@dtl@separator` The data separator character (comma by default) is stored in `\@dtl@separator`. This is the separator used in external data files, not in the LaTeX code, which always uses a comma separator.

```
\newcommand*{\@dtl@separator}{,}
```

`\DTLsetseparator`

`\DTLsetseparator{`⟨*char*⟩`}`

The sets `\@dtl@separator`, and constructs the relevent macros that require this character to be hardcoded into their definition.

```
\newcommand*{\DTLsetseparator}[1]{%
\renewcommand*{\@dtl@separator}{#1}%
\@dtl@construct@lopoffs
}
```

\DTLsettabseparator

\DTLsettabseparator makes it easier to set a tab separator.

```
\begingroup
\catcode`\ 12
\gdef\DTLsettabseparator{%
  \catcode`\ 12
  \DTLsetseparator{ }%
}
\endgroup
```

\@dtl@delimiter

The data delimiter character (double quote by default) is stored in \@dtl@delimiter. This is used in external data files, not in the LaTeX code.

```
\begingroup
\catcode`\"12\relax
\gdef\@dtl@delimiter{"}
\endgroup
```

\DTLsetdelimiter

\DTLsetdelimiter{⟨*char*⟩}

This sets the delimiter.

```
\newcommand*\DTLsetdelimiter[1]{%
\renewcommand*{\@dtl@delimiter}{#1}%
\@dtl@construct@lopoffs}
```

\@dtl@construct@lopoff

\@dtl@construct@lopoff⟨*separator char*⟩⟨*delimiter char*⟩

This defines

\@dtl@lopoff⟨*first element*⟩⟨*sep*⟩⟨*rest of list*⟩\to⟨*cmd1*⟩⟨*cmd2*⟩

for the current separator and delimiter.

```
\edef\@dtl@construct@lopoff#1#2{%
\noexpand\long\noexpand\def\noexpand\@dtl@lopoff#1##1##2\noexpand
\to##3##4{%
\noexpand\ifx#2##1\noexpand\relax
\noexpand\@dtl@qlopoff#1##1##2\noexpand\to##3##4\relax
\noexpand\else
\noexpand\@dtl@lop@ff#1##1##2\noexpand\to##3##4\relax
\noexpand\fi
}}
```

\@dtl@construct@qlopoff

\@dtl@construct@qlopoff⟨*separator char*⟩⟨*delimiter char*⟩

This constructs \@dtl@qlopoff to be used when the entry is surrounded by the current delimiter value.

```
\edef\@dtl@construct@qlopoff#1#2{%
\noexpand\long\noexpand\def\noexpand\@dtl@qlopoff#1#2##1#2#1##2\noexpand
\to##3##4{%
```

```
\noexpand\def##4{##1}\noexpand\def##3{#1##2}%
}}
```

\@dtl@construct@lop@ff

\@dtl@construct@lop@ff⟨*separator char*⟩

This constructs \@dtl@lop@ff to be used when the entry isn't surrouded by the delimiter.

```
\edef\@dtl@construct@lop@ff#1{%
\noexpand\long\noexpand\def\noexpand\@dtl@lop@ff#1##1#1##2\noexpand
\to##3##4{%
\noexpand\def##4{##1}\noexpand\def##3{#1##2}%
}}
```

\@dtl@construct@lopoffs

\@dtl@construct@lopoffs

This constructs all the lopoff macros using the given separator and delimiter characters.

```
\newcommand{\@dtl@construct@lopoffs}{%
\edef\@dtl@chars{{\@dtl@separator}{\@dtl@delimiter}}%
\expandafter\@dtl@construct@lopoff\@dtl@chars
\expandafter\@dtl@construct@qlopoff\@dtl@chars
\expandafter\@dtl@construct@lop@ff\expandafter{\@dtl@separator}%
}
```

\@dtl@decimal

The current decimal character is stored in \@dtl@decimal.

```
\newcommand*{\@dtl@decimal}{.}
```

\@dtl@numbergroupchar

The current number group character is stored in \@dtl@numbergroupchar.

```
\newcommand*{\@dtl@numbergroupchar}{,}
```

\DTLsetnumberchars

\DTLsetnumberchars{⟨*number group char*⟩}{⟨*decimal char*⟩}

This sets the decimal character and number group characters.

```
\newcommand*{\DTLsetnumberchars}[2]{%
\renewcommand*{\@dtl@numbergroupchar}{#1}%
\renewcommand*{\@dtl@decimal}{#2}%
\@dtl@construct@getnums
\@dtl@construct@stripnumgrpchar{#1}}
```

\@dtl@construct@getintfrac

\@dtl@construct@getintfrac{⟨*char*⟩}

This constructs the macros for extracting integer and fractional parts from a real number using the decimal character ⟨*char*⟩.

\DTLconverttodecimal{⟨*num*⟩}{⟨*cmd*⟩}

\DTLconverttodecimal will convert locale dependent ⟨*num*⟩ a decimal number in a form that can be used in the macros defined in the fp package. The resulting number is stored in ⟨*cmd*⟩. This command has to be redefined whenever the decimal and number group characters are changed as they form part of the command definitions.

```
\edef\@dtl@construct@getintfrac#1{%
\noexpand\def\noexpand\@dtl@getintfrac##1#1##2\noexpand\relax{%
\noexpand\@dtl@get@intpart{##1}%
\noexpand\def\noexpand\@dtl@fracpart{##2}%
\noexpand\ifx\noexpand\@empty\noexpand\@dtl@fracpart
 \noexpand\def\noexpand\@dtl@fracpart{0}%
\noexpand\else
 \noexpand\@dtl@getfracpart##2\noexpand\relax
 \noexpand\@dtl@choptrailingzeroes{\noexpand\@dtl@fracpart}%
\noexpand\fi
}%
\noexpand\def\noexpand\@dtl@getfracpart##1#1\noexpand\relax{%
\noexpand\def\noexpand\@dtl@fracpart{##1}%
}%
\noexpand\def\noexpand\DTLconverttodecimal##1##2{%
\noexpand\dtl@ifsingle{##1}%
{\noexpand\expandafter\noexpand\toks@\noexpand\expandafter{##1}%
\noexpand\edef\noexpand\@dtl@tmp{\noexpand\the\noexpand\toks@}}%
{\noexpand\def\noexpand\@dtl@tmp{##1}}%
 \noexpand\@dtl@standardize@currency\noexpand\@dtl@tmp
 \noexpand\ifx\noexpand\@dtl@org@currency\noexpand\@empty
 \noexpand\else
   \noexpand\let\noexpand\@dtl@currency\noexpand\@dtl@org@currency
 \noexpand\fi
\noexpand\expandafter
\noexpand\@dtl@getintfrac\noexpand\@dtl@tmp#1\noexpand\relax
\noexpand\edef##2{\noexpand\@dtl@intpart.\noexpand\@dtl@fracpart}}%
}
```

\@dtl@construct@getnums The following calls the above with the relevant decimal character:

```
\newcommand*{\@dtl@construct@getnums}{%
\expandafter\@dtl@construct@getintfrac\expandafter{\@dtl@decimal}}
```

\@dtl@get@intpart The following gets the integer part (adjusting for repeating +/- signs if necessary.) Sets \@dtl@intpart.

```
\newcommand*{\@dtl@get@intpart}[1]{%
\@dtl@tmpcount=1\relax
\def\@dtl@intpart{#1}%
\ifx\@dtl@intpart\@empty
  \def\@dtl@intpart{0}%
\else
  \def\@dtl@intpart{}%
  \@dtl@get@int@part#1.\relax%
\fi
\ifnum\@dtl@tmpcount<0\relax
  \edef\@dtl@intpart{-\@dtl@intpart}%
\fi
\@dtl@strip@numgrpchar{\@dtl@intpart}%
}
```

\@dtl@get@int@part

```
\def\@dtl@get@int@part#1#2\relax{%
\def\@dtl@argi{#1}%
\def\@dtl@argii{#2}%
\ifx\protect#1\relax%
  \let\@dtl@get@nextintpart=\@dtl@get@int@part
\else
  \expandafter\ifx\@dtl@argi\$%
    \let\@dtl@get@nextintpart=\@dtl@get@int@part
  \else
    \ifx-#1%
      \multiply\@dtl@tmpcount by -1\relax
      \let\@dtl@get@nextintpart=\@dtl@get@int@part
    \else
      \if\@dtl@argi+%
        \let\@dtl@get@nextintpart=\@dtl@get@int@part
      \else
        \def\@dtl@intpart{#1}%
        \ifx.\@dtl@argii
          \let\@dtl@get@nextintpart=\@gobble
        \else
          \let\@dtl@get@nextintpart=\@dtl@get@next@intpart
        \fi
      \fi
    \fi
  \fi
\fi
\@dtl@get@nextintpart#2\relax
}
```

\@dtl@get@next@intpart

```
\def\@dtl@get@next@intpart#1.\relax{%
\edef\@dtl@intpart{\@dtl@intpart#1}%
}
```

\@dtl@choptrailingzeroes

\@dtl@choptrailingzeroes{⟨*cmd*⟩}

Chops trailing zeroes from number given by ⟨*cmd*⟩.

```
\newcommand*{\@dtl@choptrailingzeroes}[1]{%
\def\@dtl@tmpcpz{}%
\expandafter\@dtl@chop@trailingzeroes#1\@nil%
\let#1=\@dtl@tmpcpz
}
```

\@dtl@chop@trailingzeroes

Trailing zeroes are chopped using a recursive algorithm. \@dtl@tmpcpz needs to be set before using this. (The chopped number is put in this control sequence.)

```
\def\@dtl@chop@trailingzeroes#1#2\@nil{%
\FPifeq{#2}{0}%
  \edef\@dtl@tmpcpz{\@dtl@tmpcpz#1}%
  \let\@dtl@chopzeroesnext=\@dtl@gobbletonil
\else
 \edef\@dtl@tmpcpz{\@dtl@tmpcpz#1}%
```

```
 \let\@dtl@chopzeroesnext=\@dtl@chop@trailingzeroes
\fi
\@dtl@chopzeroesnext#2\@nil
}
```

No-op macro to end recursion:

\@dtl@gobbletonil

```
\def\@dtl@gobbletonil#1\@nil{}
```

\dtl@truncatedecimal

\dtl@truncatedecimal⟨*cmd*⟩

Truncates decimal given by ⟨*cmd*⟩ to an integer (assumes the number is in decimal format with full stop as decimal point.)

```
\newcommand*{\dtl@truncatedecimal}[1]{%
\expandafter\@dtl@truncatedecimal#1.\@nil#1}
```

\@dtl@truncatedecimal

```
\def\@dtl@truncatedecimal#1.#2\@nil#3{%
\def#3{#1}}
```

\@dtl@strip@numgrpchar

\@dtl@strip@numgrpchar{⟨*cmd*⟩}

Strip the number group character from the number given by ⟨*cmd*⟩.

```
\newcommand*{\@dtl@strip@numgrpchar}[1]{%
\def\@dtl@stripped{}%
\edef\@dtl@do@stripnumgrpchar{%
\noexpand\@@dtl@strip@numgrpchar#1\@dtl@numbergroupchar
\noexpand\relax}%
\@dtl@do@stripnumgrpchar
\let#1=\@dtl@stripped
}
```

dtl@construct@stripnumgrpchar

The following macro constructs \@@dtl@strip@numgrpchar.

```
\edef\@dtl@construct@stripnumgrpchar#1{%
\noexpand\def\noexpand\@@dtl@strip@numgrpchar##1#1##2\noexpand\relax{%
\noexpand\expandafter\noexpand\toks@\noexpand\expandafter
{\noexpand\@dtl@stripped}%
\noexpand\edef\noexpand\@dtl@stripped{\noexpand\the\noexpand\toks@
##1}%
\noexpand\def\noexpand\@dtl@tmp{##2}%
\noexpand\ifx\noexpand\@dtl@tmp\noexpand\@empty
 \noexpand\let\noexpand\@dtl@next=\noexpand\relax
\noexpand\else
 \noexpand\let\noexpand\@dtl@next=\noexpand\@@dtl@strip@numgrpchar
\noexpand\fi
\noexpand\@dtl@next##2\noexpand\relax
}%
}
```

\DTLdecimaltolocale

\DTLdecimaltolocale{⟨*number*⟩}{⟨*cmd*⟩}

Define command to convert a decimal number into the locale dependent format. Stores result in ⟨*cmd*⟩ which must be a control sequence.

```
\newcommand*{\DTLdecimaltolocale}[2]{%
\edef\@dtl@tmpdtl{#1}%
\expandafter\@dtl@decimaltolocale\@dtl@tmpdtl.\relax
\FPifeq{\@dtl@fracpart}{0}%
 \edef#2{\@dtl@intpart}%
\else
 \edef#2{\@dtl@intpart\@dtl@decimal\@dtl@fracpart}%
\fi
}
```

\@dtl@decimaltolocale

Convert the integer part (store in \@dtl@intpart)

```
\def\@dtl@decimaltolocale#1.#2\relax{%
\@dtl@decimaltolocaleint{#1}%
\def\@dtl@fracpart{#2}%
\ifx\@dtl@fracpart\@empty
 \def\@dtl@fracpart{0}%
\else
 \@dtl@decimaltolocalefrac#2\relax
\fi
}
```

\@dtl@decimaltolocaleint

```
\def\@dtl@decimaltolocaleint#1{%
\@dtl@tmpcount=0\relax
\@dtl@countdigits#1.\relax
\@dtl@numgrpsepcount=\@dtl@tmpcount\relax
\divide\@dtl@numgrpsepcount by 3\relax
\multiply\@dtl@numgrpsepcount by 3\relax
\advance\@dtl@numgrpsepcount by -\@dtl@tmpcount\relax
\ifnum\@dtl@numgrpsepcount<0\relax
  \advance\@dtl@numgrpsepcount by 3\relax
\fi
\def\@dtl@intpart{}%
\@dtl@decimal@to@localeint#1.\relax
}
```

\@dtl@countdigits

Counts the number of digits until #2 is a full stop. (increments \@dtl@tmpcount.)

```
\def\@dtl@countdigits#1#2\relax{%
\advance\@dtl@tmpcount by 1\relax
\ifx.#2\relax
 \let\@dtl@countnext=\@gobble
\else
 \let\@dtl@countnext=\@dtl@countdigits
\fi
\@dtl@countnext#2\relax
}
```

\@dtl@decimal@to@localeint

```
\def\@dtl@decimal@to@localeint#1#2\relax{%
\advance\@dtl@numgrpsepcount by 1\relax
\ifx.#2\relax
 \edef\@dtl@intpart{\@dtl@intpart#1}%
 \let\@dtl@localeintnext=\@gobble
\else
 \ifnum\@dtl@numgrpsepcount=3\relax
   \edef\@dtl@intpart{\@dtl@intpart#1\@dtl@numbergroupchar}%
   \@dtl@numgrpsepcount=0\relax
 \else
   \ifnum\@dtl@numgrpsepcount>3\relax
     \@dtl@numgrpsepcount=0\relax
   \fi
   \edef\@dtl@intpart{\@dtl@intpart#1}%
 \fi
 \let\@dtl@localeintnext=\@dtl@decimal@to@localeint
\fi
\@dtl@localeintnext#2\relax
}
```

`\@dtl@decimaltolocalefrac` Convert the fractional part (store in `\@dtl@fracpart`)

```
%   \end{macrocode}
\def\@dtl@decimaltolocalefrac#1.\relax{%
\def\@dtl@fracpart{#1}%
\@dtl@choptrailingzeroes{\@dtl@fracpart}%
}
%\end{macro}
%
%\begin{macro}{\DTLdecimaltocurrency}
%\begin{definition}
% \cs{DTLdecimaltocurrency}\marg{number}\marg{cmd}
%\end{definition}
% This converts a decimal number into the locale
% dependent currency format. Stores result in \meta{cmd} which must be
% a control sequence.
%    \begin{macrocode}
\newcommand*{\DTLdecimaltocurrency}[2]{%
\edef\@dtl@tmpdtl{#1}%
\expandafter\@dtl@decimaltolocale\@dtl@tmpdtl.\relax
\dtl@truncatedecimal\@dtl@tmpdtl
\@dtl@tmpcount=\@dtl@tmpdtl\relax
\expandafter\@dtl@toks\expandafter{\@dtl@currency}%
\FPifeq{\@dtl@fracpart}{0}%
 \ifnum\@dtl@tmpcount<0\relax
   \@dtl@tmpcount = -\@dtl@tmpcount\relax
   \edef#2{-\the\@dtl@toks\the\@dtl@tmpcount\@dtl@decimal00}%
 \else
   \edef#2{\the\@dtl@toks\@dtl@intpart\@dtl@decimal00}%
 \fi
\else
 \ifnum\@dtl@tmpcount<0\relax
   \@dtl@tmpcount = -\@dtl@tmpcount\relax
   \ifnum\@dtl@fracpart<10\relax
     \edef#2{-\the\@dtl@toks\number\@dtl@tmpcount
```

```
        \@dtl@decimal\@dtl@fracpart0}%
    \else
      \edef#2{-\the\@dtl@toks\number\@dtl@tmpcount
        \@dtl@decimal\@dtl@fracpart}%
    \fi
  \else
    \ifnum\@dtl@fracpart<10\relax
      \edef#2{\the\@dtl@toks\@dtl@intpart\@dtl@decimal\@dtl@fracpart0}%
    \else
      \edef#2{\the\@dtl@toks\@dtl@intpart\@dtl@decimal\@dtl@fracpart}%
    \fi
  \fi
\fi
}
```

Set the defaults:

```
\@dtl@construct@lopoffs
\@dtl@construct@getnums
\expandafter\@dtl@construct@stripnumgrpchar\expandafter
{\@dtl@numbergroupchar}
```

Define key for package option separator.

```
\define@key{datatool.sty}{separator}{%
\DTLsetseparator{#1}}
```

Define key for package option delimiter.

```
\define@key{datatool.sty}{delimiter}{%
\DTLsetdelimiter{#1}}
```

Define key for package option verbose. (This also switches the fp messages on/off)

```
\define@boolkey{datatool.sty}[dtl]{verbose}[true]{%
\ifdtlverbose \FPmessagestrue\else \FPmessagesfalse\fi}
```

`\dtl@message` `\dtl@message{`⟨*message string*⟩`}`

Displays message only if the verbose option is set.

```
\newcommand*{\dtl@message}[1]{%
\ifdtlverbose\typeout{#1}\fi}
```

Process package options:

```
\ProcessOptionsX
```

`\DTLpar` Many of the commands used by this package are short commands. This means that you can't use `\par` in the data. To get around this, define the robust command `\DTLpar` to use instead.

```
\DeclareRobustCommand\DTLpar{\@par}
```

10.3 Determining Data Types

The control sequence `\@dtl@checknumerical` checks the data type of its argument, and sets `\@dtl@datatype` to 0 if the argument is a string, 1 if the argument is an integer or 2 if the argument is a real number. First define `\@dtl@datatype`:

`\@dtl@datatype`

```
\newcount\@dtl@datatype
```

`\@dtl@tmpcount` Define temporary count register

```
\newcount\@dtl@tmpcount
```

`\dtl@tmplength` Define temporary length register:

```
\newlength\dtl@tmplength
```

`\@dtl@numgrpsepcount` Define count register to count the digits between the number group separators.

```
\newcount\@dtl@numgrpsepcount
```

`\@dtl@checknumerical`

`\@dtl@checknumerical{`⟨*arg*⟩`}`

Checks if ⟨*arg*⟩ is numerical (includes decimal numbers, but not scientific notation.) Sets `\@dtl@datatype`, as described above.

```
\newcommand{\@dtl@checknumerical}[1]{%
\@dtl@numgrpsepfalse
\def\@dtl@tmp{#1}%
\ifx\@empty#1\@empty
 \@dtl@datatype=0\relax
\else
 \dtl@ifsingle{#1}%
 {\expandafter\toks@\expandafter{#1}%
 \edef\@dtl@tmp{\the\toks@}}%
 {\def\@dtl@tmp{#1}}%
 \@dtl@tmpcount=0\relax
 \@dtl@datatype=0\relax
 \@dtl@numgrpsepcount=2\relax
 \@dtl@standardize@currency\@dtl@tmp
 \ifx\@dtl@org@currency\@empty
 \else
   \let\@dtl@currency\@dtl@org@currency
 \fi
 \expandafter\@dtl@checknumericalstart\@dtl@tmp\@nil\@nil
\fi
\ifnum\@dtl@numgrpsepcount>-1\relax
  \if@dtl@numgrpsep
    \ifnum\@dtl@numgrpsepcount=3\relax
    \else
      \@dtl@datatype=0\relax
    \fi
  \fi
\fi
}
```

`\@dtl@checknumericalstart` Check first character for checknumerical process to see if it's a plus or minus sign.

```
\def\@dtl@checknumericalstart#1#2\@nil\@nil{%
\ifx#1\protect
  \@dtl@checknumericalstart#2\@nil\@nil\relax
\else
```

```
    \ifx-#1\relax
      \def\@dtl@tmp{#2}%
      \ifx\@empty\@dtl@tmp
        \@dtl@datatype=0\relax
      \else
        \ifnum\@dtl@datatype=0\relax
          \@dtl@datatype=1\relax
        \fi
        \@dtl@checknumericalstart#2\@nil\@nil\relax
      \fi
    \else
      \ifx+#1\relax
        \def\@dtl@tmp{#2}%
        \ifx\@empty\@dtl@tmp
          \@dtl@datatype=0\relax
        \else
          \ifnum\@dtl@datatype=0\relax
            \@dtl@datatype=1\relax
          \fi
          \@dtl@checknumericalstart#2\@nil\@nil\relax
        \fi
      \else
      \def\@dtl@tmp{#1}%
        \ifx#1\$\relax
          \@dtl@datatype=3\relax
          \@dtl@checknumericalstart#2\@nil\@nil\relax
        \else
          \ifx\@empty\@dtl@tmp
            \@dtl@datatype=0\relax
          \else
            \ifnum\@dtl@datatype=0\relax
              \@dtl@datatype=1\relax
            \fi
            \@dtl@checknumericalloop#1#2\@nil\@nil\relax
          \fi
        \fi
      \fi
    \fi
  \fi
  }
```

`\if@dtl@numgrpsep` The conditional `\if@dtl@numgrpsep` is set the first time `\@dtl@checknumericalloop` encounters the number group separator.

```
\newif\if@dtl@numgrpsep
```

`\@dtl@ifDigitOrDecimalSep` Check if argument is either a digit or the decimal separator.

```
\newcommand*{\@dtl@ifDigitOrDecimalSep}[3]{%
\ifx0#1\relax
 #2%
\else
 \ifx1#1\relax
  #2%
 \else
   \ifx2#1\relax
```

```
      #2%
    \else
      \ifx3#1\relax
        #2%
      \else
        \ifx4#1\relax
          #2%
        \else
          \ifx5#1\relax
           #2%
          \else
            \ifx6#1\relax
              #2%
            \else
              \ifx7#1\relax
                #2%
              \else
                \ifx8#1\relax
                  #2%
                \else
                  \ifx9#1\relax
                    #2%
                  \else
                    \expandafter\ifx\@dtl@decimal#1\relax
                      #2%
                    \else
                      #3%
                    \fi
                  \fi
                \fi
              \fi
            \fi
          \fi
        \fi
      \fi
    \fi
  \fi
\fi
}
```

`\@dtl@checknumericalloop` Check numerical loop. This iterates through each character until `\@nil` is reached, or invalid character found. Increments `\@dtl@tmpcount` each time it encounters a decimal character.

```
\def\@dtl@checknumericalloop#1#2\@nil{%
\def\@dtl@tmp{#1}%
\ifx\@nnil\@dtl@tmp\relax
 \let\@dtl@chcknumnext=\@dtl@checknumericalnoop%
\else
  \@dtl@ifDigitOrDecimalSep{#1}{%
  \let\@dtl@chcknumnext=\@dtl@checknumericalloop%
  \expandafter\ifx\@dtl@decimal#1\relax
     \if@dtl@numgrpsep
       \ifnum\@dtl@numgrpsepcount=3\relax
        \@dtl@numgrpsepcount=-1\relax
```

```
        \else
         \@dtl@datatype=0\relax
         \let\@dtl@chcknumnext=\@dtl@checknumericalnoop
        \fi
      \else
        \@dtl@numgrpsepcount=-1\relax
      \fi
    \else
      \ifnum\@dtl@numgrpsepcount=-1\relax
      \else
        \advance\@dtl@numgrpsepcount by 1\relax
      \fi
    \fi
}{%
\ifx\@dtl@numbergroupchar\@dtl@tmp\relax
  \@dtl@numgrpseptrue
  \ifnum\@dtl@numgrpsepcount<3\relax
    \@dtl@datatype=0\relax
    \let\@dtl@chcknumnext=\@dtl@checknumericalnoop
  \else
     \@dtl@numgrpsepcount=0\relax
  \fi
\else
  \@dtl@datatype=0\relax
  \let\@dtl@chcknumnext=\@dtl@checknumericalnoop
\fi
}%
  \ifx\@dtl@decimal\@dtl@tmp\relax
   \ifnum\@dtl@datatype<3\relax
     \@dtl@datatype=2\relax
   \fi
   \advance\@dtl@tmpcount by 1\relax
   \ifnum\@dtl@tmpcount>1\relax
     \@dtl@datatype=0\relax
     \let\@dtl@chcknumnext=\@dtl@checknumericalnoop%
   \fi
  \fi
\fi
\@dtl@chcknumnext#2\@nil
}
```

`\@dtl@checknumericalnoop` End loop

```
\def\@dtl@checknumericalnoop#1\@nil#2{}
```

`\DTLifnumerical`

`\DTLifnumerical{`⟨*arg*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

Tests the first argument, if its numerical do second argument, otherwise do third argument.

```
\newcommand{\DTLifnumerical}[3]{%
\@dtl@checknumerical{#1}%
\ifnum\@dtl@datatype=0\relax#3\else#2\fi
}
```

\DTLifreal

\DTLifreal{⟨*arg*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Tests the first argument, if it's a real number (not an integer) do second argument, otherwise do third argument.

```
\newcommand{\DTLifreal}[3]{%
\@dtl@checknumerical{#1}%
\ifnum\@dtl@datatype=2\relax #2\else #3\fi
}
```

\DTLifint

\DTLifint{⟨*arg*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Tests the first argument, if it's an integer do second argument, otherwise do third argument.

```
\newcommand{\DTLifint}[3]{%
\@dtl@checknumerical{#1}%
\ifnum\@dtl@datatype=1\relax #2\else #3\fi
}
```

\DTLifstring

\DTLifstring{⟨*arg*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Tests the first argument, if it's a string do second argument, otherwise do third argument.

```
\newcommand{\DTLifstring}[3]{%
\@dtl@checknumerical{#1}%
\ifnum\@dtl@datatype=0\relax #2\else #3\fi
}
```

\DTLifcurrency

\DTLifcurrency{⟨*arg*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Tests the first argument, if it starts with the currency symbol do second argument, otherwise do third argument.

```
\newcommand{\DTLifcurrency}[3]{%
\@dtl@checknumerical{#1}%
\ifnum\@dtl@datatype=3\relax #2\else #3\fi
}
```

\DTLifcurrencyunit

\DTLifcurrencyunit{⟨*arg*⟩}{⟨*symbol*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

This tests if ⟨*arg*⟩ is currency, and uses the currency unit ⟨*symbol*⟩. If true do third argument, otherwise do fourth argument.

```
\newcommand*{\DTLifcurrencyunit}[4]{%
\@dtl@checknumerical{#1}%
\ifnum\@dtl@datatype=3\relax
  \ifthenelse{\equal{\@dtl@org@currency}{#2}}{#3}{#4}%
\else
```

```
  #4%
\fi
}
```

\DTLifcasedatatype

\DTLifcasedatatype{⟨*arg*⟩}{⟨*string case*⟩}{⟨*int case*⟩}{⟨*real case*⟩}{⟨*currency case*⟩}

If ⟨*arg*⟩ is a string, do ⟨*string case*⟩, if ⟨*arg*⟩ is an integer do ⟨*int case*⟩, if ⟨*arg*⟩ is a real number, do ⟨*real case*⟩, if ⟨*arg*⟩ is currency, do ⟨*curreny case*⟩.

```
\newcommand{\DTLifcasedatatype}[5]{%
\@dtl@checknumerical{#1}%
\ifcase\@dtl@datatype
 #2% string
\or
 #3% integer
\or
 #4% number
\or
 #5% currency
\fi
}
```

\dtl@testbothnumerical

\dtl@testbothnumerical{⟨*arg1*⟩}{⟨*arg2*⟩}

Tests if both arguments are numerical. This sets the conditional `\if@dtl@condition`.

```
\newcommand*{\dtl@testbothnumerical}[2]{%
\dtl@ifsingle{#1}{%
 \edef\@dtl@tmp{#1}}{%
 \def\@dtl@tmp{#1}}%
\expandafter\@dtl@checknumerical\expandafter{\@dtl@tmp}%
\edef\@dtl@firsttype{\number\@dtl@datatype}%
\dtl@ifsingle{#2}{%
 \edef\@dtl@tmp{#2}}{%
 \def\@dtl@tmp{#2}}%
\expandafter\@dtl@checknumerical\expandafter{\@dtl@tmp}%
\multiply\@dtl@datatype by \@dtl@firsttype\relax
\ifnum\@dtl@datatype>0\relax
 \@dtl@conditiontrue
\else
 \@dtl@conditionfalse
\fi
}
```

\DTLifnumlt

\DTLifnumlt{⟨*num1*⟩}{⟨*num2*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Determines if {⟨*num1*⟩} < {⟨*num2*⟩}. Both numbers need to have the decimal separator changed to a dot to ensure that it works with `\FPiflt`

```
\newcommand*{\DTLifnumlt}[4]{%
  \DTLconverttodecimal{#1}{\@dtl@numi}%
```

```
    \DTLconverttodecimal{#2}{\@dtl@numii}%
    \FPiflt{\@dtl@numi}{\@dtl@numii}%
     #3%
    \else
     #4%
    \fi
  }
```

\dtlcompare

\dtlcompare{⟨*count*⟩}{⟨*string1*⟩}{⟨*string2*⟩}

Compares ⟨*string1*⟩ and ⟨*string2*⟩, and stores the result in the count register ⟨*count*⟩. The result may be one of:

-1	if ⟨*string1*⟩ is considered to be less than ⟨*string2*⟩
0	if ⟨*string1*⟩ is considered to be the same as ⟨*string2*⟩
1	if ⟨*string1*⟩ is considered to be greater than ⟨*string2*⟩

Note that for the purposes of string comparisons, commands within ⟨*string1*⟩ and ⟨*string2*⟩ are ignored, except for `\space` and `~`, which are both treated as a space (character code 32.) The following examples assume that the count register `\mycount` has been defined as follows:

```
\newcount\mycount
```

Examples:

1. `\dtlcompare{\mycount}{Z\"oe}{Zoe}\number\mycount`

   produces: 0, since the accent command is ignored.

2. `\dtlcompare{\mycount}{foo}{Foo}\number\mycount`

   produces: 1, since the comparison is case sensitive, however, note the following example:

3. `\dtlcompare{\mycount}{foo}{\uppercase{f}oo}\number\mycount`

   which produces: 0, since the `\uppercase` command is ignored.

4. You can "trick" `\dtlcompare` using a command which doesn't output any text. Suppose you have defined the following command:

   ```
   \newcommand*{\noopsort}[1]{}
   ```

   then `\noopsort{a}foo` produces the text: foo, however the following

   ```
   \dtlcompare{\mycount}{\noopsort{a}foo}{bar}\number\mycount
   ```

   produces: -1, since the command `\noopsort` is disregarded when the comparison is made, so `\dtlcompare` just compares `{a}foo` with `bar`, and since `a` is less than `b`, the first string is considered to be less than the second string.

5. Note that this also means that:

   ```
   \def\mystr{abc}%
   \dtlcompare{\mycount}{\mystr}{abc}\number\mycount
   ```

   produces: -1, since the command `\mystr` is disregarded, which means that `\dtlcompare` is comparing an empty string with the string `abc`.

6. Spaces count in the comparison:

   ```
   \dtlcompare{\mycount}{ab cd}{abcd}\number\mycount
   ```

   produces: -1, but sequential spaces are treated as a single space:

   ```
   \dtlcompare{\mycount}{ab  cd}{ab cd}\number\mycount
   ```

   produces: 0.

7. As usual, spaces following command names are ignored, so

   ```
   \dtlcompare{\mycount}{ab\relax cd}{ab cd}\number\mycount
   ```

   produces: 1.

8. `~` and `\space` are considered to be the same as a space:

   ```
   \dtlcompare{\mycount}{ab cd}{ab~cd}\number\mycount
   ```

   produces: 0.

```
\newcommand*{\dtlcompare}[3]{%
\dtl@subnobrsp{#2}{\@dtl@argA}%
\dtl@subnobrsp{#3}{\@dtl@argB}%
\ifx\@dtl@argA\@empty
 \ifx\@dtl@argB\@empty
   #1=0\relax
 \else
   #1=-1\relax
 \fi
\else
  \ifx\@dtl@argB\@empty
    #1=1\relax
  \else
    \DTLsubstituteall{\@dtl@argA}{ }{\space }%
    \DTLsubstituteall{\@dtl@argB}{ }{\space }%
    \expandafter\dtl@getfirst\@dtl@argA\end
    \let\dtl@firstA=\dtl@first
    \let\dtl@restA=\dtl@rest
    \expandafter\dtl@getfirst\@dtl@argB\end
    \let\dtl@firstB=\dtl@first
    \let\dtl@restB=\dtl@rest
    \expandafter\dtl@ifsingle\expandafter{\dtl@firstA}{%
    \expandafter\dtl@ifsingle\expandafter{\dtl@firstB}{%
    \expandafter\dtl@setcharcode\expandafter{\dtl@firstA}{\dtl@codeA}%
```

```
\expandafter\dtl@setcharcode\expandafter{\dtl@firstB}{\dtl@codeB}%
\ifnum\dtl@codeA=-1\relax
  \ifnum\dtl@codeB=-1\relax
     \protected@edef\dtl@donext{%
       \noexpand\dtlcompare{\noexpand#1}{\dtl@restA}{\dtl@restB}}%
     \dtl@donext
  \else
     \protected@edef\dtl@donext{%
       \noexpand\dtlcompare
         {\noexpand#1}{\dtl@restA}{\dtl@firstB\dtl@restB}}%
     \dtl@donext
  \fi
\else
  \ifnum\dtl@codeB=-1\relax
     \protected@edef\dtl@donext{%
       \noexpand\dtlcompare
         {\noexpand#1}{\dtl@firstA\dtl@restA}{\dtl@restB}}%
     \dtl@donext
  \else
    \ifnum\dtl@codeA<\dtl@codeB
      #1=-1\relax
    \else
      \ifnum\dtl@codeA>\dtl@codeB
        #1=1\relax
      \else
         \ifx\dtl@restA\@empty
           \ifx\dtl@restB\@empty
             #1=0\relax
           \else
             #1=-1\relax
           \fi
         \else
           \ifx\restB\@empty
             #1=1\relax
           \else
             \protected@edef\dtl@donext{%
               \noexpand\dtlcompare
                 {\noexpand#1}{\dtl@restA}{\dtl@restB}}%
             \dtl@donext
           \fi
         \fi
      \fi
    \fi
  \fi
\fi
}{%
\protected@edef\dtl@donext{%
  \noexpand\dtlcompare
    {\noexpand#1}{\dtl@firstA\dtl@restA}{\dtl@firstB\dtl@restB}}%
\dtl@donext
}}{%
\protected@edef\dtl@donext{%
  \noexpand\dtlcompare
    {\noexpand#1}{\dtl@firstA\dtl@restA}{\dtl@firstB\dtl@restB}}%
```

```
      \dtl@donext
      }%
    \fi
  \fi
  }
```

\dtl@getfirst Gets the first object, and stores in \dtl@first. The remainder is stored in \dtl@rest.

```
\def\dtl@getfirst#1#2\end{%
\def\dtl@first{#1}%
\ifx\dtl@first\@empty
 \def\dtl@rest{#2}%
\else
 \dtl@ifsingle{#1}{\def\dtl@rest{#2}}{\dtl@getfirst#1#2\end}%
\fi
}
```

Count registers to store character codes:

```
\newcount\dtl@codeA
\newcount\dtl@codeB
```

\dtl@setcharcode

\dtl@setcharcode{⟨*c*⟩}{⟨*count register*⟩}

Sets ⟨*count register*⟩ to the character code of ⟨*c*⟩, or to -1 if ⟨*c*⟩ is a control sequence, unless ⟨*c*⟩ is either \space or | | in which case it sets ⟨*count register*⟩ to the character code of the space character.

```
\newcommand*{\dtl@setcharcode}[2]{%
\def\@dtl@tmp{#1}%
\ifx\@dtl@tmp\@empty
 #2=-1\relax
\else
 \ifx#1\space\relax
   #2=`\ \relax
 \else
   \ifx#1~\relax
     #2=`\ \relax
   \else
     \ifcat\noexpand#1\relax%
       #2=-1\relax
     \else
       #2=`#1\relax
     \fi
   \fi
 \fi
\fi
}
```

\dtl@setlccharcode

\dtl@setlccharcode{⟨*c*⟩}{⟨*count register*⟩}

As \dtl@setlccharcode except it sets ⟨*count register*⟩ to the lower case character code of ⟨*c*⟩, unless ⟨*c*⟩ is a control sequence, in which case it does the same as \dtl@setcharcode.

```
\newcommand*{\dtl@setlccharcode}[2]{%
\def\@dtl@tmp{#1}%
\ifx\@dtl@tmp\@empty
 #2=-1\relax
\else
 \ifx#1\space\relax
   #2=`\ \relax
 \else
   \ifx#1~\relax
     #2=`\ \relax
   \else
     \ifcat\noexpand#1\relax%
       #2=-1\relax
     \else
       #2=\lccode`#1\relax
     \fi
   \fi
 \fi
\fi
}
```

\dtlicompare

\dtlicompare{⟨*count*⟩}{⟨*string1*⟩}{⟨*string2*⟩}

As \dtlcompare but ignores case.

```
\newcommand*{\dtlicompare}[3]{%
\dtl@subnobrsp{#2}{\@dtl@argA}%
\dtl@subnobrsp{#3}{\@dtl@argB}%
\ifx\@dtl@argA\@empty
 \ifx\@dtl@argB\@empty
   #1=0\relax
 \else
   #1=-1\relax
 \fi
\else
  \ifx\@dtl@argB\@empty
    #1=1\relax
  \else
    \DTLsubstituteall{\@dtl@argA}{ }{\space }%
    \DTLsubstituteall{\@dtl@argB}{ }{\space }%
    \expandafter\dtl@getfirst\@dtl@argA\end
    \let\dtl@firstA=\dtl@first
    \let\dtl@restA=\dtl@rest
    \expandafter\dtl@getfirst\@dtl@argB\end
    \let\dtl@firstB=\dtl@first
    \let\dtl@restB=\dtl@rest
    \expandafter\dtl@ifsingle\expandafter{\dtl@firstA}{%
    \expandafter\dtl@ifsingle\expandafter{\dtl@firstB}{%
    \expandafter\dtl@setlccharcode\expandafter{\dtl@firstA}{\dtl@codeA}%
    \expandafter\dtl@setlccharcode\expandafter{\dtl@firstB}{\dtl@codeB}%
```

```
\ifnum\dtl@codeA=-1\relax
  \ifnum\dtl@codeB=-1\relax
     \protected@edef\dtl@donext{%
       \noexpand\dtlicompare{\noexpand#1}{\dtl@restA}{\dtl@restB}}%
     \dtl@donext
  \else
     \protected@edef\dtl@donext{%
       \noexpand\dtlicompare
         {\noexpand#1}{\dtl@restA}{\dtl@firstB\dtl@restB}}%
     \dtl@donext
  \fi
\else
  \ifnum\dtl@codeB=-1\relax
     \protected@edef\dtl@donext{%
       \noexpand\dtlicompare
         {\noexpand#1}{\dtl@firstA\dtl@restA}{\dtl@restB}}%
     \dtl@donext
  \else
    \ifnum\dtl@codeA<\dtl@codeB
      #1=-1\relax
    \else
      \ifnum\dtl@codeA>\dtl@codeB
        #1=1\relax
      \else
         \ifx\dtl@restA\@empty
           \ifx\dtl@restB\@empty
             #1=0\relax
           \else
             #1=-1\relax
           \fi
         \else
           \ifx\restB\@empty
             #1=1\relax
           \else
             \protected@edef\dtl@donext{%
               \noexpand\dtlicompare
                 {\noexpand#1}{\dtl@restA}{\dtl@restB}}%
             \dtl@donext
           \fi
         \fi
      \fi
    \fi
  \fi
\fi
}{%
\protected@edef\dtl@donext{%
  \noexpand\dtlicompare
    {\noexpand#1}{\dtl@firstA\dtl@restA}{\dtl@firstB\dtl@restB}}%
\dtl@donext
}}{%
\protected@edef\dtl@donext{%
  \noexpand\dtlicompare
    {\noexpand#1}{\dtl@firstA\dtl@restA}{\dtl@firstB\dtl@restB}}%
\dtl@donext
```

```
    }%
  \fi
\fi
}
```

\DTLifstringlt

\DTLifstringlt{⟨*string1*⟩}{⟨*string2*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

String comparison (Starred version ignores case)

```
\newcommand*{\DTLifstringlt}{\@ifstar\@sDTLifstringlt\@DTLifstringlt}
```

Unstarred version

```
\newcommand*{\@DTLifstringlt}[4]{%
\protected@edef\@dtl@tmpcmp{%
  \noexpand\dtlcompare{\noexpand\@dtl@tmpcount}{#1}{#2}}%
\@dtl@tmpcmp
\ifnum\@dtl@tmpcount<0\relax
  #3%
\else
  #4%
\fi
}
```

Starred version

```
\newcommand*{\@sDTLifstringlt}[4]{%
\protected@edef\@dtl@tmpcmp{%
  \noexpand\dtlicompare{\noexpand\@dtl@tmpcount}{#1}{#2}}%
\@dtl@tmpcmp
\ifnum\@dtl@tmpcount<0\relax
  #3%
\else
  #4%
\fi
}
```

\DTLiflt

\DTLiflt{⟨*arg1*⟩}{⟨*arg2*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Does \DTLifnumlt if both ⟨*arg1*⟩ and ⟨*arg2*⟩ are numerical, otherwise do \DTLifstringlt (unstarred version) or \DTLifstringlt* (starred version).

```
\newcommand*{\DTLiflt}{\@ifstar\@sDTLiflt\@DTLiflt}
```

Unstarred version

```
\newcommand*{\@DTLiflt}[4]{%
\dtl@testbothnumerical{#1}{#2}%
\if@dtl@condition
 \DTLifnumlt{#1}{#2}{#3}{#4}%
\else
 \@DTLifstringlt{#1}{#2}{#3}{#4}%
\fi
}
```

Starred version

```
\newcommand*{\@sDTLiflt}[4]{%
```

```
\dtl@testbothnumerical{#1}{#2}%
\if@dtl@condition
 \DTLifnumlt{#1}{#2}{#3}{#4}%
\else
 \@sDTLifstringlt{#1}{#2}{#3}{#4}%
\fi
}
```

\DTLifnumgt

\DTLifnumgt{⟨*num1*⟩}{⟨*num2*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Determines if {⟨*num1*⟩} > {⟨*num2*⟩}. Both numbers need to have the decimal separator changed to a dot to ensure that it works with \FPifgt

```
\newcommand*{\DTLifnumgt}[4]{%
  \DTLconverttodecimal{#1}{\@dtl@numi}%
  \DTLconverttodecimal{#2}{\@dtl@numii}%
  \FPifgt{\@dtl@numi}{\@dtl@numii}%
   #3%
  \else
   #4%
  \fi
}
```

\DTLifstringgt

\DTLifstringgt{⟨*string1*⟩}{⟨*string2*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

String comparison (starred version ignores case)

```
\newcommand*{\DTLifstringgt}{\@ifstar\@sDTLifstringgt\@DTLifstringgt}
```

Unstarred version

```
\newcommand*{\@DTLifstringgt}[4]{%
\protected@edef\@dtl@tmpcmp{%
  \noexpand\dtlcompare{\noexpand\@dtl@tmpcount}{#1}{#2}}%
\@dtl@tmpcmp
\ifnum\@dtl@tmpcount>0\relax
  #3%
\else
  #4%
\fi
}
```

Starred version

```
\newcommand*{\@sDTLifstringgt}[4]{%
\protected@edef\@dtl@tmpcmp{%
  \noexpand\dtlicompare{\noexpand\@dtl@tmpcount}{#1}{#2}}%
\@dtl@tmpcmp
\ifnum\@dtl@tmpcount>0\relax
  #3%
\else
  #4%
\fi
}
```

\DTLifgt

\DTLifgt{⟨*arg1*⟩}{⟨*arg2*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Does \DTLifnumgt if both ⟨*arg1*⟩ and ⟨*arg2*⟩ are numerical, otherwise do \DTLifstringgt or \DTLifstringgt*.

```
\newcommand*{\DTLifgt}{\@ifstar\@sDTLifgt\@DTLifgt}
```

Unstarred version

```
\newcommand*{\@DTLifgt}[4]{%
\dtl@testbothnumerical{#1}{#2}%
\if@dtl@condition
 \DTLifnumgt{#1}{#2}{#3}{#4}%
\else
 \@DTLifstringgt{#1}{#2}{#3}{#4}%
\fi
}
```

Starred version

```
\newcommand*{\@sDTLifgt}[4]{%
\dtl@testbothnumerical{#1}{#2}%
\if@dtl@condition
 \DTLifnumgt{#1}{#2}{#3}{#4}%
\else
 \@sDTLifstringgt{#1}{#2}{#3}{#4}%
\fi
}
```

\DTLifnumeq

\DTLifnumeq{⟨*num1*⟩}{⟨*num2*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Determines if {⟨*num1*⟩} = {⟨*num2*⟩}. Both numbers need to have the decimal separator changed to a dot to ensure that it works with \FPifeq

```
\newcommand*{\DTLifnumeq}[4]{%
  \DTLconverttodecimal{#1}{\@dtl@numi}%
  \DTLconverttodecimal{#2}{\@dtl@numii}%
  \FPifeq{\@dtl@numi}{\@dtl@numii}%
   #3%
  \else
   #4%
  \fi
}
```

\DTLifstringeq

\DTLifstringeq{⟨*string1*⟩}{⟨*string2*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

String comparison (starred version ignores case)

```
\newcommand*{\DTLifstringeq}{\@ifstar\@sDTLifstringeq\@DTLifstringeq}
```

Unstarred version

```
\newcommand*{\@DTLifstringeq}[4]{%
\protected@edef\@dtl@tmpcmp{%
  \noexpand\dtlcompare{\noexpand\@dtl@tmpcount}{#1}{#2}}%
\@dtl@tmpcmp
```

```
\ifnum\@dtl@tmpcount=0\relax
  #3%
\else
  #4%
\fi
}
```

Starred version

```
\newcommand*{\@sDTLifstringeq}[4]{%
\protected@edef\@dtl@tmpcmp{%
  \noexpand\dtlicompare{\noexpand\@dtl@tmpcount}{#1}{#2}}%
\@dtl@tmpcmp
\ifnum\@dtl@tmpcount=0\relax
  #3%
\else
  #4%
\fi
}
```

`\DTLifeq` `\DTLifeq{⟨arg1⟩}{⟨arg2⟩}{⟨true part⟩}{⟨false part⟩}`

Does `\DTLifnumeq` if both ⟨*arg1*⟩ and ⟨*arg2*⟩ are numerical, otherwise do `\DTLifstringeq` or `\DTLifstringeq*`.

```
\newcommand*{\DTLifeq}{\@ifstar\@sDTLifeq\@DTLifeq}
```

Unstarred version

```
\newcommand*{\@DTLifeq}[4]{%
\dtl@testbothnumerical{#1}{#2}%
\if@dtl@condition
 \DTLifnumeq{#1}{#2}{#3}{#4}%
\else
 \@DTLifstringeq{#1}{#2}{#3}{#4}%
\fi
}
```

Starred version

```
\newcommand*{\@sDTLifeq}[4]{%
\dtl@testbothnumerical{#1}{#2}%
\if@dtl@condition
 \DTLifnumeq{#1}{#2}{#3}{#4}%
\else
 \@sDTLifstringeq{#1}{#2}{#3}{#4}%
\fi
}
```

`\DTLifSubString` `\DTLifSubString{⟨string⟩}{⟨sub string⟩}{⟨true part⟩}{⟨false part⟩}`

If ⟨*sub string*⟩ is contained in ⟨*string*⟩ does ⟨*true part*⟩, otherwise does ⟨*false part*⟩.

```
\newcommand*{\DTLifSubString}[4]{%
\protected@edef\@dtl@tmp{\noexpand\dtl@testifsubstring
```

```
{#1}{#2}}%
\@dtl@tmp
\if@dtl@condition
 #3%
\else
 #4%
\fi
}
```

\dtl@testifsubstring

```
\newcommand*{\dtl@testifsubstring}[2]{%
\dtl@subnobrsp{#1}{\@dtl@argA}%
\dtl@subnobrsp{#2}{\@dtl@argB}%
\ifx\@dtl@argB\@empty
 \@dtl@conditiontrue
\else
 \ifx\@dtl@argA\@empty
   \@dtl@conditionfalse
 \else
   \dtl@teststartswith{#1}{#2}%
   \if@dtl@condition
   \else
     \DTLsubstituteall{\@dtl@argA}{ }{\space }%
     \expandafter\dtl@getfirst\@dtl@argA\end
     \expandafter\dtl@ifsingle\expandafter{\dtl@first}{%
       \expandafter\dtl@testifsubstring\expandafter{\dtl@rest}{#2}%
     }{%
       \protected@edef\@dtl@donext{\noexpand\dtl@testifsubstring
          {\dtl@first\dtl@rest}{\@dtl@argB}}%
       \@dtl@donext
     }%
   \fi
 \fi
\fi
}
```

\DTLifStartsWith

\DTLifStartsWith{⟨*string*⟩}{⟨*substring*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

If ⟨*string*⟩ starts with ⟨*substring*⟩, this does ⟨*true part*⟩, otherwise it does ⟨*false part*⟩.

```
\newcommand*{\DTLifStartsWith}[4]{%
\@dtl@conditionfalse
\protected@edef\@dtl@tmp{\noexpand\dtl@teststartswith{#1}{#2}}%
\@dtl@tmp
\if@dtl@condition
 #3%
\else
 #4%
\fi
}
```

\dtl@teststartswith

\dtl@teststartswith{⟨*string*⟩}{⟨*prefix*⟩}

Tests if ⟨*string*⟩ starts with ⟨*prefix*⟩. This sets \if@dtl@condition.

```
\newcommand*{\dtl@teststartswith}[2]{%
\dtl@subnobrsp{#1}{\@dtl@argA}%
\dtl@subnobrsp{#2}{\@dtl@argB}%
\ifx\@dtl@argA\@empty
 \ifx\@dtl@argB\@empty
   \@dtl@conditiontrue
 \else
   \@dtl@conditionfalse
 \fi
\else
  \ifx\@dtl@argB\@empty
   \@dtl@conditiontrue
  \else
    \DTLsubstituteall{\@dtl@argA}{ }{\space }%
    \DTLsubstituteall{\@dtl@argB}{ }{\space }%
    \expandafter\dtl@getfirst\@dtl@argA\end
    \let\dtl@firstA=\dtl@first
    \let\dtl@restA=\dtl@rest
    \expandafter\dtl@getfirst\@dtl@argB\end
    \let\dtl@firstB=\dtl@first
    \let\dtl@restB=\dtl@rest
    \expandafter\dtl@ifsingle\expandafter{\dtl@firstA}{%
    \expandafter\dtl@ifsingle\expandafter{\dtl@firstB}{%
    \expandafter\dtl@setcharcode\expandafter{\dtl@firstA}{\dtl@codeA}%
    \expandafter\dtl@setcharcode\expandafter{\dtl@firstB}{\dtl@codeB}%
    \ifnum\dtl@codeA=-1\relax
      \ifnum\dtl@codeB=-1\relax
         \protected@edef\dtl@donext{%
           \noexpand\dtl@teststartswith{\dtl@restA}{\dtl@restB}}%
         \dtl@donext
      \else
         \protected@edef\dtl@donext{%
           \noexpand\dtl@teststartswith
             {\dtl@restA}{\dtl@firstB\dtl@restB}}%
         \dtl@donext
      \fi
    \else
      \ifnum\dtl@codeB=-1\relax
         \protected@edef\dtl@donext{%
           \noexpand\dtl@teststartswith
             {\dtl@firstA\dtl@restA}{\dtl@restB}}%
         \dtl@donext
      \else
        \ifnum\dtl@codeA=\dtl@codeB
          \protected@edef\dtl@donext{%
            \noexpand\dtl@teststartswith{\dtl@restA}{\dtl@restB}}%
          \dtl@donext
        \else
          \@dtl@conditionfalse
        \fi
```

```
      \fi
    \fi
    }{%
    \protected@edef\dtl@donext{%
      \noexpand\dtl@teststartswith
        {\dtl@firstA\dtl@restA}{\dtl@firstB\dtl@restB}}%
    \dtl@donext
    }}{%
    \protected@edef\dtl@donext{%
      \noexpand\dtl@teststartswith
        {\dtl@firstA\dtl@restA}{\dtl@firstB\dtl@restB}}%
    }%
  \fi
\fi
}
```

\DTLifnumclosedbetween

\DTLifnumclosedbetween{⟨*num*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Determines if ⟨*min*⟩ ≤ ⟨*num*⟩ ≤ ⟨*max*⟩.

```
\newcommand*{\DTLifnumclosedbetween}[5]{%
\DTLconverttodecimal{#1}{\@dtl@numi}%
\DTLconverttodecimal{#2}{\@dtl@numii}%
\DTLconverttodecimal{#3}{\@dtl@numiii}%
\DTLifFPclosedbetween{\@dtl@numi}{\@dtl@numii}{\@dtl@numiii}{#4}{#5}%
}
```

\DTLifstringclosedbetween

\DTLifstringclosedbetween{⟨*string*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

String comparison (starred version ignores case)

```
\newcommand*{\DTLifstringclosedbetween}{%
\@ifstar\@sDTLifstringclosedbetween\@DTLifstringclosedbetween}
```

Unstarred version

```
\newcommand*{\@DTLifstringclosedbetween}[5]{%
\protected@edef\@dtl@tmpcmp{%
  \noexpand\dtlcompare{\noexpand\@dtl@tmpcount}{#1}{#2}}%
\@dtl@tmpcmp
\let\@dtl@dovalue\relax
\ifnum\@dtl@tmpcount<0\relax
  \def\@dtl@dovalue{#5}%
\fi
\ifx\@dtl@dovalue\relax
  \protected@edef\@dtl@tmpcmp{%
    \noexpand\dtlcompare{\noexpand\@dtl@tmpcount}{#1}{#3}}%
  \@dtl@tmpcmp
  \ifnum\@dtl@tmpcount>0\relax
    \def\@dtl@dovalue{#5}%
  \else
    \def\@dtl@dovalue{#4}%
  \fi
```

```
\fi
\@dtl@dovalue
}
```

Starred version

```
\newcommand*{\@sDTLifstringclosedbetween}[5]{%
\protected@edef\@dtl@tmpcmp{%
  \noexpand\dtlicompare{\noexpand\@dtl@tmpcount}{#1}{#2}}%
\@dtl@tmpcmp
\let\@dtl@dovalue\relax
\ifnum\@dtl@tmpcount<0\relax
  \def\@dtl@dovalue{#5}%
\fi
\ifx\@dtl@dovalue\relax
  \protected@edef\@dtl@tmpcmp{%
    \noexpand\dtlicompare{\noexpand\@dtl@tmpcount}{#1}{#3}}%
  \@dtl@tmpcmp
  \ifnum\@dtl@tmpcount>0\relax
    \def\@dtl@dovalue{#5}%
  \else
    \def\@dtl@dovalue{#4}%
  \fi
\fi
\@dtl@dovalue
}
```

`\DTLifclosedbetween` `\DTLifclosedbetween{`⟨*arg*⟩`}{`⟨*min*⟩`}{`⟨*max*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

Does `\DTLifnumclosedbetween` if `{`⟨*arg*⟩`}`, ⟨*min*⟩ and ⟨*max*⟩ are numerical, otherwise do `\DTLifstringclosedbetween` or `\DTLifstringclosedbetween*`.

```
\newcommand*{\DTLifclosedbetween}{%
\@ifstar\@sDTLifclosedbetween\@DTLifclosedbetween}
```

Unstarred version

```
\newcommand*{\@DTLifclosedbetween}[5]{%
\dtl@testbothnumerical{#2}{#3}%
\if@dtl@condition
  \dtl@ifsingle{#1}{%
    \edef\@dtl@tmp{#1}}{%
    \def\@dtl@tmp{#1}}%
  \expandafter\@dtl@checknumerical\expandafter{\@dtl@tmp}%
  \ifnum\@dtl@datatype>0\relax
    \DTLifnumclosedbetween{#1}{#2}{#3}{#4}{#5}%
  \else
    \@DTLifstringclosedbetween{#1}{#2}{#3}{#4}{#5}%
  \fi
\else
  \@DTLifstringclosedbetween{#1}{#2}{#3}{#4}{#5}%
\fi
}
```

Starred version

```
\newcommand*{\@sDTLifclosedbetween}[5]{%
```

```
\dtl@testbothnumerical{#2}{#3}%
\if@dtl@condition
  \dtl@ifsingle{#1}{%
    \edef\@dtl@tmp{#1}}{%
    \def\@dtl@tmp{#1}}%
  \expandafter\@dtl@checknumerical\expandafter{\@dtl@tmp}%
  \ifnum\@dtl@datatype>0\relax
    \DTLifnumclosedbetween{#1}{#2}{#3}{#4}{#5}%
  \else
    \@sDTLifstringclosedbetween{#1}{#2}{#3}{#4}{#5}%
  \fi
\else
  \@sDTLifstringclosedbetween{#1}{#2}{#3}{#4}{#5}%
\fi
}
```

\DTLifnumopenbetween

\DTLifnumopenbetween{⟨*num*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Determines if ⟨*min*⟩ < ⟨*num*⟩ < ⟨*max*⟩.

```
\newcommand*{\DTLifnumopenbetween}[5]{%
\DTLconverttodecimal{#1}{\@dtl@numi}%
\DTLconverttodecimal{#2}{\@dtl@numii}%
\DTLconverttodecimal{#3}{\@dtl@numiii}%
\DTLifFPopenbetween{\@dtl@numi}{\@dtl@numii}{\@dtl@numiii}{#4}{#5}%
}
```

\DTLifstringopenbetween

\DTLifstringopenbetween{⟨*string*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

String comparison (starred version ignores case)

```
\newcommand*{\DTLifstringopenbetween}{%
\@ifstar\@sDTLifstringopenbetween\@DTLifstringopenbetween}
```

Unstarred version:

```
\newcommand*{\@DTLifstringopenbetween}[5]{%
\protected@edef\@dtl@tmpcmp{%
  \noexpand\dtlcompare{\noexpand\@dtl@tmpcount}{#1}{#2}}%
\@dtl@tmpcmp
\let\@dtl@dovalue\relax
\ifnum\@dtl@tmpcount>0\relax
\else
  \def\@dtl@dovalue{#5}%
\fi
\ifx\@dtl@dovalue\relax
  \protected@edef\@dtl@tmpcmp{%
    \noexpand\dtlcompare{\noexpand\@dtl@tmpcount}{#1}{#3}}%
  \@dtl@tmpcmp
  \ifnum\@dtl@tmpcount<0\relax
    \def\@dtl@dovalue{#4}%
  \else
    \def\@dtl@dovalue{#5}%
```

```
    \fi
  \fi
  \@dtl@dovalue
  }
```

Starred version

```
  \newcommand*{\@sDTLifstringopenbetween}[5]{%
  \protected@edef\@dtl@tmpcmp{%
    \noexpand\dtlicompare{\noexpand\@dtl@tmpcount}{#1}{#2}}%
  \@dtl@tmpcmp
  \let\@dtl@dovalue\relax
  \ifnum\@dtl@tmpcount>0\relax
  \else
    \def\@dtl@dovalue{#5}%
  \fi
  \ifx\@dtl@dovalue\relax
    \protected@edef\@dtl@tmpcmp{%
      \noexpand\dtlicompare{\noexpand\@dtl@tmpcount}{#1}{#3}}%
    \@dtl@tmpcmp
    \ifnum\@dtl@tmpcount<0\relax
      \def\@dtl@dovalue{#4}%
    \else
      \def\@dtl@dovalue{#5}%
    \fi
  \fi
  \@dtl@dovalue
  }
```

\DTLifopenbetween

\DTLifopenbetween{⟨*arg*⟩}{⟨*min*⟩}{⟨*max*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Does \DTLifnumopenbetween if {⟨*arg*⟩}, ⟨*min*⟩ and ⟨*max*⟩ are numerical, otherwise do \DTLifstringopenbetween or \DTLifstringopenbetween*.

```
  \newcommand*{\DTLifopenbetween}{%
  \@ifstar\@sDTLifopenbetween\@DTLifopenbetween}
```

Unstarred version

```
  \newcommand*{\@DTLifopenbetween}[5]{%
  \dtl@testbothnumerical{#2}{#3}%
  \if@dtl@condition
    \dtl@ifsingle{#1}{%
      \edef\@dtl@tmp{#1}}{%
      \def\@dtl@tmp{#1}}%
    \expandafter\@dtl@checknumerical\expandafter{\@dtl@tmp}%
    \ifnum\@dtl@datatype>0\relax
      \DTLifnumopenbetween{#1}{#2}{#3}{#4}{#5}%
    \else
      \@DTLifstringopenbetween{#1}{#2}{#3}{#4}{#5}%
    \fi
  \else
    \@DTLifstringopenbetween{#1}{#2}{#3}{#4}{#5}%
  \fi
  }
```

Starred version

```
\newcommand*{\@sDTLifopenbetween}[5]{%
\dtl@testbothnumerical{#2}{#3}%
\if@dtl@condition
  \dtl@ifsingle{#1}{%
    \edef\@dtl@tmp{#1}}{%
    \def\@dtl@tmp{#1}}%
  \expandafter\@dtl@checknumerical\expandafter{\@dtl@tmp}%
  \ifnum\@dtl@datatype>0\relax
    \DTLifnumopenbetween{#1}{#2}{#3}{#4}{#5}%
  \else
    \@sDTLifstringopenbetween{#1}{#2}{#3}{#4}{#5}%
  \fi
\else
  \@sDTLifstringopenbetween{#1}{#2}{#3}{#4}{#5}%
\fi
}
```

`\DTLifFPopenbetween`

`\DTLifFPopenbetween{`⟨*num*⟩`}{`⟨*min*⟩`}{`⟨*max*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

Determines if ⟨*min*⟩ < ⟨*num*⟩ < ⟨*max*⟩ where all arguments are in standard fixed point notation.

```
\newcommand*{\DTLifFPopenbetween}[5]{%
  \let\@dtl@dovalue\relax
  \FPifgt{#1}{#2}%
  \else
    \def\@dtl@dovalue{#5}%
  \fi
  \FPiflt{#1}{#3}%
    \ifx\@dtl@dovalue\relax
      \def\@dtl@dovalue{#4}%
    \fi
  \else
    \def\@dtl@dovalue{#5}%
  \fi
\@dtl@dovalue
}
```

`\DTLifFPclosedbetween`

`\DTLifFPclosedbetween{`⟨*num*⟩`}{`⟨*min*⟩`}{`⟨*max*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}`

Determines if ⟨*min*⟩ ≤ ⟨*num*⟩ ≤ ⟨*max*⟩.

```
\newcommand*{\DTLifFPclosedbetween}[5]{%
  \let\@dtl@dovalue\relax
  \FPifgt{#1}{#3}%
    \def\@dtl@dovalue{#5}%
  \fi
  \FPiflt{#1}{#2}%
    \ifx\@dtl@dovalue\relax
      \def\@dtl@dovalue{#5}%
    \fi
```

```
    \else
      \def\@dtl@dovalue{#4}%
    \fi
  \@dtl@dovalue
  }
```

The following conditionals are only meant to be used within \DTLforeach as they depend on the counter DTLrow⟨*n*⟩.

\DTLiffirstrow

\DTLiffirstrow{⟨*true part*⟩}{⟨*false part*⟩}

Test if the current row is the first row. (This takes ⟨*condition*⟩, the optional argument of \DTLforeach, into account, so it may not correspond to row 1 of the database.) Can only be used in \DTLforeachrow.

```
\newcommand{\DTLiffirstrow}[2]{%
  \PackageError{datatool}{\string\DTLiffirstrow\space can only
  be used inside \string\DTLforeach}{}%
}
```

\DTLiflastrow

\DTLiflastrow{⟨*true part*⟩}{⟨*false part*⟩}

Checks if the current row is the last row of the database. It doesn't take the condition (the optional argument of \DTLforeach) into account, so its possible it may never do ⟨*true part*⟩, as the last row of the database may not meet the condition. It is therefore not very useful and is confusing since it behaves differently to \DTLiffirstrow which does take the condition into account, so I have removed its description from the main part of the manual. If you need to use the optional argument of \DTLforeach, you will first have to iterate through the database to count up the number of rows which meet the condition, and then do another pass, checking if the current row has reached that number.

```
\newcommand{\DTLiflastrow}[2]{%
  \PackageError{datatool}{\string\DTLiflastrow\space can only
  be used inside \string\DTLforeach}{}%
}
```

\DTLifoddrow

\DTLifoddrow{⟨*true part*⟩}{⟨*false part*⟩}

Determines whether the current row is odd (takes the optional argument of \DTLforeach into account.)

```
\newcommand{\DTLifoddrow}[2]{%
  \PackageError{datatool}{\string\DTLifoddrow\space can only
  be used inside \string\DTLforeach}{}%
}
```

10.4 ifthen Conditionals

The following commands provide conditionals \DTLis... which can be used in \ifthenelse. First need to define a new conditional:

`\if@dtl@condition`

```
\newif\if@dtl@condition
```

`\dtl@testlt` Command to test if first argument is less than second argument. If either argument is a string, a case sensitive string comparison is used instead. This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testlt}[2]{%
\DTLiflt{#1}{#2}{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

`\DTLislt` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLislt}[2]{%
\TE@throw\noexpand\dtl@testlt{#1}{#2}\noexpand\if@dtl@condition}
```

`\dtl@testiclt` Command to test if first argument is less than second argument. If either argument is a string, a case insensitive string comparison is used instead. This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testiclt}[2]{%
\@sDTLiflt{#1}{#2}{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

`\DTLisilt` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisilt}[2]{%
\TE@throw\noexpand\dtl@testiclt{#1}{#2}\noexpand\if@dtl@condition}
```

`\dtl@testgt` Command to test if first number is greater than second number. This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testgt}[2]{%
\DTLifgt{#1}{#2}{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

`\DTLisgt` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisgt}[2]{%
\TE@throw\noexpand\dtl@testgt{#1}{#2}\noexpand\if@dtl@condition}
```

`\dtl@testicgt` Command to test if first number is greater than second number (ignores case). This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testicgt}[2]{%
\@sDTLifgt{#1}{#2}{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

`\DTLisigt` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisigt}[2]{%
\TE@throw\noexpand\dtl@testicgt{#1}{#2}\noexpand\if@dtl@condition}
```

`\dtl@testeq` Command to test if first number is equal to the second number. This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testeq}[2]{%
\DTLifeq{#1}{#2}{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

`\DTLiseq` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLiseq}[2]{%
\TE@throw\noexpand\dtl@testeq{#1}{#2}\noexpand\if@dtl@condition}
```

`\dtl@testiceq` Command to test if first number is equal to the second number (ignores case). This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testiceq}[2]{%
\@sDTLifeq{#1}{#2}{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

`\DTLisieq` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisieq}[2]{%
\TE@throw\noexpand\dtl@testiceq{#1}{#2}\noexpand\if@dtl@condition}
```

`\DTLisSubString` Tests if second argument is contained in first argument.

```
\newcommand*{\DTLisSubString}[2]{%
\TE@throw\noexpand\dtl@testifsubstring{#1}{#2}%
\noexpand\if@dtl@condition}
```

`\DTLisPrefix` Tests if first argument starts with second argument.

```
\newcommand*{\DTLisPrefix}[2]{%
\TE@throw\noexpand\dtl@teststartswith{#1}{#2}%
\noexpand\if@dtl@condition}
```

`\dtl@testclosedbetween` Command to test if first value lies between second and third values. (End points included, case sensitive.) This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testclosedbetween}[3]{%
\DTLifclosedbetween{#1}{#2}{#3}{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

`\DTLisclosedbetween` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisclosedbetween}[3]{%
\TE@throw\noexpand\dtl@testclosedbetween{#1}{#2}{#3}%
\noexpand\if@dtl@condition}
```

`\dtl@testiclosedbetween` Command to test if first value lies between second and third values. (End points included, case ignored.) This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testiclosedbetween}[3]{%
\@sDTLifclosedbetween{#1}{#2}{#3}{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

`\DTLisiclosedbetween` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisiclosedbetween}[3]{%
\TE@throw\noexpand\dtl@testiclosedbetween{#1}{#2}{#3}%
\noexpand\if@dtl@condition}
```

`\dtl@testopenbetween` Command to test if first value lies between second and third values. (End points excluded, case sensitive.) This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testopenbetween}[3]{%
\DTLifopenbetween{#1}{#2}{#3}{\@dtl@conditiontrue
}{\@dtl@conditionfalse}}
```

`\DTLisopenbetween` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisopenbetween}[3]{%
\TE@throw\noexpand\dtl@testopenbetween{#1}{#2}{#3}%
\noexpand\if@dtl@condition}
```

`\dtl@testiopenbetween` Command to test if first value lies between second and third values. (End points excluded, case ignored.) This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testiopenbetween}[3]{%
\@sDTLifopenbetween{#1}{#2}{#3}{\@dtl@conditiontrue
}{\@dtl@conditionfalse}}
```

`\DTLisiopenbetween` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisiopenbetween}[3]{%
\TE@throw\noexpand\dtl@testiopenbetween{#1}{#2}{#3}%
\noexpand\if@dtl@condition}
```

`\dtl@testclosedbetween` Command to test if first number lies between second and third numbers. (End points included, all arguments are fixed point numbers in standard format.) This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testFPclosedbetween}[3]{%
\DTLifFPclosedbetween{#1}{#2}{#3}%
{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

Provide conditional command for use in `\ifthenelse`

`\DTLisFPclosedbetween`

```
\newcommand*{\DTLisFPclosedbetween}[3]{%
\TE@throw\noexpand\dtl@testFPclosedbetween{#1}{#2}{#3}%
\noexpand\if@dtl@condition}
```

`\dtl@testopenbetween` Command to test if first number lies between second and third numbers. (End points excluded, all arguments are fixed point numbers in standard format.) This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testFPopenbetween}[3]{%
\DTLifFPopenbetween{#1}{#2}{#3}{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

`\DTLisFPopenbetween` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisFPopenbetween}[3]{%
\TE@throw\noexpand\dtl@testFPopenbetween{#1}{#2}{#3}%
\noexpand\if@dtl@condition}
```

`\dtl@testFPislt` Command to test if first number is less than second number where both numbers are in standard format. This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testFPislt}[2]{%
\FPiflt{#1}{#2}\@dtl@conditiontrue\else\@dtl@conditionfalse\fi}
```

`\DTLisFPlt` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisFPlt}[2]{%
\TE@throw\noexpand\dtl@testFPislt{#1}{#2}%
\noexpand\if@dtl@condition}
```

`\dtl@testFPisgt` Command to test if first number is greater than second number where both numbers are in standard format. This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testFPisgt}[2]{%
\FPifgt{#1}{#2}\@dtl@conditiontrue\else\@dtl@conditionfalse\fi}
```

`\DTLisFPgt` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisFPgt}[2]{%
\TE@throw\noexpand\dtl@testFPisgt{#1}{#2}%
\noexpand\if@dtl@condition}
```

`\dtl@testFPiseq` Command to test if two numbers are equal, where both numbers are in standard decimal format

```
\newcommand*{\dtl@testFPiseq}[2]{%
\FPifeq{#1}{#2}\@dtl@conditiontrue\else\@dtl@conditionfalse\fi}
```

`\DTLisFPeq` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisFPeq}[2]{%
\TE@throw\noexpand\dtl@testFPiseq{#1}{#2}%
\noexpand\if@dtl@condition}
```

`\dtl@testFPislteq` Command to test if first number is less than or equal to second number where both numbers are in standard format. This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testFPislteq}[2]{%
\FPiflt{#1}{#2}\@dtl@conditiontrue\else\@dtl@conditionfalse\fi
\if@dtl@condition
\else
 \dtl@testFPiseq{#1}{#2}%
\fi
}
```

`\DTLisFPlteq` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisFPlteq}[2]{%
\TE@throw\noexpand\dtl@testFPislteq{#1}{#2}%
\noexpand\if@dtl@condition}
```

`\dtl@testFPisgteq` Command to test if first number is greater than or equal to second number where both numbers are in standard format. This sets `\if@dtl@condition`.

```
\newcommand*{\dtl@testFPisgteq}[2]{%
\FPifgt{#1}{#2}\@dtl@conditiontrue\else\@dtl@conditionfalse\fi
\if@dtl@condition
\else
 \dtl@testFPiseq{#1}{#2}%
\fi
}
```

`\DTLisFPgteq` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisFPgteq}[2]{%
\TE@throw\noexpand\dtl@testFPisgteq{#1}{#2}%
\noexpand\if@dtl@condition}
```

`\dtl@teststring` Command to test if argument is a string. This sets `\if@dtl@condition`

```
\newcommand*{\dtl@teststring}[1]{%
\DTLifstring{#1}{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

`\DTLisstring` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisstring}[1]{%
\TE@throw\noexpand\dtl@teststring{#1}\noexpand\if@dtl@condition}
```

`\dtl@testnumerical` Command to test if argument is a numerical. This sets `\if@dtl@condition`

```
\newcommand*{\dtl@testnumerical}[1]{%
\DTLifnumerical{#1}{\@dtl@conditiontrue}{\@dtl@conditionfalse}%
}
```

`\DTLisnumerical` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisnumerical}[1]{%
\TE@throw\noexpand\dtl@testnumerical{#1}\noexpand\if@dtl@condition}
```

`\dtl@testint` Command to test if argument is an integer. This sets `\if@dtl@condition`

```
\newcommand*{\dtl@testint}[1]{%
\DTLifint{#1}{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

`\DTLisint` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisint}[1]{%
\TE@throw\noexpand\dtl@testint{#1}\noexpand\if@dtl@condition}
```

`\dtl@testreal` Command to test if argument is a real. This sets `\if@dtl@condition`

```
\newcommand*{\dtl@testreal}[1]{%
\DTLifreal{#1}{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

`\DTLisreal` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLisreal}[1]{%
\TE@throw\noexpand\dtl@testreal{#1}\noexpand\if@dtl@condition}
```

`\dtl@testcurrency` Command to test if argument is a currency. This sets `\if@dtl@condition`

```
\newcommand*{\dtl@testcurrency}[1]{%
\DTLifcurrency{#1}{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

`\DTLiscurrency` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLiscurrency}[1]{%
\TE@throw\noexpand\dtl@testcurrency{#1}\noexpand\if@dtl@condition}
```

`\dtl@testcurrencyunit` Command to test if argument is a currency with given unit. This sets `\if@dtl@condition`

```
\newcommand*{\dtl@testcurrencyunit}[2]{%
\DTLifcurrencyunit{#1}{#2}{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

`\DTLiscurrencyunit` Provide conditional command for use in `\ifthenelse`

```
\newcommand*{\DTLiscurrencyunit}[2]{%
\TE@throw\noexpand\dtl@testcurrencyunit{#1}{#2}%
\noexpand\if@dtl@condition}
```

10.5 Defining New Databases

As from v2.0, the internal structure of the database has changed to make it more efficient.[11] The database is now stored in a token register instead of a macro. Each row is represented as:

`\db@row@elt@w` `\db@row@id@w` ⟨*row idx*⟩`\db@row@id@end@` ⟨*column data*⟩ `\db@row@id@w` ⟨*row idx*⟩`\db@row@id@end@` `\db@row@elt@end@`

[11] Thanks to Morten Høgholm for the suggestion.

where ⟨*row idx*⟩ is the row index and ⟨*column data*⟩ is the data for each column in the row. Each column for a given row is stored as:

`\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 entry for the given column and row.

Each row only has an associated index, but columns have a unique identifying key as well as an associated index. Columns also have an associated data type which may be: 0 (column contains strings), 1 (column contains integers), 2 (column contains real numbers), 3 (column contains currency) or ⟨*empty*⟩ (column contains no data). Since the key sometimes has to be expanded, a header is also available in the event that the user wants to use `\DTLdisplaydb` or `\DTLdisplaylongdb` and requires a column header that would cause problems if used as a key. The general column information is stored in a token register where each column has information stored in the form:

`\db@plist@elt@w` `\db@col@id@w` ⟨*index*⟩`\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` ⟨*type*⟩`\db@header@id@end@` `\db@col@id@w` ⟨*index*⟩`\db@col@id@end@` `\db@plist@elt@end@`

The column name (⟨*key*⟩) is mapped to the column index using `\dtl@ci@`⟨*db*⟩`@`⟨*key*⟩ where ⟨*db*⟩ is the database name.

`\DTLnewdb` `\DTLnewdb{`⟨*name*⟩`}` initialises a database called ⟨*name*⟩.

```
\newcommand*{\DTLnewdb}[1]{%
```

Check if there is already a database with this name.

```
  \DTLifdbexists{#1}%
  {%
     \PackageError{datatool}{Database ‘#1’ already exists}{}%
  }%
  {%
```

Define new database. Add information message if in verbose mode.

```
     \dtl@message{Creating database ‘#1’}%
```

Define token register used to store the contents of the database.

```
     \expandafter\newtoks\csname dtldb@#1\endcsname
```

Define token register used to store the column header information.

```
     \expandafter\newtoks\csname dtlkeys@#1\endcsname{}%
```

Define count register used to store the row count.

```
     \expandafter\newcount\csname dtlrows@#1\endcsname
```

Define count register used to store the column count.

```
     \expandafter\newcount\csname dtlcols@#1\endcsname
  }%
}
```

`\DTLrowcount` `\DTLrowcount{`⟨*db name*⟩`}`

The number of rows in the database called ⟨*db name*⟩. (Doesn't check if database exists.)

```
\newcommand*{\DTLrowcount}[1]{%
\expandafter\number\csname dtlrows@#1\endcsname}
```

\DTLcolumncount

\DTLcolumncount{⟨*db name*⟩}

The number of columns in the database called ⟨*db name*⟩. (Doesn't check if database exists.)

```
\newcommand*{\DTLcolumncount}[1]{%
\expandafter\number\csname dtlcols@#1\endcsname}
```

\DTLifdbempty

\DTLifdbempty{⟨*name*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Check if named database is empty (i.e. no rows have been added).

```
\newcommand{\DTLifdbempty}[3]{%
  \DTLifdbexists{#1}%
    {\@DTLifdbempty{#1}{#2}{#3}}%
    {\PackageError{Can't check if database `#1' is empty:
       database doesn't exist}{}{}}%
}
```

\@DTLifdbempty

\@sDTLifdbempty{⟨*name*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Check if named existing database is empty. (No check performed to determine if the database exists.)

```
\newcommand{\@DTLifdbempty}[3]{%
  \expandafter\ifnum\csname dtlrows@#1\endcsname=0\relax
    #2%
  \else
    #3%
  \fi
}
```

\DTLnewrow

\DTLnewrow{⟨*db name*⟩}

Add a new row to named database. The starred version doesn't check for the existence of the database.

```
\newcommand*{\DTLnewrow}{%
  \@ifstar\@sDTLnewrow\@DTLnewrow
}
```

\@DTLnewrow

\@DTLnewrow{⟨*db name*⟩}

Add a new row to named database. (Checks for the existance of the database.)

```
\newcommand*{\@DTLnewrow}[1]{%
\DTLifdbexists{#1}%
  {\@sDTLnewrow{#1}}%
  {\PackageErrors{datatool}{Can't add new row to database `#1':
     database doesn't exist}{}}%
}
```

\@sDTLnewrow

\@DTLnewrow{⟨*db name*⟩}

Add a new row to named existing database. (No check performed to determine if the database exists.)

```
\newcommand*{\@sDTLnewrow}[1]{%
```

Increment row count.

```
  \global\advance\csname dtlrows@#1\endcsname by 1\relax
```

Append an empty row to the database

```
  \toks@gput@right@cx{dtldb@#1}{%
    \noexpand\db@row@elt@w%
      \noexpand\db@row@id@w \number\csname dtlrows@#1\endcsname
      \noexpand\db@row@id@end@%
      \noexpand\db@row@id@w \number\csname dtlrows@#1\endcsname
      \noexpand\db@row@id@end@%
    \noexpand\db@row@elt@end@%
  }%
```

Display message on terminal and log file if in verbose mode.

```
  \dtl@message{New row added to database `#1'}%
}
```

\dtlcolumnnum

Count register to keep track of column index.

```
\newcount\dtlcolumnnum
```

\dtlrownum

Count register to keep track of row index.

```
\newcount\dtlrownum
```

\DTLifhaskey

\DTLifhaskey⟨*db name*⟩⟨*key*⟩⟨*true part*⟩⟨*false part*⟩

Checks if the named database ⟨*db name*⟩ has a column with label ⟨*key*⟩. If column exists, do ⟨*true part*⟩ otherwise do ⟨*false part*⟩. The starred version doesn't check if the named database exists.

```
\newcommand*{\DTLifhaskey}{\@ifstar\@sDTLifhaskey\@DTLifhaskey}
```

\@DTLifhaskey

Unstarred version of \DTLifhaskey

```
\newcommand{\@DTLifhaskey}[4]{%
  \DTLifdbexists{#1}%
  {%
    \@sDTLifhaskey{#1}{#2}{#3}{#4}%
  }%
  {%
    \PackageError{datatool}{Database `#1' doesn't exist}{}%
  }%
}
```

\@sDTLifhaskey

Starred version of \DTLifhaskey

```
\newcommand{\@sDTLifhaskey}[4]{%
  \@ifundefined{dtl@ci@#1@#2}%
  {%
```

Key not defined

```
      #4%
    }%
    {%
```

Key defined

```
      #3%
    }%
  }
```

\DTLgetcolumnindex

\DTLgetcolumnindex{⟨*cs*⟩}{⟨*db*⟩}{⟨*key*⟩}

Gets index for column with label ⟨*key*⟩ from database ⟨*db*⟩ and stores in ⟨*cs*⟩ which must be a control sequence. Unstarred version checks if database and key exist, unstarred version doesn't perform any checks.

```
\newcommand*{\DTLgetcolumnindex}{%
  \@ifstar\@sdtl@getcolumnindex\@dtl@getcolumnindex
}
```

\@dtl@getcolumnindex

Unstarred version of \DTLgetcolumnindex

```
\newcommand*{\@dtl@getcolumnindex}[3]{%
```

Check if database exists.

```
  \DTLifdbexists{#2}%
  {%
```

Database exists. Now check if key exists.

```
    \@sDTLifhaskey{#2}{#3}%
    {%
```

Key exists so go ahead and get column index.

```
      \@sdtl@getcolumnindex{#1}{#2}{#3}%
    }%
    {%
```

Key doesn't exists in named database.

```
      \PackageError{datatool}{Database ‘#2’ doesn’t contain
       key ‘#3’}{}%
    }%
  }%
  {%
```

Named database doesn't exist.

```
    \PackageError{datatool}{Database ‘#2’ doesn’t exist}{}%
  }%
}
```

\@dtl@getcolumnindex

Starred version of \DTLgetcolumnindex.

```
\newcommand*{\@sdtl@getcolumnindex}[3]{%
  \expandafter\let\expandafter#1\csname dtl@ci@#2@#3\endcsname
}
```

\dtl@columnindex

\dtl@columnindex{⟨*db*⟩}{⟨*key*⟩}

Column index corresponding to ⟨*key*⟩ in database ⟨*db*⟩. (No check for existance of database or key.)

```
\newcommand*{\dtl@columnindex}[2]{%
  \csname dtl@ci@#1@#2\endcsname
}
```

\DTLgetkeyforcolumn

\DTLgetkeyforcolumn{⟨*key cs*⟩}{⟨*db*⟩}{⟨*column index*⟩}

Gets the key associated with the given column index and stores in ⟨*key cs*⟩. Unstarred version doesn't perform checks.

```
\newcommand*{\DTLgetkeyforcolumn}{%
  \@ifstar\@sdtlgetkeyforcolumn\@dtlgetkeyforcolumn}
```

\@dtlgetkeyforcolumn

```
\newcommand*{\@dtlgetkeyforcolumn}[3]{%
  \DTLifdbexists{#2}%
  {%
```

Check if index is in range.

```
    \ifnum#3<1\relax
      \PackageError{datatool}{Invalid column index \number#3}{%
      Column indices start at 1}%
    \else
      \expandafter\ifnum\csname dtlcols@#2\endcsname<#3\relax
        \PackageError{datatool}{Index \number#3\space out of
        range for database ‘#2’}{Database ‘#2’ only has
        \expandafter\number\csname dtlcols@#2\endcsname\space
        columns}%
      \else
        \@sdtlgetkeyforcolumn{#1}{#2}{#3}%
      \fi
    \fi
  }%
  {%
    \PackageError{datatool}{Database ‘#2’ doesn't exists}{}%
  }%
}
```

\@sdtlgetkeyforcolumn

\@sdtlgetkeyforcolumn{⟨*key cs*⟩}{⟨*db*⟩}{⟨*column index*⟩}

Gets the key associated with the given column index and stores in ⟨*key cs*⟩

```
\newcommand*{\@sdtlgetkeyforcolumn}[3]{%
  \edef\@dtl@dogetkeyforcolumn{\noexpand\@dtl@getkeyforcolumn
    {\noexpand#1}{#2}{\number#3}}%
  \@dtl@dogetkeyforcolumn
}
```

`\@dtl@getkeyforcolumn` Column index must be fully expanded before use.

```
\newcommand*{\@dtl@getkeyforcolumn}[3]{%
  \def\@dtl@get@keyforcolumn##1% before stuff
    \db@plist@elt@w% start of block
    \db@col@id@w #3\db@col@id@end@% index
    \db@key@id@w ##2\db@key@id@end@% key
    \db@type@id@w ##3\db@type@id@end@% data type
    \db@header@id@w ##4\db@header@id@end@% header
    \db@col@id@w #3\db@col@id@end@% index
    \db@plist@elt@end@% end of block
    ##5\q@nil{\def#1{##2}}%
  \edef\@dtl@tmp{\expandafter\the\csname dtldb@#2\endcsname}%
  \expandafter\@dtl@getkeyforcolumn\@dtl@tmp
    \db@plist@elt@w% start of block
    \db@col@id@w #3\db@col@id@end@ index
    \db@key@id@w \@nil\db@key@id@end@% key
    \db@type@id@w \db@type@id@end@% data type
    \db@header@id@w \db@header@id@end@% header
    \db@col@id@w #3\db@col@id@end@% index
    \db@plist@elt@end@% end of block
    \q@nil
}
```

Define some commands to indicate the various data types a database may contain.

`\DTLunsettype` Unknown data type. (All entries in the column are blank so the type can't be determined.)

```
\def\DTLunsettype{}
```

`\DTLstringtype` Data type representing strings.

```
\def\DTLstringtype{0}
```

`\DTLinttype` Data type representing integers.

```
\def\DTLinttype{1}
```

`\DTLrealtype` Data type representing real numbers.

```
\def\DTLrealtype{2}
```

`\DTLcurrencytype` Data type representing currency.

```
\def\DTLcurrencytype{3}
```

`\DTLgetdatatype` `\DTLgetdatatype{⟨cs⟩}{⟨db⟩}{⟨key⟩}`

Gets data type associated with column labelled ⟨*key*⟩ in database ⟨*db*⟩ and stores in ⟨*cs*⟩. Type may be: ⟨*empty*⟩ (unset), 0 (string), 1 (int), 2 (real), 3 (currency). Unstarred version checks if the database and key exist, starred version doesn't.

```
\newcommand*{\DTLgetdatatype}{%
  \@ifstar\@sdtlgetdatatype\@dtlgetdatatype
}
```

`\@dtlgetdatatype` Unstarred version of `\DTLgetdatatype`.

```
\newcommand*{\@dtlgetdatatype}[3]{%
```

Check if database exists.

```
  \DTLifdbexists{#2}%
  {%
```

Check if key exists in this database.

```
    \@sDTLifhaskey{#2}{#3}%
    {%
```

Get data type for this database and key.

```
      \@sdtlgetdatatype{#1}{#2}{#3}%
    }%
    {%
```

Key doesn't exist in this database.

```
      \PackageError{datatool}{Key ‘#3’ undefined in database ‘#2’}{}%
    }%
  }%
  {%
```

Database doesn't exist.

```
    \PackageError{datatool}{Database ‘#2’ doesn't exist}{}%
  }%
}
```

`\@sdtlgetdatatype` Starred version of `\DTLgetdatatype`. This ensures that the key is fully expanded before begin passed to `\@dtl@getdatatype`.

```
\newcommand*{\@sdtlgetdatatype}[3]{%
  \edef\@dtl@dogetdata{\noexpand\@dtl@getdatatype{\noexpand#1}%
   {\expandafter\the\csname dtlkeys@#2\endcsname}%
   {\dtl@columnindex{#2}{#3}}}%
  \@dtl@dogetdata
}
```

`\@dtl@getdatatype` `\@dtl@getdatatype{`⟨*cs*⟩`}{`⟨*data specs*⟩`}{`⟨*column index*⟩`}`

Column index must be expanded.

```
\newcommand*{\@dtl@getdatatype}[3]{%
  \def\@dtl@get@keydata##1% stuff before
    \db@plist@elt@w% start of key block
     \db@col@id@w #3\db@col@id@end@% column index
      \db@key@id@w ##2\db@key@id@end@% key id
      \db@type@id@w ##3\db@type@id@end@% data type
      \db@header@id@w ##4\db@header@id@end@% header
     \db@col@id@w #3\db@col@id@end@% column index
    \db@plist@elt@end@% end of key block
    ##5% stuff afterwards
    \q@nil{\def#1{##3}}%
  \@dtl@get@keydata#2\q@nil
}
```

\@dtl@getprops

\@dtl@getprops{⟨*key cs*⟩}{⟨*type cs*⟩}{⟨*header toks*⟩}{⟨*before toks*⟩}{⟨*after toks*⟩}{⟨*data specs*⟩}{⟨*column index*⟩}

Column index must be expanded.

```
\newcommand*{\@dtl@getprops}[7]{%
  \def\@dtl@get@keydata##1% stuff before
    \db@plist@elt@w% start of key block
     \db@col@id@w #7\db@col@id@end@% column index
      \db@key@id@w ##2\db@key@id@end@% key id
      \db@type@id@w ##3\db@type@id@end@% data type
      \db@header@id@w ##4\db@header@id@end@% header
     \db@col@id@w #7\db@col@id@end@% column index
    \db@plist@elt@end@% end of key block
    ##5% stuff afterwards
    \q@nil{%
      \def#1{##2}% key
      \def#2{##3}% data type
      #3={##4}% header
      #4={##1}% before stuff
      #5={##5}% after stuff
    }%
  \@dtl@get@keydata#6\q@nil
}
```

\@dtl@before

```
\newtoks\@dtl@before
```

\@dtl@after

```
\newtoks\@dtl@after
```

\@dtl@colhead

```
\newtoks\@dtl@colhead
```

\@dtl@updatekeys

\@dtl@updatekeys{⟨*db*⟩}{⟨*key*⟩}{⟨*value*⟩}

Adds key to database's key list if it doesn't exist. The value is used to update the data type associated with that key. Key must be fully expanded. Doesn't check if database exists.

```
\newcommand*{\@dtl@updatekeys}[3]{%
```

Check if key already exists

```
  \@sDTLifhaskey{#1}{#2}%
  {%
```

Key exists, may need to update data type. First get the column index.

```
    \expandafter\dtlcolumnnum\expandafter
      =\dtl@columnindex{#1}{#2}\relax
```

Get the properties for this column

```
    \edef\@dtl@dogetprops{\noexpand\@dtl@getprops
      {\noexpand\@dtl@key}{\noexpand\@dtl@type}%
      {\noexpand\@dtl@colhead}{\noexpand\@dtl@before}%
```

```
      {\noexpand\@dtl@after}{\the\csname dtlkeys@#1\endcsname}%
      {\number\dtlcolumnnum}}
    \@dtl@dogetprops
```

Is the value empty?

```
    \def\@dtl@tmp{#3}%
    \ifx\@dtl@tmp\@empty
```

Leave data type as it is

```
    \else
```

Make a copy of current data type

```
      \let\@dtl@oldtype\@dtl@type
```

Check the data type for this entry (stored in `\@dtl@datatype`)

```
      \@dtl@checknumerical{#3}%
```

If this column currently has no data type assigned to it then use the new type.

```
      \ifx\@dtl@type\@empty
        \edef\@dtl@type{\number\@dtl@datatype}%
      \else
```

This column already has an associated data type but it may need updating.

```
        \ifcase\@dtl@datatype % string
```

String overrides all other types

```
          \def\@dtl@type{0}%
        \or % int
```

All other types override int, so leave it as it is

```
        \or % real
```

Real overrides int, but not currency or string

```
          \ifnum\@dtl@type=1\relax
            \def\@dtl@type{2}%
          \fi
        \or % currency
```

Currency overrides int and real but not string

```
          \ifnum\@dtl@type>0\relax
            \def\@dtl@type{3}%
          \fi
        \fi
      \fi
```

Has the data type been updated?

```
      \ifx\@dtl@oldtype\@dtl@type
```

No change needed

```
      \else
```

Update required

```
        \toks@gconcat@middle@cx{dtlkeys@#1}%
        {\@dtl@before}%
        {%
          \noexpand\db@plist@elt@w% start of key block
            \noexpand\db@col@id@w \the\dtlcolumnnum
              \noexpand\db@col@id@end@% column index
            \noexpand\db@key@id@w #2\noexpand\db@key@id@end@% key id
```

```
            \noexpand\db@type@id@w \@dtl@type
              \noexpand\db@type@id@end@% data type
            \noexpand\db@header@id@w \the\@dtl@colhead
              \noexpand\db@header@id@end@% header
            \noexpand\db@col@id@w \the\dtlcolumnnum
              \noexpand\db@col@id@end@% column index
            \noexpand\db@plist@elt@end@% end of key block
          }%
          {\@dtl@after}%
        \fi
      \fi
    }%
    {%
```

Key doesn't exist. Increment column count.

```
      \expandafter\global\expandafter\advance
        \csname dtlcols@#1\endcsname by 1\relax
      \dtlcolumnnum=\csname dtlcols@#1\endcsname\relax
```

Set column index for this key

```
      \expandafter\xdef\csname dtl@ci@#1@#2\endcsname{%
        \number\dtlcolumnnum}%
```

Get data type for this entry (stored in `\@dtl@datatype`)

```
      \def\@dtl@tmp{#3}%
      \ifx\@dtl@tmp\@empty
        \edef\@dtl@type{}% don't know data type yet
      \else
        \@dtl@checknumerical{#3}%
        \edef\@dtl@type{\number\@dtl@datatype}%
      \fi
```

Append to property list

```
      \toks@gput@right@cx{dtlkeys@#1}%
      {%
        \noexpand\db@plist@elt@w
        \noexpand\db@col@id@w \the\dtlcolumnnum
          \noexpand\db@col@id@end@
        \noexpand\db@key@id@w #2\noexpand\db@key@id@end@
        \noexpand\db@type@id@w \@dtl@type
          \noexpand\db@type@id@end@
        \noexpand\db@header@id@w #2\noexpand\db@header@id@end@
        \noexpand\db@col@id@w \the\dtlcolumnnum
          \noexpand\db@col@id@end@
        \noexpand\db@plist@elt@end@
      }%
    }%
  }
```

`\DTLsetheader` `\DTLsetheader{⟨db⟩}{⟨key⟩}{⟨header⟩}`

Sets header for column given by ⟨*key*⟩ in database ⟨*db*⟩. Starred version doesn't check for existance of database or key.

```
\newcommand*{\DTLsetheader}{\@ifstar\@sDTLsetheader\@DTLsetheader}
```

\@DTLsetheader Unstarred version

```
\newcommand*{\@DTLsetheader}[3]{%
```

Check if database exists

```
  \DTLifdbexists{#1}%
  {%
```

Check if key exists.

```
    \@sDTLifhaskey{#1}{#2}%
    {%
      \@sDTLsetheader{#1}{#2}{#3}%
    }%
    {%
      \PackageError{datatool}{Database ‘#1’ doesn’t contain key
      ‘#2’}{}%
    }%
  }%
  {%
    \PackageError{datatool}{Database ‘#1’ doesn’t exist}{}%
  }%
}
```

\@sDTLsetheader Starred version

```
\newcommand*{\@sDTLsetheader}[3]{%
  \expandafter\dtlcolumnnum\expandafter
    =\dtl@columnindex{#1}{#2}\relax
  \@dtl@setheaderforindex{#1}{\dtlcolumnnum}{#3}%
}
```

\@dtl@setheaderforindex

\@dtl@setheaderforindex{⟨*db*⟩}{⟨*column index*⟩}{⟨*header*⟩}

Sets the header for column given by ⟨*column index*⟩ in database ⟨*db*⟩. The header must be expanded.

```
\newcommand*{\@dtl@setheaderforindex}[3]{%
```

Get the properties for this column

```
  \edef\@dtl@dogetprops{\noexpand\@dtl@getprops
    {\noexpand\@dtl@key}{\noexpand\@dtl@type}%
    {\noexpand\@dtl@colhead}{\noexpand\@dtl@before}%
    {\noexpand\@dtl@after}{\the\csname dtlkeys@#1\endcsname}%
    {\number#2}}
  \@dtl@dogetprops
```

Store the header in \@dtl@toks

```
  \@dtl@colhead={#3}%
```

Reconstruct property list

```
  \edef\@dtl@colnum{\number#2}\relax
  \toks@gconcat@middle@cx{dtlkeys@#1}%
  {\@dtl@before}%
  {%
    \noexpand\db@plist@elt@w% start of block
      \noexpand\db@col@id@w \@dtl@colnum
        \noexpand\db@col@id@end@% index
```

```
        \noexpand\db@key@id@w \@dtl@key\noexpand\db@key@id@end@% key
        \noexpand\db@type@id@w \@dtl@type
          \noexpand\db@type@id@end@% data type
        \noexpand\db@header@id@w \the\@dtl@colhead
          \noexpand\db@header@id@end@% header
        \noexpand\db@col@id@w \@dtl@colnum
          \noexpand\db@col@id@end@% index
      \noexpand\db@plist@elt@end@% end of block
    }%
    {\@dtl@after}%
  }
```

`\DTLnewdbentry`

`\DTLnewdbentry{`⟨*db name*⟩`}{`⟨*id*⟩`}{`⟨*value*⟩`}`.

Adds an entry to the last row (adds new row if database is empty) and updates general column information if necessary. The starred version doesn't check if the database exists.

```
\newcommand{\DTLnewdbentry}{%
   \@ifstar\@sDTLnewdbentry\@DTLnewdbentry
}
```

`\@DTLnewdbentry` Unstarred version of `\DTLnewdbentry`.

```
\newcommand{\@DTLnewdbentry}[3]{%
  \DTLifdbexists{#1}%
    {\@sDTLnewdbentry{#1}{#2}{#3}}%
    {\PackageError{datatool}{Can't add new entry to database `#1':
      database doesn't exist}{}}%
}
```

`\@sDTLnewdbentry` Starred version of `\DTLnewdbentry` (doesn't check if the database exists).

```
\newcommand*{\@sDTLnewdbentry}[3]{%
```

Update key list

```
  \@dtl@updatekeys{#1}{#2}{#3}%
```

Get the column index

```
  \expandafter\dtlcolumnnum\expandafter
    =\dtl@columnindex{#1}{#2}\relax
```

Get the current row:

```
  \edef\dtl@dogetrow{\noexpand\dtlgetrow{#1}%
    {\number\csname dtlrows@#1\endcsname}}%
  \dtl@dogetrow
```

Check if this row already has an entry for the given column.

```
  \edef\dtl@dogetentry{\noexpand\dtlgetentryfromcurrentrow
     {\noexpand\dtl@entry}{\number\dtlcolumnnum}%
  }%
  \dtl@dogetentry
  \ifx\dtl@entry\dtlnovalue
```

Store the value of this entry in `\@dtl@toks`

```
  \@dtl@toks{#3}%
```

There are no entries in this row for the given column. Add this entry.

```
      \toks@gconcat@middle@cx{dtldb@#1}%
      {\dtlbeforerow}%
      {%
```

Start of this row:

```
        \noexpand\db@row@elt@w%
```

Row ID:

```
        \noexpand\db@row@id@w \number\csname dtlrows@#1\endcsname
          \noexpand\db@row@id@end@%
```

Current row so far

```
        \the\dtlcurrentrow
```

New column: Column ID

```
        \noexpand\db@col@id@w \number\dtlcolumnnum
          \noexpand\db@col@id@end@%
```

Value:

```
            \noexpand\db@col@elt@w \the\@dtl@toks
              \noexpand\db@col@elt@end@%
```

Column ID:

```
        \noexpand\db@col@id@w \number\dtlcolumnnum
          \noexpand\db@col@id@end@%
```

Row ID:

```
        \noexpand\db@row@id@w \number\csname dtlrows@#1\endcsname
          \noexpand\db@row@id@end@%
```

End of this row

```
        \noexpand\db@row@elt@end@%
      }%
```

Rest (this should be empty)

```
      {\dtlafterrow}%
```

Print information message if in verbose mode.

```
      \dtl@message{Added #2\space -> #3\space to database ‘#1’}%
    \else
```

There's already an entry for the given column in this row

```
      \PackageError{datatool}{Can’t add entry with ID ‘#2’ to
        current row of database ‘#1’}{There is already an entry with
        this ID in the current row}%
    \fi
  }
```

`\DTLifdbexists` `\DTLifdbexists{⟨db name⟩}{⟨true part⟩}{⟨false part⟩}`

Checks if a data base with the given name exists.

```
\newcommand{\DTLifdbexists}[3]{%
  \@ifundefined{dtldb@#1}{#3}{#2}}
```

10.6 Accessing Data

\@dtl@assign

\@dtl@assign{⟨*list*⟩}{⟨*db*⟩}

Assigns commands according to the given keys. The current row must be stored in \dtlcurrentrow.

```
\newcommand*{\@dtl@assign}[2]{%
  \@dtl@assigncmd#1,\@nil\@@{#2}%
}
```

\@dtl@assigncmd

\@dtl@assigncmd⟨*cmd*⟩=⟨*id*⟩\@nil

```
\def\@dtl@assigncmd#1=#2,#3\@@#4{%
```

get entry for ID given by #2 and store in #1

```
  \@sDTLifhaskey{#4}{#2}%
  {%
    \edef\@dtl@dogetentry{%
      \noexpand\dtlgetentryfromcurrentrow
        {\noexpand#1}{\dtl@columnindex{#4}{#2}}}%
    \@dtl@dogetentry
```

Set to null if required

```
    \ifx#1\dtlnovalue
      \@@dtl@setnull{#1}{#2}%
    \fi
```

Make it global

```
    \global\let#1=#1\relax
  }%
  {%
    \PackageError{datatool}{Can't assign \string#1\space: there
     is no key `#2' in data base `#4'}{}%
```

Set to null

```
    \global\let#1\DTLstringnull
  }%
```

Recurse?

```
  \def\dtl@tmp{#3}%
  \ifx\@nnil\dtl@tmp
    \let\@dtl@next\@dtl@assigncmdnoop
  \else
    \let\@dtl@next\@dtl@assigncmd
  \fi
  \@dtl@next#3\@@{#4}%
}
```

\@dtl@assigncmdnoop

End loop

```
\def\@dtl@assigncmdnoop#1\@@#2{}
```

`\@dtl@setnull` `\@dtl@setnull{⟨cmd⟩}{⟨id⟩}` sets ⟨*cmd*⟩ to either `\DTLstringnull` or `\DTLnumbernull` depending on the data type for ⟨*id*⟩. (Database name should be stored in `\@dtl@dbname` prior to use.)

```
\newcommand*{\@dtl@setnull}[2]{%
```

Check if database given by `\@dtl@dbname` has the required key.

```
  \@sDTLifhaskey{\@dtl@dbname}{#2}%
  {%
```

Set to null

```
    \@@dtl@setnull{#1}{#2}%
  }%
  {%
```

Key not defined in database `\@dtl@dbname`.

```
    \global\let#1=\DTLstringnull
  }%
}
```

`\@@dtl@setnull` As above, but doesn't check if key exists

```
\newcommand*{\@@dtl@setnull}[2]{%
```

Get the data type associated with this key and store in `\@dtl@type`.

```
  \@sdtlgetdatatype{\@dtl@type}{\@dtl@dbname}{#2}%
```

Check data type.

```
  \ifnum0\@dtl@type=0\relax
```

Data type is ⟨*empty*⟩ or 0, so set to string null.

```
    \global\let#1=\DTLstringnull
  \else
```

Data type is numerical, so set to number null.

```
    \global\let#1=\DTLnumbernull
  \fi
}
```

`\DTLstringnull` String null value:

```
\newcommand*{\DTLstringnull}{NULL}
```

`\DTLnumbernull` Number null value:

```
\newcommand*{\DTLnumbernull}{0}
```

`\DTLifnull` `\DTLifnull{⟨value⟩}{⟨true part⟩}{⟨false part⟩}`

Checks if ⟨*value*⟩ is null (either `\DTLstringnull` or `\DTLnumbernull`) if true, does ⟨*true part*⟩ otherwise does ⟨*false part*⟩.

```
\newcommand*{\DTLifnull}[3]{%
  \ifx\DTLstringnull#1\relax
    #2%
  \else
    \ifx\DTLnumbernull#1\relax
      #2%
    \else
```

```
      #3%
    \fi
  \fi
}
```

\@dtlnovalue

```
\def\@dtlnovalue{Undefined Value}
```

\dtlnovalue

```
\def\dtlnovalue{\@dtlnovalue}
```

\DTLgetkeydata

\DTLgetkeydata{⟨*key*⟩}{⟨*db*⟩}{⟨*col cs*⟩}{⟨*type cs*⟩}{⟨*header cs*⟩}

Gets data for given key in database ⟨*db*⟩: the column index is stored in ⟨*col cs*⟩ and data type is stored in ⟨*type cs*⟩. The unstarred version checks for the existance of the database and key, the starred version doesn't.

```
\newcommand*{\DTLgetkeydata}{%
  \@ifstar\@sdtlgetkeydata\@dtlgetkeydata
}
```

\@dtlgetkeydata Unstarred version of \DTLgetkeydata

```
\newcommand*{\@dtlgetkeydata}[5]{%
```

Check if the database exists.

```
  \DTLifdbexists{#2}%
  {%
```

Check if the given key exists in the database.

```
    \@sDTLifhaskey{#2}{#1}%
    {%
```

Get the data.

```
      \@sdtlgetkeydata{#1}{#2}{#3}{#4}{#5}%
    }%
    {%
```

Key not defined in the given database.

```
      \PackageError{datatool}{Key ‘#1’ not defined in database
        ‘#2’}{}%
    }%
  }%
  {%
```

Database not defined.

```
    \PackageError{datatool}{Database ‘#2’ doesn't exist}{}%
  }%
}
```

\@sdtlgetkeydata \@sdtlgetkeydata{⟨*key*⟩}{⟨*db*⟩}{⟨*col cs*⟩}{⟨*type cs*⟩}{⟨*header cs*⟩} Starred verison of \DTLgetkeydata.

```
\newcommand*{\@sdtlgetkeydata}[5]{%
  \@sdtl@getcolumnindex{#3}{#2}{#1}%
  \edef\@dtl@dogetkeydata{\noexpand\@dtl@getprops
```

```
    {\noexpand\@dtl@key}{\noexpand#4}{\noexpand\@dtl@colhead}%
    {\noexpand\@dtl@before}{\noexpand\@dtl@after}%
    {\expandafter\the\csname dtlkeys@#2\endcsname}%
    {#3}}%
  \@dtl@dogetkeydata
  \edef#5{\the\@dtl@toks}%
}
```

`\dtl@gathervalues`

`\dtl@gathervalues[`⟨*label*⟩`]{`⟨*db name*⟩`}{`⟨*row toks*⟩`}`

Stores each element of ⟨*row*⟩ in ⟨*db name*⟩ into the command `\@dtl@`⟨*label*⟩`@`⟨*key*⟩, where ⟨*key*⟩ is the key for that element, and ⟨*label*⟩ defaults to `key`.

```
\newcommand{\dtl@gathervalues}[3][key]{%
  \dtlforeachkey(\@dtl@key,\@dtl@col,\@dtl@type,\@dtl@head)\in{#2}\do
  {%
    \dtlgetentryfromrow{\@dtl@tmp}{\@dtl@col}{\dtlcurrentrow}%
    \ifx\@dtl@tmp\dtlnovalue
      \@dtl@setnull{\@dtl@tmp}{\@dtl@key}%
    \fi
    \expandafter\let\csname @dtl@#1@\@dtl@key\endcsname\@dtl@tmp
  }%
}
```

`\dtlcurrentrow` Define token register to store current row.

```
\newtoks\dtlcurrentrow
```

`\dtlbeforerow` Define token register to store everything before the current row.

```
\newtoks\dtlbeforerow
```

`\dtlafterrow` Define token register to store everything after the current row.

```
\newtoks\dtlafterrow
```

`\dtlgetrow`

`\dtlgetrow{`⟨*db*⟩`}{`⟨*row idx*⟩`}`

Gets row with index ⟨*row idx*⟩ from database named ⟨*db*⟩ and stores the row in `\dtlcurrentrow`, the preceding rows in `\dtlbeforerow` and the following rows in `\dtlafterrow`. This assumes that the given row exists.

```
\newcommand*{\dtlgetrow}[2]{%
  \expandafter\toks@\expandafter=\csname dtldb@#1\endcsname
  \edef\@dtl@dogetrow{\noexpand\@dtlgetrow{\the\toks@}{\number#2}}%
  \@dtl@dogetrow
}
```

`\@dtlgetrow`

`\@dtlgetrow{`⟨*data specs*⟩`}{`⟨*row idx*⟩`}` Gets the row specs from ⟨*data specs*⟩ for row with index ⟨*row idx*⟩ which must be fully expanded.

```
\newcommand*{\@dtlgetrow}[2]{%
  \def\@dtl@getrow##1% before stuff
```

```
    \db@row@elt@w% start of the row
      \db@row@id@w #2\db@row@id@end@% row id
        ##2%
      \db@row@id@w #2\db@row@id@end@% row id
    \db@row@elt@end@% end of the row
        ##3% after stuff
    \q@nil{\dtlbeforerow={##1}\dtlcurrentrow={##2}\dtlafterrow={##3}}%
  \@dtl@getrow#1\q@nil
}
```

\dtlgetentryfromcurrentrow

\dtlgetentryfromcurrentrow{⟨*cs*⟩}{⟨*col num*⟩}

Gets value for column ⟨*col num*⟩ from \dtlcurrentrow and stores in ⟨*cs*⟩. If not found, ⟨*cs*⟩ is set to \dtlnovalue.

```
\newcommand*{\dtlgetentryfromcurrentrow}[2]{%
  \dtlgetentryfromrow{#1}{#2}{\dtlcurrentrow}%
}
```

\dtlgetentryfromrow

\dtlgetentryfromrow{⟨*cs*⟩}{⟨*col num*⟩}{⟨*row toks*⟩}

```
\newcommand*{\dtlgetentryfromrow}[3]{%
  \edef\@dtl@do@getentry{\noexpand\dtl@getentryfromrow
    {\noexpand#1}{\number#2}{\the#3}}%
  \@dtl@do@getentry
}
```

\dtl@getentryfromrow

\dtl@getentryfromrow{⟨*cs*⟩}{⟨*col num*⟩}{⟨*row specs*⟩}

```
\newcommand*{\dtl@getentryfromrow}[3]{%
  \def\dtl@dogetentry##1% before stuff
    \db@col@id@w #2\db@col@id@end@% Column id
      \db@col@elt@w ##2\db@col@elt@end@% Value
    \db@col@id@w #2\db@col@id@end@% Column id
    ##3% Remaining stuff
    \q@nil{\def#1{##2}}%
  \dtl@dogetentry#3%
    \db@col@id@w #2\db@col@id@end@%
      \db@col@elt@w \@dtlnovalue\db@col@elt@end@%
    \db@col@id@w #2\db@col@id@end@%
    \q@nil
}
```

\DTLgetvalue

\DTLgetvalue{⟨*cs*⟩}{⟨*db*⟩}{⟨*r*⟩}{⟨*c*⟩}

Gets the element in row ⟨*r*⟩, column ⟨*c*⟩ from database ⟨*db*⟩ and stores in ⟨*cs*⟩.

```
\newcommand*{\DTLgetvalue}[4]{%
  \edef\dtl@dogetvalue{\noexpand\dtl@getvalue{\noexpand#1}{#2}%
```

```
    {\number#3}{\number#4}}%
  \dtl@dogetvalue
}
```

\dtl@getvalue

```
\newcommand*{\dtl@getvalue}[4]{%
  \def\@dtl@getvalue ##1% stuff before row <r>
     \db@row@id@w #3\db@row@id@end@% row <r> id
        ##2% stuff in row <r> before column <c>
     \db@col@id@w #4\db@col@id@end@% column <c> id
       \db@col@elt@w ##3\db@col@elt@end@% value
     ##4% stuff after value
     \q@nil{\def#1{##3}}%
  \toks@=\csname dtldb@#2\endcsname
  \expandafter\@dtl@getvalue\the\toks@% contents of data base
     \db@row@id@w #3\db@row@id@end@%
       \db@col@id@w #4\db@col@id@end@%
         \db@col@elt@w \@dtlnovalue\db@col@elt@end@% undefined value
     \q@nil
  \ifx#1\dtlnovalue
    \PackageError{datatool}{There is no element at (row=#3,
      column=#4) in database `#2'}{}%
  \fi
}
```

\DTLgetlocation

\DTLgetlocation{⟨*row cs*⟩}{⟨*column cs*⟩}{⟨*database*⟩} {⟨*value*⟩}

Assigns ⟨*row cs*⟩ and ⟨*column cs*⟩ to the indices of the first entry in ⟨*database*⟩ that matches ⟨*value*⟩.

```
\newcommand*{\DTLgetlocation}[4]{%
  \def\@dtl@getlocation##1% stuff before value
    \db@col@elt@w #4\db@col@elt@end@% value
    \db@col@id@w ##2\db@col@id@end@% column id
    ##3% stuff after this column
    \db@row@id@w ##4\db@row@id@end@% row id
    ##5% stuff after row
    \q@nil{\def#1{##4}\def#2{##2}}%
  \toks@=\csname dtldb@#3\endcsname
  \expandafter\@dtl@getlocation\the\toks@% contents of data base
    \db@col@elt@w #4\db@col@elt@end@% value
    \db@col@id@w \@dtlnovalue\db@col@id@end@% undefined column id
    \db@row@id@w \@dtlnovalue\db@row@id@end@% undefined row id
    \q@nil
  \ifx#1\dtlnovalue
    \PackageError{datatool}{There is no element `#4' in
      database `#3'}{}%
  \fi
}
```

10.7 Iterating Through Databases

\dtlbreak Break out of loop at the end of current iteration.

```
\newcommand*{\dtlbreak}{%
  \PackageError{datatool}{Can't break out of anything}{}%
}
```

\dtlforint

\dtlforint⟨*ct*⟩=⟨*start*⟩\to⟨*end*⟩\step ⟨*inc*⟩\do{⟨*body*⟩}

⟨*ct*⟩ is a count register, ⟨*start*⟩, ⟨*end*⟩ and ⟨*inc*⟩ are integers. Group if nested or use \dtlgforint. An infinite loop may result if ⟨*inc*⟩= 0 and ⟨*start*⟩ ≤ ⟨*end*⟩ and \dtlbreak isn't used.

```
\long\def\dtlforint#1=#2\to#3\step#4\do#5{%
```

Make a copy of old version of break function

```
  \let\@dtl@orgbreak\dtlbreak
  \def\@dtl@endloophook{}%
```

Setup break function for the loop (sets ⟨*ct*⟩ to ⟨*end*⟩ at the end of the current iteration).

```
  \def\dtlbreak{\def\@dtl@endloophook{#1=#3}}%
```

Initialise ⟨*ct*⟩

```
  #1=#2\relax
```

Check if the steps are positive or negative.

```
  \ifnum#4<0\relax
```

Counting down

```
    \whiledo{\(#1>#3\)\TE@or\(#1=#3\)}%
    {%
      #5%
      \@dtl@endloophook
      \advance#1 by #4\relax
    }%
  \else
```

Counting up

```
    \whiledo{\(#1<#3\)\TE@or\(#1=#3\)}%
    {%
      #5%
      \@dtl@endloophook
      \advance#1 by #4\relax
    }%
  \fi
```

Restore break function.

```
  \let\dtlbreak\@dtl@orgbreak
}
```

\@dtl@foreach@level

Count register to keep track of global nested loops.

```
\newcount\@dtl@foreach@level
```

\dtlgforint

\dtlgforint⟨*ct*⟩=⟨*start*⟩\to⟨*end*⟩\step ⟨*inc*⟩\do{⟨*body*⟩}

⟨*ct*⟩ is a count register, ⟨*start*⟩, ⟨*end*⟩ and ⟨*inc*⟩ are integers. An infinite loop may result if ⟨*inc*⟩=0 and ⟨*start*⟩ ≤ ⟨*end*⟩ and `\dtlbreak` isn't used.

```
\long\def\dtlgforint#1=#2\to#3\step#4\do#5{%
```

Initialise

```
  \global#1=#2\relax
```

Increment level counter to allow for nested loops

```
  \global\advance\@dtl@foreach@level by 1\relax
```

Set up end loop hook

```
  \expandafter\global\expandafter
    \let\csname @dtl@endhook@\the\@dtl@foreach@level\endcsname
    \relax
```

Set up the break function: Copy current definition

```
  \expandafter\global\expandafter
    \let\csname @dtl@break@\the\@dtl@foreach@level\endcsname
    \dtlbreak
```

Set up definition for this level (sets <ct> to <end> at the end of the current iteration).

```
  \gdef\dtlbreak{\expandafter
     \gdef\csname @dtl@endhook@\the\@dtl@foreach@level\endcsname{%
       #1=#3}}%
```

check the direction

```
  \ifnum#4<0\relax
```

Counting down

```
    \whiledo{\(#1>#3\)\TE@or\(#1=#3\)}%
    {%
      #5%
      \csname @dtl@endhook@\the\@dtl@foreach@level\endcsname
      \global\advance#1 by #4\relax
    }%
  \else
```

Counting up (or 0 increments)

```
    \whiledo{\(#1<#3\)\TE@or\(#1=#3\)}%
    {%
      #5%
      \csname @dtl@endhook@\the\@dtl@foreach@level\endcsname
      \global\advance#1 by #4\relax
    }%
  \fi
```

Restore break function

```
  \expandafter\global\expandafter\let\expandafter\dtlbreak
    \csname @dtl@break@\the\@dtl@foreach@level\endcsname
```

Decrement level counter

```
  \global\advance\@dtl@foreach@level by -1\relax
}
```

`\@dtlforeachrow`

`\@dtlforeachrow`(⟨*idx cs*⟩,⟨*row cs*⟩)`\in{`⟨*db*⟩`} \do{`⟨*body*⟩`}`

Iterates through each row in database. Assigns the current row index to ⟨*idx cs*⟩ and the row specs to ⟨*row cs*⟩

```
\long\def\@dtlforeachrow(#1,#2)\in#3\do#4{%
  \edef\dtl@tmp{\expandafter\the\csname dtldb@#3\endcsname}%
  \expandafter\@dtl@foreachrow\dtl@tmp
    \db@row@elt@w%
    \db@row@id@w \@nil\db@row@id@end@%
    \db@row@id@w \@nil\db@row@id@end@%
    \db@row@elt@end@%
    \@@{#1}{#2}{#4}\q@nil
}
```

`\@dtl@foreachrow`

```
\long\def\@dtl@foreachrow\db@row@elt@w%
\db@row@id@w #1\db@row@id@end@%
#2\db@row@id@w #3\db@row@id@end@%
\db@row@elt@end@#4\@@#5#6#7\q@nil{%
```

Define control sequence given by `#5`

```
\gdef#5{#1}%
```

Hide the loop body in a macro

```
\gdef\@dtl@loopbody{#7}%
```

Increment level counter to allow for nested loops

```
\global\advance\@dtl@foreach@level by 1\relax
```

Check if we have reached the end of the loop

```
\ifx#5\@nnil
  \expandafter\global\expandafter
    \let\csname @dtl@foreachnext\the\@dtl@foreach@level\endcsname
      =\@dtl@foreachnoop
\else
  \gdef#6{#2}%
```

Set up the break function: Make a copy of current break function

```
  \expandafter\let
    \csname @dtl@break@\the\@dtl@foreach@level\endcsname
    \dtlbreak
```

Setup break function for this level

```
  \gdef\dtlbreak{\expandafter\global\expandafter
    \let\csname @dtl@foreachnext\the\@dtl@foreach@level\endcsname
      =\@dtl@foreachnoop}%
```

Initialise

```
  \expandafter\global\expandafter
    \let\csname @dtl@foreachnext\the\@dtl@foreach@level\endcsname
      =\@dtl@foreachrow
```

Do body of loop

```
  \@dtl@loopbody
```

Restore break function

```
  \expandafter\let\expandafter\dtlbreak
    \csname @dtl@break@\the\@dtl@foreach@level\endcsname
\fi
```

Set up what to do next.

```
    \expandafter\let\expandafter\@dtl@foreachnext
      \csname @dtl@foreachnext\the\@dtl@foreach@level\endcsname
```

Decrement level counter.

```
    \global\advance\@dtl@foreach@level by -1\relax
```

Repeat loop if necessary.

```
    \@dtl@foreachnext#4\@@{#5}{#6}{#7}\q@nil
  }
```

`\@dtl@foreachnoop`

```
  \long\def\@dtl@foreachnoop#1\@@#2\q@nil{}
```

`\dtlforeachkey` `\dtlforeachkey`(⟨*key cs*⟩,⟨*col cs*⟩,⟨*type cs*⟩,⟨*header cs*⟩) `\in{`⟨*db*⟩`}\do{`⟨*body*⟩`}`

Iterates through all the keys in database ⟨*db*⟩. In each iteration, ⟨*key cs*⟩ stores the key, ⟨*col cs*⟩ stores the column index and ⟨*type cs*⟩ stores the data type.

```
  \long\def\dtlforeachkey(#1,#2,#3,#4)\in#5\do#6{%
    \gdef\@dtl@loopbody{#6}%
    \edef\@dtl@keys{\expandafter\the\csname dtlkeys@#5\endcsname}%
    \expandafter\@dtl@foreachkey\@dtl@keys
      \db@plist@elt@w%
      \db@col@id@w -1\db@col@id@end@%
      \db@key@id@w \db@key@id@end@%
      \db@type@id@w \db@type@id@end@%
      \db@header@id@w \db@header@id@end@%
      \db@col@id@w -1\db@col@id@end@%
      \db@plist@elt@end@%
      \@@{\@dtl@updatefkcs{#1}{#2}{#3}{#4}}\q@nil
  }
```

`\@dtl@updatefkcs`

```
  \newcommand*{\@dtl@updatefkcs}[8]{%
    \gdef#1{#5}%
    \gdef#2{#6}%
    \gdef#3{#7}%
    \gdef#4{#8}%
  }
```

`\@dtl@foreachkey` Sets everything globally in case it occurs in a tabular environment Loop body needs to be stored in `\@dtl@loopbody`. `#7` indicates an update macro.

```
  \long\def\@dtl@foreachkey\db@plist@elt@w%
  \db@col@id@w #1\db@col@id@end@%
  \db@key@id@w #2\db@key@id@end@%
  \db@type@id@w #3\db@type@id@end@%
  \db@header@id@w #4\db@header@id@end@%
  \db@col@id@w #5\db@col@id@end@%
  \db@plist@elt@end@#6\@@#7\q@nil{%
    \ifnum#1=-1\relax
```

Terminate loop

```
      \let\@dtl@foreachnext\@dtl@foreachnoop
    \else
```

Set up loop variables

```
      #7{#2}{#1}{#3}{#4}%
```

Increment level counter to allow for nested loops

```
      \global\advance\@dtl@foreach@level by 1\relax
```

Set up the break function

```
      \expandafter\let
        \csname @dtl@break@\the\@dtl@foreach@level\endcsname
        \dtlbreak
      \gdef\dtlbreak{\expandafter\global\expandafter
        \let\csname @dtl@foreachnext\the\@dtl@foreach@level\endcsname
          =\@dtl@foreachnoop}%
```

Initialise

```
      \expandafter\global\expandafter
        \let\csname @dtl@foreachnext\the\@dtl@foreach@level\endcsname
          =\@dtl@foreachkey
```

Do body of loop

```
      \@dtl@loopbody
```

Set up what to do next

```
      \expandafter\let\expandafter\@dtl@foreachnext
        \csname @dtl@foreachnext\the\@dtl@foreach@level\endcsname
```

Restore break function

```
      \expandafter\let\expandafter\dtlbreak
        \csname @dtl@break@\the\@dtl@foreach@level\endcsname
```

Decrement level counter

```
      \global\advance\@dtl@foreach@level by -1\relax
    \fi
```

Recurse if necessary

```
    \@dtl@foreachnext#6\@@{#7}\q@nil
  }
```

`\dtlforcolumn`

`\dtlforcolumn{⟨cs⟩}{⟨db⟩}{⟨key⟩}{⟨body⟩}`

Iterates through column given by ⟨*key*⟩ in database ⟨*db*⟩. ⟨*cs*⟩ is assign to the element of the column in the current iteration. Starred version doesn't check if data base exists

```
  \newcommand*{\dtlforcolumn}{\@ifstar\@sdtlforcolumn\@dtlforcolumn}
```

`\@dtlforcolumn`

```
  \newcommand{\@dtlforcolumn}[4]{%
```

Check if data base exists

```
    \DTLifdbexists{#2}%
    {%
      \@DTLifhaskey{#2}{#3}%
```

```
      {%
        \@sdtlforcolumn{#1}{#2}{#3}{#4}%
      }%
```

key not in data base

```
      {%
        \PackageError{datatool}{Database ‘#2’ doesn’t contain
          key ‘#3’}{}%
      }%
    }%
  %
    {%
      \PackageError{datatool}{Database ‘#2’ doesn’t exist}{}%
    }%
  }
```

\@sdtlforcolumn

```
  \newcommand{\@sdtlforcolumn}[4]{%
      \toks@{#4}%
      \edef\@dtl@doforcol{\noexpand\dtl@forcolumn{\noexpand#1}%
        {\expandafter\the\csname dtldb@#2\endcsname}%
        {\dtl@columnindex{#2}{#3}}{\the\toks@}%
      }%
      \@dtl@doforcol%
  }
  %    end{macrocode}
  %\end{macro}
  %
  %\begin{macro}{\dtlforcolumnidx}
  %\begin{definition}
  %\cs{dtlforcolumnidx}\marg{cs}\marg{db}\marg{col num}\marg{body}
  %\end{definition}
  % Iterates through the column with index <col num> in database <db>.
  % Starred version doesn’t check if database exists.
  %\changes{2.0}{2009 February 27}{new}
  %   \begin{macrocode}
  \newcommand*{\dtlforcolumnidx}{%
    \@ifstar\@sdtlforcolumnidx\@dtlforcolumnidx
  }
  %   \end{macrocode}
  %\end{macro}
  %
  %\begin{macro}{\@dtlforcolumnidx}
  %    \begin{macrocode}
  \newcommand{\@dtlforcolumnidx}[4]{%
    \DTLifdbexists{#2}%
    {%
      \expandafter\ifnum\csname dtlcols@#2\endcsname<#3\relax
        \PackageError{datatool}{Column index \number#3\space out of
          bounds for database ‘#2’}{Database ‘#2’ only has
          \expandafter\number\csname dtlcols@#2\endcsname\space
          columns}%
      \else
        \ifnum#3<1\relax
```

```
          \PackageError{datatool}{Column index \number#3\space out of
           bounds for database ‘#2’}{Indices start from 1}%
        \else
          \@sdtlforcolumnidx{#1}{#2}{#3}{#4}%
        \fi
      \fi
    }%
```

data base doesn't exist

```
    {%
      \PackageError{datatool}{Database ‘#2’ doesn't exist}{}%
    }%
  }
```

\@sdtlforcolumnidx

```
  \newcommand{\@sdtlforcolumnidx}[4]{%
      \toks@{#4}%
      \edef\@dtl@doforcol{\noexpand\dtl@forcolumn{\noexpand#1}%
        {\expandafter\the\csname dtldb@#2\endcsname}%
        {\number#3}{\the\toks@}%
      }%
      \@dtl@doforcol
  }
  %   \end{macrocode}
  %\end{macro}
  %
  %\begin{macro}{\dtl@forcolumn}
  %\begin{definition}
  %\cs{dtl@forcolumn}\marg{cs}\marg{db specs}\marg{col num}\marg{body}
  %\end{definition}
  % \meta{col num} needs to be fully expanded
  %    \begin{macrocode}
  \newcommand{\dtl@forcolumn}[4]{%
```

make a copy of break function

```
    \let\@dtl@oldbreak\dtlbreak
```

set up break function

```
    \def\dtlbreak{\let\@dtl@forcolnext=\@dtl@forcolnoop}%
```

define loop macro for this column

```
    \def\@dtl@forcolumn##1% before stuff
      \db@col@id@w #3\db@col@id@end@% column index
        \db@col@elt@w ##2\db@col@elt@end@% entry
      \db@col@id@w #3\db@col@id@end@% column index
      ##3% after stuff
      \q@nil{%
        \def#1{##2}% assign value to <cs>
```

check if end of loop

```
        \ifx#1\@nnil
          \let\@dtl@forcolnext=\@dtl@forcolnoop
        \else
```

do body of loop

```
          #4%
```

```
          \let\@dtl@forcolnext=\@dtl@forcolumn
        \fi
```

repeat if necessary

```
        \@dtl@forcolnext##3\q@nil
      }%
```

do loop

```
    \@dtl@forcolumn#2%
     \db@col@id@w #3\db@col@id@end@%
       \db@col@elt@w \@nil\db@col@elt@end@%
     \db@col@id@w #3\db@col@id@end@\q@nil
```

restore break function

```
    \let\dtlbreak\@dtl@oldbreak
  }
```

`\@dtl@forcolnoop`

```
  \def\@dtl@forcolnoop#1\q@nil{}
```

`\dtlforeachlevel` `\DTLforeach` can only be nested up to three levels. `\dtlforeachlevel` keeps track of the current level.

```
  \newcount\dtlforeachlevel
```

The counter DTLrow⟨n⟩ keeps track of each row of data during the ⟨n⟩ nested `\DTLforeach`. It is only incremented in the conditions (given by the optional argument) are met.

```
  \newcounter{DTLrowi}
  \newcounter{DTLrowii}
  \newcounter{DTLrowiii}
```

Keep hyperref happy

```
  \newcounter{DTLrow}
  \def\theHDTLrow{\arabic{DTLrow}}
  \def\theHDTLrowi{\theHDTLrow.\arabic{DTLrowi}}
  \def\theHDTLrowii{\theHDTLrowi.\arabic{DTLrowii}}
  \def\theHDTLrowiii{\theHDTLrowii.\arabic{DTLrowiii}}

  \newcount\dtl@rowi
  \newcount\dtl@rowii
  \newcount\dtl@rowiii

  \newtoks\@dtl@curi
  \newtoks\@dtl@previ
  \newtoks\@dtl@nexti
  \newtoks\@dtl@curii
  \newtoks\@dtl@previi
  \newtoks\@dtl@nextii
  \newtoks\@dtl@curiii
  \newtoks\@dtl@previii
  \newtoks\@dtl@nextiii
```

`\DTLsaverowcount`

`\DTLsavelastrowcount{`⟨cmd⟩`}`

Stores the maximum row count for the last `\DTLforeach`.

```
\newcommand*{\DTLsavelastrowcount}[1]{%
\ifnum\dtlforeachlevel>2\relax
  \def#1{0}%
\else
  \ifnum\dtlforeachlevel<0\relax
    \def#1{0}%
  \else
    \@dtl@tmpcount=\dtlforeachlevel
    \advance\@dtl@tmpcount by 1\relax
    \edef#1{\expandafter\number
      \csname c@DTLrow\romannumeral\@dtl@tmpcount\endcsname}%
  \fi
\fi}
```

\DTLforeach

\DTLforeach[⟨*conditions*⟩]{⟨*db name*⟩}{⟨*values*⟩}{⟨*text*⟩}

For each row of data in the database given by ⟨*db name*⟩, do ⟨*text*⟩, if the specified conditions are satisfied. The argument {⟨*values*⟩} is a comma separated list of ⟨*cmd*⟩=⟨*key*⟩ pairs. At the start of each row, each of the commands in this list are set to the value of the entry with the corresponding key ⟨*key*⟩. (\gdef is used to ensure \DTLforeach works in a tabular environment.) The database may be edited in the unstarred version, in the starred version the database is read only.

```
\newcommand*{\DTLforeach}{\@ifstar\@sDTLforeach\@DTLforeach}
```

\@DTLforeach

\@DTLforeach is the unstarred version of \DTLforeach. The database is reconstructed to allow for rows to be edited. Use the starred version for faster access.

```
\newcommand{\@DTLforeach}[4][\boolean{true}]{%
```

Check database exists

```
  \DTLifdbexists{#2}%
  {%
```

Keep hyperref happy

```
    \refstepcounter{DTLrow}%
```

Make it global (so that it works in tabular environment)

```
    \global\c@DTLrow=\c@DTLrow\relax
```

Store database name

```
    \gdef\@dtl@dbname{#2}%
```

Increment level and check not exceeded 3

```
    \global\advance\dtlforeachlevel by 1\relax
    \ifnum\dtlforeachlevel>3\relax
      \PackageError{datatool}{\string\DTLforeach\space nested too
        deeply}{Only 3 levels are allowed}%
    \else
       \@DTLifdbempty{#2}%
```

Do nothing if database is empty

```
       {}%
       {%
```

Set level dependent information (needs to be global to ensure it works in the tabular environment). Row counter:

```
\expandafter\global
  \csname c@DTLrow\romannumeral\dtlforeachlevel\endcsname
    = 0\relax
```

Store previous value of \DTLiffirstrow

```
\expandafter\global\expandafter\let%
  \csname @dtl@iffirstrow\the\dtlforeachlevel\endcsname
  \DTLiffirstrow
```

Define current \DTLiffirstrow

```
\gdef\DTLiffirstrow##1##2{%
  \expandafter\ifnum
   \csname c@DTLrow\romannumeral\dtlforeachlevel\endcsname
   =1\relax
    ##1%
  \else
    ##2%
  \fi}%
```

Store previous value of \DTLiflastrow

```
\expandafter\global\expandafter\let%
  \csname @dtl@iflastrow\the\dtlforeachlevel\endcsname
  \DTLiflastrow
```

Define current \DTLiflastrow

```
\gdef\DTLiflastrow##1##2{%
  \expandafter\ifnum
   \csname c@DTLrow\romannumeral\dtlforeachlevel\endcsname
   =\csname dtlcols@#2\endcsname\relax
    ##1%
  \else
    ##2%
  \fi}%
```

Store previous value of \DTLifoddrow

```
\expandafter\global\expandafter\let%
  \csname @dtl@ifoddrow\the\dtlforeachlevel\endcsname
  \DTLifoddrow
```

Define current \DTLifoddrow

```
\gdef\DTLifoddrow##1##2{%
  \expandafter\ifodd
   \csname c@DTLrow\romannumeral\dtlforeachlevel\endcsname
    ##1%
  \else
    ##2%
  \fi}%
```

Store data base name for current level

```
\expandafter\global\expandafter\let
  \csname @dtl@dbname@\romannumeral\dtlforeachlevel\endcsname
    =\@dtl@dbname
```

Mark it as not read only

```
\expandafter\global\expandafter\let
```

```
\csname @dtl@ro@\romannumeral\dtlforeachlevel\endcsname
  = 0\relax
```

Loop through each row. Loop counter given by `\dtl@row@`⟨*level*⟩

```
\dtlgforint
  \csname dtl@row\romannumeral\dtlforeachlevel\endcsname
  =1\to\csname dtlrows@#2\endcsname\step1\do
{%
```

Get current row from the data base

```
  \@dtl@tmpcount=
    \csname dtl@row\romannumeral\dtlforeachlevel\endcsname
  \edef\dtl@dogetrow{\noexpand\dtlgetrow{#2}%
    {\number\@dtl@tmpcount}}%
  \dtl@dogetrow
```

Store the current row for this level

```
  \expandafter\global
    \csname @dtl@cur\romannumeral\dtlforeachlevel\endcsname
      = \dtlcurrentrow
```

Store the previous rows for this level

```
  \expandafter\global
    \csname @dtl@prev\romannumeral\dtlforeachlevel\endcsname
      = \dtlbeforerow
```

Store the subsequent rows for this level

```
  \expandafter\global
    \csname @dtl@next\romannumeral\dtlforeachlevel\endcsname
      = \dtlafterrow
```

Assign commands to the required entries

```
  \ifx\relax#3\relax
  \else
    \@dtl@assign{#3}{#2}%
  \fi
```

Do the main body of text if condition is satisfied

```
  \ifthenelse{#1}%
  {%
```

Increment user row counter

```
    \refstepcounter{DTLrow\romannumeral\dtlforeachlevel}%
    \expandafter\edef\expandafter\DTLcurrentindex%
      \expandafter{%
          \arabic{DTLrow\romannumeral\dtlforeachlevel}}%
    #4%
```

Has this row been marked for deletion?

```
    \edef\@dtl@tmp{\expandafter\the
      \csname @dtl@cur\romannumeral
        \dtlforeachlevel\endcsname}%
    \ifx\@dtl@tmp\@nnil
```

Row needs to be deleted Decrement row indices for rows with a higher index than this one

```
      \expandafter\dtl@decrementrows\expandafter
         {\csname @dtl@prev\romannumeral
```

```
        \dtlforeachlevel\endcsname
      }{\csname dtl@row\romannumeral
        \dtlforeachlevel\endcsname}%
    \expandafter\dtl@decrementrows\expandafter
      {\csname @dtl@next\romannumeral
        \dtlforeachlevel\endcsname
      }{\csname dtl@row\romannumeral
        \dtlforeachlevel\endcsname}%
```

Reconstruct data base without this row

```
    \edef\@dtl@tmp{%
      \expandafter\the
        \csname @dtl@prev\romannumeral
          \dtlforeachlevel\endcsname
      \expandafter\the
        \csname @dtl@next\romannumeral
            \dtlforeachlevel\endcsname
      }%
    \expandafter\global\expandafter
       \csname dtldb@#2\endcsname\expandafter{\@dtl@tmp}%
```

Decrement the row count for this database:

```
    \expandafter\global\expandafter
      \advance\csname dtlrows@#2\endcsname by -1\relax
```

Decrement the counter for this loop

```
    \expandafter\global\expandafter
      \advance\csname dtl@row\romannumeral
         \dtlforeachlevel\endcsname by -1\relax
  \else
```

Reconstruct data base

```
    \@dtl@before=\csname @dtl@prev\romannumeral
      \dtlforeachlevel\endcsname
    \@dtl@after=\csname @dtl@next\romannumeral
      \dtlforeachlevel\endcsname
    \toks@gconcat@middle@cx{dtldb@#2}%
    {\@dtl@before}%
    {%
```

This row

```
      \noexpand\db@row@elt@w%
      \noexpand\db@row@id@w \expandafter\number
        \csname dtl@row\romannumeral
          \dtlforeachlevel\endcsname
      \noexpand\db@row@id@end@%
      \expandafter\the
        \csname @dtl@cur\romannumeral
          \dtlforeachlevel\endcsname
      \noexpand\db@row@id@w \expandafter\number
        \csname dtl@row\romannumeral
          \dtlforeachlevel\endcsname
      \noexpand\db@row@id@end@%
      \noexpand\db@row@elt@end@%
      }%
    {\@dtl@after}%
```

```
            \fi
          }%
```

Condition not met so ignore

```
          {}%
        }%
```

Restore previous value of \DTLiffirstrow

```
        \expandafter\global\expandafter\let\expandafter\DTLiffirstrow
          \csname @dtl@iffirstrow\the\dtlforeachlevel\endcsname
```

Restore previous value of \DTLiflastrow

```
        \expandafter\global\expandafter\let\expandafter\DTLiflastrow
          \csname @dtl@iflastrow\the\dtlforeachlevel\endcsname
```

Restore previous value of \DTLifoddrow

```
        \expandafter\global\expandafter\let\expandafter\DTLifoddrow
          \csname @dtl@ifoddrow\the\dtlforeachlevel\endcsname
      }%
    \fi
```

Decrement level

```
    \global\advance\dtlforeachlevel by -1\relax
  }%
```

else part (data base doesn't exist):

```
  {%
    \PackageError{datatool}{Database `#2' doesn't exist}{}%
  }%
}
```

\@sDTLforeach

\@sDTLforeach is the starred version of \DTLforeach. The database rows can't be edited.

```
\newcommand{\@sDTLforeach}[4][\boolean{true}]{%
```

Check database exists

```
  \DTLifdbexists{#2}%
  {%
```

Keep hyperref happy

```
    \refstepcounter{DTLrow}%
```

Make it global (so that it works in tabular environment)

```
    \global\c@DTLrow=\c@DTLrow
```

Increment level and check not exceeded 3

```
    \global\advance\dtlforeachlevel by 1\relax
    \ifnum\dtlforeachlevel>3\relax
      \PackageError{datatool}{\string\DTLforeach\space nested too
        deeply}{Only 3 levels are allowed}%
    \else
       \@DTLifdbempty{#2}%
```

Do nothing if database is empty

```
       {}%
       {%
```

Set level dependent information (needs to be global to ensure it works in the tabular environment). Row counter:

```
\expandafter\global
  \csname c@DTLrow\romannumeral\dtlforeachlevel\endcsname
    = 0\relax
```

Store previous value of \DTLiffirstrow

```
\expandafter\global\expandafter\let%
  \csname @dtl@iffirstrow\the\dtlforeachlevel\endcsname
  \DTLiffirstrow
```

Define current \DTLiffirstrow

```
\gdef\DTLiffirstrow##1##2{%
  \expandafter\ifnum
   \csname c@DTLrow\romannumeral\dtlforeachlevel\endcsname
   =1\relax
    ##1%
  \else
    ##2%
  \fi}%
```

Store previous value of \DTLiflastrow

```
\expandafter\global\expandafter\let%
  \csname @dtl@iflastrow\the\dtlforeachlevel\endcsname
  \DTLiflastrow
```

Define current \DTLiflastrow

```
\gdef\DTLiflastrow##1##2{%
  \expandafter\ifnum
   \csname c@DTLrow\romannumeral\dtlforeachlevel\endcsname
   =\csname dtlcols@#2\endcsname\relax
    ##1%
  \else
    ##2%
  \fi}%
```

Store previous value of \DTLifoddrow

```
\expandafter\global\expandafter\let%
  \csname @dtl@ifoddrow\the\dtlforeachlevel\endcsname
  \DTLifoddrow
```

Define current \DTLifoddrow

```
\gdef\DTLifoddrow##1##2{%
  \expandafter\ifodd
   \csname c@DTLrow\romannumeral\dtlforeachlevel\endcsname
    ##1%
  \else
    ##2%
  \fi}%
```

Store data base name for current level

```
\expandafter\gdef\csname @dtl@dbname@\romannumeral
  \dtlforeachlevel\endcsname{#2}%
```

Mark it as read only

```
\expandafter\global\expandafter\let
  \csname @dtl@ro@\romannumeral\dtlforeachlevel\endcsname
    = 1\relax
```

Iterate through each row.

```
\@dtlforeachrow(\dtl@thisidx,\dtl@thisrow)\in{#2}\do%
{%
```

Assign row number (not sure if this is needed here)

```
  \csname dtl@row\romannumeral\dtlforeachlevel\endcsname
    = \dtl@thisidx\relax
```

Store the current row specs for this level

```
  \expandafter\global
    \csname @dtl@cur\romannumeral\dtlforeachlevel\endcsname
      = \expandafter{\dtl@thisrow}%
```

Assign commands to the required entries

```
  \ifx\relax#3\relax
  \else
```

Need to set `\dtlcurrentrow` for `\@dtl@assign`

```
    \dtlcurrentrow=\expandafter{\dtl@thisrow}%
    \@dtl@assign{#3}{#2}%
  \fi
```

Do the main body of text if condition is satisfied

```
  \ifthenelse{#1}%
  {%
```

Increment user row counter

```
    \refstepcounter{DTLrow\romannumeral\dtlforeachlevel}%
    \expandafter\edef\expandafter\DTLcurrentindex%
      \expandafter{%
          \arabic{DTLrow\romannumeral\dtlforeachlevel}}%
    #4%
  }%
```

Condition not met so ignore

```
  {}%
 }%
```

Restore previous value of `\DTLiffirstrow`

```
\expandafter\global\expandafter\let\expandafter\DTLiffirstrow
  \csname @dtl@iffirstrow\the\dtlforeachlevel\endcsname
```

Restore previous value of `\DTLiflastrow`

```
\expandafter\global\expandafter\let\expandafter\DTLiflastrow
  \csname @dtl@iflastrow\the\dtlforeachlevel\endcsname
```

Restore previous value of `\DTLifoddrow`

```
\expandafter\global\expandafter\let\expandafter\DTLifoddrow
  \csname @dtl@ifoddrow\the\dtlforeachlevel\endcsname
}%
\fi
```

Decrement level

```
  \global\advance\dtlforeachlevel by -1\relax
}%
```

else part (data base doesn't exist):

```
    {%
      \PackageError{datatool}{Database ‘#2’ doesn’t exist}{}%
    }%
  }
```

\@dtlifreadonly

\@dtlifreadonly{⟨*true part*⟩}{⟨*false part*⟩}

Checks if current loop level is read only

```
\newcommand*{\@dtlifreadonly}[2]{%
  \expandafter\ifx
    \csname @dtl@ro@\romannumeral\dtlforeachlevel\endcsname1\relax
```

Read only

```
      #1%
    \else
```

Not read only

```
      #2%
    \fi
  }
```

\DTLappendtorow

\DTLappendtorow{⟨*key*⟩}{⟨*value*⟩}

Appends entry to current row. (The current row is given by \@dtl@cur⟨*n*⟩ where ⟨*n*⟩ is roman numeral value of \dtlforeachlevel. One level expansion is applied to ⟨*value*⟩.

```
\newcommand*{\DTLappendtorow}[2]{%
  \ifnum\dtlforeachlevel=0\relax
    \PackageError{datatool}{\string\DTLappendrow\space can only be
      used inside \string\DTLforeach}{}%
  \else
```

Set \@dtl@thisdb to the current database name:

```
      \expandafter\let\expandafter\@dtl@thisdb
        \csname @dtl@dbname@\romannumeral\dtlforeachlevel\endcsname
```

Check this isn't in \DTLforeach*

```
      \@dtlifreadonly
      {%
        \PackageError{datatool}{\string\DTLappendtorow\space can’t
         be used inside \DTLforeach*}{The starred version of
         \string\DTLforeach\space is read only}%
      }%
      {%
```

Store current row number in \dtlrownum

```
        \dtlrownum=
          \csname dtl@row\romannumeral\dtlforeachlevel\endcsname\relax
```

Update information about this column (adding new column if necessary)

```
        \@dtl@updatekeys{\@dtl@thisdb}{#1}{#2}%
```

Get column index and store in \dtlcolumnnum

```
\expandafter\dtlcolumnnum\expandafter
  =\dtl@columnindex{\@dtl@thisdb}{#1}\relax
```

Set \dtlcurrentrow to the current row

```
\dtlcurrentrow =
  \csname @dtl@cur\romannumeral\dtlforeachlevel\endcsname
```

Does this row already have an entry with this key?

```
\edef\dtl@dogetentry{\noexpand\dtlgetentryfromcurrentrow
  {\noexpand\dtl@entry}{\number\dtlcolumnnum}%
}%
\dtl@dogetentry
\ifx\dtl@entry\dtlnovalue
```

There are no entries in this row for the given key. Store value in \@dtl@toks with one level expansion.

```
  \expandafter\@dtl@toks\expandafter{#2}%
```

Append this entry to the current row.

```
  \toks@gput@right@cx{@dtl@cur\romannumeral\dtlforeachlevel}%
  {%
    \noexpand\db@col@id@w \number\dtlcolumnnum
      \noexpand\db@col@id@end@
    \noexpand\db@col@elt@w \the\@dtl@toks
      \noexpand\db@col@elt@end@
    \noexpand\db@col@id@w \number\dtlcolumnnum
      \noexpand\db@col@id@end@
  }%
```

Print information to terminal and log file if in verbose mode.

```
   \dtl@message{Appended #1\space -> #2\space to database
     `\@dtl@thisdb'}%
\else
```

There is already an entry in this row for the given key

```
  \PackageError{datatool}{Can't append entry to row:
    there is already an entry for key `#1' in this row}{}%
\fi
}%
\fi
}
```

\DTLremoveentryfromrow

\DTLremoveentryfromrow{⟨*key*⟩}

Removes entry given by ⟨*key*⟩ from current row. (The current row is given by \@dtl@cur⟨*n*⟩ where ⟨*n*⟩ is roman numeral value of \dtlforeachlevel.

```
\newcommand*{\DTLremoveentryfromrow}[1]{%
  \ifnum\dtlforeachlevel=0\relax
    \PackageError{datatool}{\string\DTLremoventryfromrow\space
      can only be used inside \string\DTLforeach}{}%
  \else
```

Set \@dtl@thisdb to the current database name:

```
\expandafter\let\expandafter\@dtl@thisdb
  \csname @dtl@dbname@\romannumeral\dtlforeachlevel\endcsname
```

Check this isn't in \DTLforeach*

```
\@dtlifreadonly
{%
  \PackageError{datatool}{\string\DTLremoveentryfromrow\space
    can't be used inside \string\DTLforeach*}{The starred
    version of \string\DTLforeach\space is read only}%
}%
{%
```

Store current row number in \dtlrownum

```
  \dtlrownum=
    \csname dtl@row\romannumeral\dtlforeachlevel\endcsname\relax
```

Is there a column corresponding to this key?

```
  \@DTLifhaskey{\@dtl@thisdb}{#1}%
  {%
```

There exists a column for this key, so get the index:

```
    \@dtl@getcolumnindex{\thiscol}{\@dtl@thisdb}{#1}\relax
    \dtlcolumnnum=\thiscol\relax
```

Set \dtlcurrentrow to the current row

```
    \dtlcurrentrow =
      \csname @dtl@cur\romannumeral\dtlforeachlevel\endcsname
```

Does this row have an entry with this key?

```
    \edef\dtl@dogetentry{\noexpand\dtlgetentryfromcurrentrow
      {\noexpand\dtl@entry}{\number\dtlcolumnnum}%
    }%
    \dtl@dogetentry
    \ifx\dtl@entry\dtlnovalue
```

This row doesn't contain an entry with this key

```
      \PackageError{datatool}{Can't remove entry given by `#1'
         from current row in database `\@dtl@thisdb': no such
         entry}{The current row doesn't contain an entry for
         key `#1'}%
    \else
```

Split the current row around the unwanted entry

```
      \edef\@dtl@dosplitrow{%
        \noexpand\dtlsplitrow{\the\dtlcurrentrow}%
          {\number\dtlcolumnnum}{\noexpand\dtl@pre}%
          {\noexpand\dtl@post}%
      }%
      \@dtl@dosplitrow
```

Reconstruct row without unwanted entry

```
      \expandafter\@dtl@toks\expandafter{\dtl@pre}%
      \expandafter\toks@\expandafter{\dtl@post}%
      \edef\@dtl@tmp{\the\@dtl@toks \the\toks@}%
      \dtlcurrentrow=\expandafter{\@dtl@tmp}%
      \expandafter\global
```

```
            \csname @dtl@cur\romannumeral\dtlforeachlevel\endcsname
              = \dtlcurrentrow
          \dtl@message{Removed entry given by #1\space from current
            row of database `\@dtl@thisdb'}%
        \fi
      }%
      {%
        \PackageError{datatool}{Can't remove entry given by
          `#1' - no such key exists}{}%
      }%
    }%
  \fi
}
```

\DTLreplaceentryforrow

\DTLreplaceentryforrow{⟨*key*⟩}{⟨*value*⟩}

Replaces entry given by ⟨*key*⟩ in current row with ⟨*value*⟩. (The current row is given by the token register \@dtl@cur⟨*n*⟩ where ⟨*n*⟩ is roman numeral value of \dtlforeachlevel.

```
\newcommand*{\DTLreplaceentryforrow}[2]{%
  \ifnum\dtlforeachlevel=0\relax
    \PackageError{datatool}{\string\DTLreplaceentryforrow\space
      can only be used inside \string\DTLforeach}{}%
  \else
```

Set \@dtl@thisdb to the current database name:

```
    \expandafter\let\expandafter\@dtl@thisdb
      \csname @dtl@dbname@\romannumeral\dtlforeachlevel\endcsname
```

Check this isn't in \DTLforeach*

```
    \@dtlifreadonly
    {%
      \PackageError{datatool}{\string\DTLreplaceentryforrow\space
        can't be used inside \string\DTLforeach*}{The starred version
        of \string\DTLforeach\space is read only}%
    }%
    {%
```

Store current row number in \dtlrownum

```
      \dtlrownum=
        \csname dtl@row\romannumeral\dtlforeachlevel\endcsname\relax
```

Is there a column corresponding to this key?

```
      \@DTLifhaskey{\@dtl@thisdb}{#1}%
      {%
```

There exists a column for this key, so get the index:

```
        \@dtl@getcolumnindex{\thiscol}{\@dtl@thisdb}{#1}\relax
        \dtlcolumnnum=\thiscol\relax
```

Set \dtlcurrentrow to the current row

```
        \dtlcurrentrow =
          \csname @dtl@cur\romannumeral\dtlforeachlevel\endcsname
```

Does this row have an entry with this key?

```
\edef\dtl@dogetentry{\noexpand\dtlgetentryfromcurrentrow
  {\noexpand\dtl@entry}{\number\dtlcolumnnum}%
}%
\dtl@dogetentry
\ifx\dtl@entry\dtlnovalue
```

This row doesn't contain an entry with this key

```
  \PackageError{datatool}{Can't replace entry given by `#1'
     from current row in database `\@dtl@thisdb': no such
     entry}{The current row doesn't contain an entry for
     key `#1'}%
\else
```

Split the current row around the requested entry

```
  \edef\@dtl@dosplitrow{%
    \noexpand\dtlsplitrow{\the\dtlcurrentrow}%
      {\number\dtlcolumnnum}{\noexpand\dtl@pre}%
      {\noexpand\dtl@post}%
  }%
  \@dtl@dosplitrow
```

Reconstruct row with new entry

```
  \@dtl@toks{#2}% new value
  \expandafter\@dtl@before\expandafter{\dtl@pre}%
  \expandafter\@dtl@after\expandafter{\dtl@post}%
  \toks@gconcat@middle@cx
    {@dtl@cur\romannumeral\dtlforeachlevel}%
    {\@dtl@before}%
    {%
      \noexpand\db@col@id@w \number\dtlcolumnnum
        \noexpand\db@col@id@end@%
      \noexpand\db@col@elt@w \the\@dtl@toks
        \noexpand\db@col@elt@end@%
      \noexpand\db@col@id@w \number\dtlcolumnnum
        \noexpand\db@col@id@end@%
    }%
    {\@dtl@after}%
```

Print information to terminal and log file if in verbose mode.

```
  \dtl@message{Updated #1\space -> #2\space in database
    `\@dtl@thisdb'}%
\fi
}%
{%
```

There doesn't exist a column for this key.

```
  \PackageError{datatool}{Can't replace key `#1' - no such
    key in database `\@dtl@thisdb'}{}%
}%
}%
\fi
}
```

`\DTLremovecurrentrow`

```
\DTLremovecurrentrow
```

Removes current row. This just sets the current row to empty

```
\newcommand*{\DTLremovecurrentrow}{%
  \ifnum\dtlforeachlevel=0\relax
    \PackageError{datatool}{\string\DTLremovecurrentrow\space can
      only be used inside \string\DTLforeach}{}%
  \else
```

Set `\@dtl@thisdb` to the current database name:

```
    \expandafter\let\expandafter\@dtl@thisdb
      \csname @dtl@dbname@\romannumeral\dtlforeachlevel\endcsname
```

Check this isn't in `\DTLforeach*`

```
    \@dtlifreadonly
    {%
      \PackageError{datatool}{\string\DTLreplaceentryforrow\space
        can't be used inside \string\DTLforeach*}{The starred version
        of \string\DTLforeach\space is read only}%
    }%
    {%
```

Set the current row to `\@nil` (`\DTLforeach` needs to check for this)

```
      \expandafter\global
        \csname @dtl@cur\romannumeral\dtlforeachlevel\endcsname
          ={\@nil}%
    }%
  \fi
}
```

`\DTLaddentryforrow`

`\DTLaddentryforrow{`⟨*db name*⟩`}{`⟨*assign list*⟩`}{`⟨*condition*⟩`}{`⟨*key*⟩`}{`⟨*value*⟩`}`

Adds the entry with key given by ⟨*key*⟩ and value given by ⟨*value*⟩ to the first row in the database ⟨*db name*⟩ which satisfies the condition given by ⟨*condition*⟩. The ⟨*assign list*⟩ is the same as for `\DTLforeach` and may be used to set the values which are to be tested in ⟨*condition*⟩.

```
\newcommand{\DTLaddentryforrow}[5]{%
```

Iterate through the data base until condition is met

```
  \DTLifdbexists{#1}%
  {
    \def\@dtl@notdone{\PackageError{datatool}{Unable to add entry
      given by key `#4': condition not met for any row in database
      `#1'}{}}%
```

Iterate through each row

```
    \DTLforeach[#3]{#1}{#2}%
    {%
```

add entry to this row

```
      \DTLappendtorow{#4}{#5}%
```

disable error message

```
      \let\@dtl@notdone\relax
```

break out of loop

```
        \dtlbreak
      }%
      \@dtl@notdone
    }%
    {%
      \PackageError{datatool}{Unable to add entry given by key ‘#4’:
        database ‘#1’ doesn’t exist}{}%
    }%
  }
```

\DTLforeachkeyinrow

\DTLforeachkeyinrow{⟨*cmd*⟩}{⟨*text*⟩}

Iterates through each key in the current row of \DTLforeach, and does ⟨*text*⟩.

```
\newcommand*{\DTLforeachkeyinrow}[2]{%
  \ifnum\dtlforeachlevel=0\relax
    \PackageError{datatool}{\string\DTLforeachkeyinrow\space can only
     be used inside \string\DTLforeach}{}%
  \else
```

Set \@dtl@thisdb to the current database name:

```
    \expandafter\let\expandafter\@dtl@thisdb
      \csname @dtl@dbname@\romannumeral\dtlforeachlevel\endcsname
```

Iterate through key list

```
    \dtlforeachkey(\dtlkey,\dtlcol,\dtltype,\dtlheader)\in
      \@dtl@thisdb\do{%
```

store row in \dtlcurrentrow (This may get nested so need to do it here instead of outside this loop in case ⟨*text*⟩ changes it.)

```
      \dtlcurrentrow =
        \csname @dtl@cur\romannumeral\dtlforeachlevel\endcsname
```

Get the value for this key and store in #1

```
      \edef\dtl@dogetentry{\noexpand\dtlgetentryfromcurrentrow
        {\noexpand#1}{\dtlcol}}%
      \dtl@dogetentry
```

Check if null

```
      \ifx#1\dtlnovalue
        \ifnum0\dtltype=0\relax
```

Data type is ⟨*empty*⟩ or 0, so set to string null.

```
          \let#1=\DTLstringnull
        \else
```

Data type is numerical, so set to number null.

```
          \let#1=\DTLnumbernull
        \fi
      \fi
```

Make #1 global in case this is in a tabular environment (or something similar)

```
      \global\let#1#1%
```

Store loop body so that any scoping commands (such as &) don't cause a problem for \ifx

```
        \def\@dtl@loop@body{#2}%
        \@dtl@loop@body
      }%
    \fi
  }
```

10.8 Displaying Database

This section defines commands to display the entire database in a tabular or longtable environment.

\dtlbetweencols This specifies what to put between the column alignment specifiers.

```
\newcommand*{\dtlbetweencols}{}
```

\dtlbeforecols This specifies what to put before the first column alignment specifier.

```
\newcommand*{\dtlbeforecols}{}
```

\dtlaftercols This specifies what to put after the last column alignment specifier.

```
\newcommand*{\dtlaftercols}{}
```

\dtlstringalign Alignment character for columns containing strings

```
\newcommand*{\dtlstringalign}{l}
```

\dtlintalign Alignment character for columns containing integers

```
\newcommand*{\dtlintalign}{r}
```

\dtlrealalign Alignment character for columns containing real numbers

```
\newcommand*{\dtlrealalign}{r}
```

\dtlcurrencyalign Alignment character for columns containing currency numbers

```
\newcommand*{\dtlcurrencyalign}{r}
```

\dtladdalign \dtladdalign{⟨*cs*⟩}{⟨*type*⟩}{⟨*col num*⟩}{⟨*max cols*⟩}

Adds tabular column alignment character to ⟨*cs*⟩ for column ⟨*col num*⟩ which contains data type ⟨*type*⟩.

```
\newcommand*{\dtladdalign}[4]{%
  \ifnum#3=1\relax
    \protected@edef#1{\dtlbeforecols}%
  \else
    \protected@edef#1{#1\dtlbetweencols}%
  \fi
  \ifx\@empty#2\@empty
    \protected@edef#1{#1c}%
  \else
    \ifcase#2\relax
```

string

```
        \protected@edef#1{#1\dtlstringalign}%
      \or
```

integer

```
        \protected@edef#1{#1\dtlintalign}%
      \or
```

real number

```
        \protected@edef#1{#1\dtlrealalign}%
      \or
```

currency

```
        \protected@edef#1{#1\dtlcurrencyalign}%
      \else
```

Unknown type

```
        \protected@edef#1{#1c}%
        \PackageError{datatool}{Unknown data type `#2'}{}%
      \fi
    \fi
    \ifnum#3=#4\relax
      \protected@edef#1{#1\dtlaftercols}%
    \fi
  }
```

\dtlheaderformat

\dtlheaderformat{⟨*text*⟩}

Specifies how to format the column title.

```
\newcommand*{\dtlheaderformat}[1]{\null\hfil\textbf{#1}\hfil\null}
```

\dtlstringformat

\dtlstringformat{⟨*text*⟩}

Specifies how to format entries in columns with string data type.

```
\newcommand*{\dtlstringformat}[1]{#1}
```

\dtlintformat

\dtlintformat{⟨*text*⟩}

Specifies how to format entries in columns with integer data type.

```
\newcommand*{\dtlintformat}[1]{#1}
```

\dtlrealformat

\dtlrealformat{⟨*text*⟩}

Specifies how to format entries in columns with real data type.

```
\newcommand*{\dtlrealformat}[1]{#1}
```

\dtlcurrencyformat

\dtlcurrencyformat{⟨*text*⟩}

Specifies how to format entries in columns with currency data type.

```
\newcommand*{\dtlcurrencyformat}[1]{#1}
```

`\dtldisplaystarttab` Indicates what to do just after `\begin{tabular}{`⟨*column specs*⟩`}` (e.g. `\hline`).

```
\newcommand*{\dtldisplaystarttab}{}
```

`\dtldisplayendtab` Indicates what to do just before `\end{tabular}`.

```
\newcommand*{\dtldisplayendtab}{}
```

`\dtldisplayafterhead` Indicates what to do after the header row, before the first row of data.

```
\newcommand*{\dtldisplayafterhead}{}
```

`\dtldisplaystartrow` Indicates what to do at the start of each row (not including the header row or the first row of data).

```
\newcommand*{\dtldisplaystartrow}{}
```

`\DTLdisplaydb`

`\DTLdisplaydb{`⟨*db*⟩`}`

Displays the database ⟨*db*⟩ in a tabular environment.

```
\newcommand*{\DTLdisplaydb}[1]{%
```

Initialise: only want `&` between columns

```
  \def\@dtl@doamp{\gdef\@dtl@doamp{&}}
  \def\@dtl@resetdoamp{\gdef\@dtl@doamp{\gdef\@dtl@doamp{&}}}
```

Store maximum number of columns

```
  \edef\@dtl@maxcols{\expandafter\number
    \csname dtlcols@#1\endcsname}%
```

Argument for tabular environment

```
  \def\@dtl@tabargs{}%
  \dtlforeachkey(\@dtl@key,\@dtl@idx,\@dtl@type,\@dtl@head)%
    \in{#1}\do
  {%
    \dtladdalign\@dtl@tabargs\@dtl@type\@dtl@idx\@dtl@maxcols
  }%
```

Begin tabular environment

```
  \edef\@dtl@dobegintab{\noexpand\begin{tabular}{\@dtl@tabargs}}%
  \@dtl@dobegintab
```

Do start hook

```
  \dtldisplaystarttab
```

Reset `\@dtl@doamp` so it doesn't do an ampersand at the start of the first column.

```
  \@dtl@resetdoamp
```

Do the header row.

```
  \dtlforeachkey(\@dtl@key,\@dtl@idx,\@dtl@type,\@dtl@head)%
    \in{#1}\do
  {%
    \@dtl@doamp
    \dtlheaderformat{\@dtl@head}%
  }%
  \\%
```

Do the after header hook

```
\dtldisplayafterhead
```

Reset `\@dtl@doamp` so it doesn't do an ampersand at the start of the first column.

```
\@dtl@resetdoamp
```

Iterate through each row of the database

```
\DTLforeach*{#1}{}{%
```

Do the start row hook if not the first row

```
    \DTLiffirstrow{}{\\\dtldisplaystartrow}%
```

Reset `\@dtl@doamp` so it doesn't do an ampersand at the start of the first column.

```
    \@dtl@resetdoamp
```

Iterate through each column.

```
    \DTLforeachkeyinrow{\@dtl@val}%
    {%
```

Need to make value global as it needs to be used after the ampersand.

```
      \global\let\@dtl@val\@dtl@val
      \@dtl@doamp
```

`\DTLforeachkeyinrow` sets `\dtltype` to the data type for the current key. This can be used to determine which format to use for this entry.

```
      \@dtl@datatype=0\dtltype\relax
      \ifcase\@dtl@datatype
        \dtlstringformat\@dtl@val
      \or
        \dtlintformat\@dtl@val
      \or
        \dtlrealformat\@dtl@val
      \or
        \dtlcurrencyformat\@dtl@val
      \else
        \@dtl@val
      \fi
    }%
  }%
  \dtldisplayendtab
  \end{tabular}%
}
```

Define keys to use in the optional argument of `\DTLdisplaylongdb`.
The caption key sets the caption for the longtable.

```
\define@key{displaylong}{caption}{\def\@dtl@cap{#1}}
```

The contcaption key sets the continuation caption for the longtable.

```
\define@key{displaylong}{contcaption}{\def\@dtl@contcap{#1}}
```

The shortcaption key sets the lof caption for the longtable.

```
\define@key{displaylong}{shortcaption}{\def\@dtl@shortcap{#1}}
```

The label key sets the label for the longtable.

```
\define@key{displaylong}{label}{\def\@dtl@label{#1}}
```

The foot key sets the longtable foot

```
\define@key{displaylong}{foot}{\def\@dtl@foot{#1}}
```

The lastfoot key sets the longtable last foot

```
\define@key{displaylong}{lastfoot}{\def\@dtl@lastfoot{#1}}
```

`\@dtl@resetdostartrow` Resets start row hook so that it skips the first row.

```
\newcommand*{\@dtl@resetdostartrow}{%
  \gdef\@dtl@dostartrow{%
    \gdef\@dtl@dostartrow{\\\dtldisplaystartrow}}%
}
```

`\DTLdisplaylongdb` `\DTLdisplaylongdb[⟨options⟩]{⟨db⟩}`

Displays the database ⟨*db*⟩ in a longtable environment. (User needs to load longtable).

```
\newcommand*{\DTLdisplaylongdb}[2][]{%
```

Initialise.

```
  \def\@dtl@cap{\@nil}%
  \def\@dtl@contcap{\@nil}%
  \def\@dtl@label{\@nil}%
  \def\@dtl@shortcap{\@dtl@cap}%
  \def\@dtl@foot{\@nil}%
  \def\@dtl@lastfoot{\@nil}%
```

Set the options

```
  \setkeys{displaylong}{#1}%
```

Only want & between columns

```
  \def\@dtl@doamp{\gdef\@dtl@doamp{&}}
  \def\@dtl@resetdoamp{\gdef\@dtl@doamp{\gdef\@dtl@doamp{&}}}
  \@dtl@resetdostartrow
```

Store maximum number of columns

```
  \edef\@dtl@maxcols{\expandafter\number
    \csname dtlcols@#2\endcsname}%
```

Argument for longtable environment

```
  \def\@dtl@tabargs{}%
  \dtlforeachkey(\@dtl@key,\@dtl@idx,\@dtl@type,\@dtl@head)%
    \in{#2}\do
  {%
    \dtladdalign\@dtl@tabargs\@dtl@type\@dtl@idx\@dtl@maxcols
  }%
```

Start the longtable environment.

```
  \edef\@dtl@dobegintab{\noexpand\begin{longtable}{\@dtl@tabargs}}%
  \@dtl@dobegintab
```

Do start hook.

```
  \dtldisplaystarttab
```

Is a foot required?

```
  \ifx\@dtl@foot\@nnil
  \else
    \@dtl@foot\endfoot
  \fi
```

Is a last foot required?

```
\ifx\@dtl@lastfoot\@nnil
\else
  \@dtl@lastfoot\endlastfoot
\fi
```

Is a caption required?

```
\ifx\@dtl@cap\@nnil
```

No caption required, just do header row.

```
    \@dtl@resetdoamp
    \dtlforeachkey(\@dtl@key,\@dtl@idx,\@dtl@type,\@dtl@head)%
      \in{#2}\do
    {\@dtl@doamp{\dtlheaderformat{\@dtl@head}}}%
    \@dtl@resetdoamp
    \@dtl@resetdostartrow
    \endhead\dtldisplayafterhead
\else
```

Caption is required

```
    \caption[\@dtl@shortcap]{\@dtl@cap}%
```

Is a label required?

```
    \ifx\@dtl@label\@nnil
    \else
      \label{\@dtl@label}%
    \fi
    \\%
```

Do header row.

```
    \@dtl@resetdoamp
    \dtlforeachkey(\@dtl@key,\@dtl@idx,\@dtl@type,\@dtl@head)%
      \in{#2}\do
    {\@dtl@doamp{\dtlheaderformat{\@dtl@head}}}%
    \@dtl@resetdoamp
    \@dtl@resetdostartrow
    \endfirsthead
```

Is a continuation caption required?

```
    \ifx\@dtl@contcap\@nnil
      \caption{\@dtl@cap}%
    \else
      \caption{\@dtl@contcap}%
    \fi
    \\%
```

Do header row.

```
      \@dtl@resetdoamp
      \dtlforeachkey(\@dtl@key,\@dtl@idx,\@dtl@type,\@dtl@head)%
      \in{#2}\do
      {\@dtl@doamp{\dtlheaderformat{\@dtl@head}}}%
      \@dtl@resetdoamp
      \@dtl@resetdostartrow
    \endhead\dtldisplayafterhead
\fi
```

Iterate through each row of the database

```
    \DTLforeach*{#2}{}{%
      \@dtl@dostartrow
      \@dtl@resetdoamp
```

Iterate through each column

```
      \DTLforeachkeyinrow{\@dtl@val}%
      {%
        \global\let\@dtl@val\@dtl@val
        \@dtl@doamp
```

`\DTLforeachkeyinrow` sets `\dtltype` to the data type for the current key. This can be used to determine which format to use for this entry.

```
        \@dtl@datatype=0\dtltype\relax
        \ifcase\@dtl@datatype
          \dtlstringformat\@dtl@val
        \or
          \dtlintformat\@dtl@val
        \or
          \dtlrealformat\@dtl@val
        \or
          \dtlcurrencyformat\@dtl@val
        \fi
      }%
    }%
    \dtldisplayendtab
    \end{longtable}%
  }
```

10.9 Editing Databases

`\dtlswaprows`

`\dtlswaprows{`⟨*db*⟩`}{`⟨*row1 idx*⟩`}{`⟨*row2 idx*⟩`}`

Swaps the rows with indices ⟨*row1 idx*⟩ and ⟨*row2 idx*⟩ in the database ⟨*db*⟩. (Doesn't check if data base exists of if indices are out of bounds.)

```
  \newcommand*{\dtlswaprows}[3]{%
    \ifnum#2=#3\relax
```

Attempt to swap row with itself: do nothing.

```
    \else
```

Let row A be the row with the lower index and row B be the row with ther higher index.

```
      \ifnum#2<#3\relax
        \edef\@dtl@rowAidx{\number#2}%
        \edef\@dtl@rowBidx{\number#3}%
      \else
        \edef\@dtl@rowAidx{\number#3}%
        \edef\@dtl@rowBidx{\number#2}%
      \fi
```

Split the database around row A.

```
      \edef\@dtl@dosplit{\noexpand\dtlgetrow{#1}{\@dtl@rowAidx}}%
      \@dtl@dosplit
```

Store first part of database in `\@dtl@firstpart`.

```
\expandafter\def\expandafter\@dtl@firstpart\expandafter
  {\the\dtlbeforerow}%
```

Store row A in `\@dtl@toksA`.

```
\@dtl@toksA=\dtlcurrentrow
```

Split the second part (everything after row A).

```
\edef\@dtl@dosplit{\noexpand\@dtlgetrow
  {\the\dtlafterrow}{\@dtl@rowBidx}}%
\@dtl@dosplit
```

Store the mid part (everything between row A and row B)

```
\expandafter\def\expandafter\@dtl@secondpart\expandafter
  {\the\dtlbeforerow}%
```

Store row B in `\@dtl@toksB`.

```
\@dtl@toksB=\dtlcurrentrow
```

Store the last part (everything after row B).

```
\expandafter\def\expandafter\@dtl@thirdpart\expandafter
  {\the\dtlafterrow}%
```

Reconstruct database: store first part in `\toks@`

```
\toks@=\expandafter{\@dtl@firstpart}%
```

Store mid part in `\dtl@toks`

```
\@dtl@toks=\expandafter{\@dtl@secondpart}%
```

Format data for first part, row B and mid part.

```
\edef\@dtl@tmp{\the\toks@
 \noexpand\db@row@elt@w%
 \noexpand\db@row@id@w \@dtl@rowAidx\noexpand\db@row@id@end@%
 \the\@dtl@toksB
 \noexpand\db@row@id@w \@dtl@rowAidx\noexpand\db@row@id@end@%
 \noexpand\db@row@elt@end@%
 \the\@dtl@toks}%
```

Store data so far in `\toks@`.

```
\toks@=\expandafter{\@dtl@tmp}%
```

Store last part in `\dtl@toks`.

```
\@dtl@toks=\expandafter{\@dtl@thirdpart}%
```

Format row A and end part.

```
\edef\@dtl@tmp{\the\toks@
 \noexpand\db@row@elt@w%
 \noexpand\db@row@id@w \@dtl@rowBidx\noexpand\db@row@id@end@%
 \the\@dtl@toksA
 \noexpand\db@row@id@w \@dtl@rowBidx\noexpand\db@row@id@end@%
 \noexpand\db@row@elt@end@%
\the\@dtl@toks}%
```

Update the database

```
\expandafter\global\csname dtldb@#1\endcsname=\expandafter
  {\@dtl@tmp}%
\fi
}
```

\dtl@decrementrows

\dtl@decrementrows{⟨*toks*⟩}{⟨*n*⟩}

decrement by 1 all rows in ⟨*toks*⟩ with row index above ⟨*n*⟩

```
\newcommand*{\dtl@decrementrows}[2]{%
  \def\@dtl@newlist{}%
  \edef\@dtl@min{\number#2}%
  \expandafter\@dtl@decrementrows\the#1%
    \db@row@elt@w%
      \db@row@id@w \@nil\db@row@id@end@%
      \db@row@id@w \@nil\db@row@id@end@%
    \db@row@elt@end@%
    \@nil
  #1=\expandafter{\@dtl@newlist}%
}
```

\@dtl@decrementrows

```
\def\@dtl@decrementrows\db@row@elt@w\db@row@id@w #1\db@row@id@end@%
#2\db@row@id@w #3\db@row@id@end@\db@row@elt@end@#4\@nil{%
  \def\@dtl@thisrow{#1}%
  \ifx\@dtl@thisrow\@nnil
    \let\@dtl@donextdec=\@dtl@gobbletonil
  \else
    \ifnum\@dtl@thisrow>\@dtl@min
      \@dtl@tmpcount=\@dtl@thisrow\relax
      \advance\@dtl@tmpcount by -1\relax
      \toks@{#2}%
      \@dtl@toks=\expandafter{\@dtl@newlist}%
      \edef\@dtl@newlist{\the\@dtl@toks
        \noexpand\db@row@elt@w% row header
        \noexpand\db@row@id@w \number\@dtl@tmpcount
          \noexpand\db@row@id@end@% row id
         \the\toks@ % row contents
        \noexpand\db@row@id@w \number\@dtl@tmpcount
          \noexpand\db@row@id@end@% row id
        \noexpand\db@row@elt@end@% row end
      }%
    \else
      \toks@{#2}%
      \@dtl@toks=\expandafter{\@dtl@newlist}%
      \edef\@dtl@newlist{\the\@dtl@toks
        \noexpand\db@row@elt@w% row header
        \noexpand\db@row@id@w #1%
          \noexpand\db@row@id@end@% row id
         \the\toks@ % row contents
        \noexpand\db@row@id@w #3%
          \noexpand\db@row@id@end@% row id
        \noexpand\db@row@elt@end@% row end
      }%
    \fi
    \let\@dtl@donextdec=\@dtl@decrementrows
  \fi
  \@dtl@donextdec#4\@nil
}
```

\DTLremoverow

\DTLremoverow{⟨*db*⟩}{⟨*row index*⟩}

Remove row with given index from database named ⟨*db*⟩.

```
\newcommand*{\DTLremoverow}[2]{%
```

Check database exists

```
  \DTLifdbexists{#1}%
  {%
```

Check index if index is out of bounds

```
    \ifnum#2>0\relax
```

Check if data base has at least ⟨*row index*⟩ rows

```
      \expandafter\ifnum\csname dtlrows@#1\endcsname<#2\relax
        \expandafter\ifnum\csname dtlrows@#1\endcsname=1\relax
          \PackageError{datatool}{Can't remove row `\number#2' from
            database `#1': no such row}{Database `#1' only has
            1 row}%
        \else
          \PackageError{datatool}{Can't remove row `\number#2' from
            database `#1': no such row}{Database `#1' only has
            \expandafter\number\csname dtlrows@#1\endcsname\space
            rows}%
        \fi
      \else
        \@DTLremoverow{#1}{#2}%
      \fi
    \else
      \PackageError{datatool}{Can't remove row \number#2: index
        out of bounds}{Row indices start at 1}%
    \fi
  }%
  {%
    \PackageError{datatool}{Can't remove row: database `#1' doesn't
      exist}{}%
  }%
}
```

\@DTLremoverow

\@DTLremoverow{⟨*db*⟩}{⟨*row index*⟩}

Doesn't perform any checks for the existance of the database or if the index is in range.

```
\newcommand*{\@DTLremoverow}[2]{%
```

Get row from data base

```
      \edef\dtl@dogetrow{\noexpand\dtlgetrow{#1}{\number#2}}%
      \dtl@dogetrow
```

Update the row indices

```
      \expandafter\dtl@decrementrows\expandafter
        {\dtlbeforerow}{#2}%
      \expandafter\dtl@decrementrows\expandafter
        {\dtlafterrow}{#2}%
```

Reconstruct database

```
        \edef\dtl@tmp{\the\dtlbeforerow \the\dtlafterrow}%
        \expandafter\global\csname dtldb@#1\endcsname
          =\expandafter{\dtl@tmp}%
```

decrement row counter

```
        \expandafter\global\expandafter\advance
          \csname dtlrows@#1\endcsname by -1\relax
  }
```

10.10 Database Functions

\DTLsumforkeys

\DTLsumforkeys[⟨*condition*⟩][⟨*assign list*⟩]{⟨*db list*⟩}{⟨*key list*⟩}{⟨*cmd*⟩}

Sums all entries for key ⟨*key*⟩ over all databases listed in ⟨*db list*⟩, and stores in ⟨*cmd*⟩, which must be a control sequence. The first argument ⟨*condition*⟩ is the same as that for \DTLforeach. The second optional argument provides an assignment list to pass to \DTLforeach in case extra information is need by ⟨*condition*⟩.

```
  \newcommand*{\DTLsumforkeys}[1][\boolean{true}\and
   \DTLisnumerical{\DTLthisval}]{%
    \def\@dtl@cond{#1}%
    \@dtlsumforkeys
  }
```

\@dtlsumforkeys

```
  \newcommand*{\@dtlsumforkeys}[4][]{%
    \def#4{0}%
```

Iterate over all the listed data bases

```
    \@for\@dtl@dbname:=#2\do{%
```

Iterate through this database (using read only version)

```
      \@sDTLforeach{\@dtl@dbname}%
      {#1}% assignment list
      {%
```

Iterate through key list.

```
        \@for\@dtl@key:=#3\do{%
          \@sdtl@getcolumnindex{\@dtl@col}{\@dtl@dbname}{\@dtl@key}%
          \dtlcurrentrow=\expandafter{\dtl@thisrow}%
          \dtlgetentryfromrow{\DTLthisval}{\@dtl@col}{\dtlcurrentrow}%
          \expandafter\ifthenelse\expandafter{\@dtl@cond}%
            {\DTLadd{#4}{#4}{\DTLthisval}}{}%
        }%
      }%
    }%
  }
```

\DTLsumcolumn

\DTLsumcolumn{⟨*db*⟩}{⟨*key*⟩}{⟨*cmd*⟩}

Quicker version of \DTLsumforkeys that just sums over one column (specified by ⟨*key*⟩) for a single database (specified by ⟨*db*⟩) and stores the result in ⟨*cmd*⟩.

```
\newcommand*{\DTLsumcolumn}[3]{%
  \def#3{0}%
```

Check data base exists

```
  \DTLifdbexists{#1}%
  {%
```

Check column exists

```
    \@sDTLifhaskey{#1}{#2}%
    {%
      \@sdtlforcolumn{\DTLthisval}{#1}{#2}%
      {%
        \DTLadd{#3}{#3}{\DTLthisval}%
      }%
    }%
```

key not defined for this data base

```
    {%
      \PackageError{datatool}{Key ‘#2’ doesn’t
        exist in database ‘#1’}{}%
    }%
  }%
```

data base doesn't exist

```
  {%
    \PackageError{datatool}{Data base ‘#1’ doesn’t
      exist}{}%
  }%
}
```

\DTLmeanforkeys

\DTLmeanforkeys[⟨*condition*⟩][⟨*assign list*⟩]{⟨*db list*⟩}{⟨*key list*⟩}{⟨*cmd*⟩}

Computes the arithmetic mean of all entries for each key in ⟨*key list*⟩ over all databases in ⟨*db list*⟩, and stores in ⟨*cmd*⟩, which must be a control sequence. The first argument ⟨*condition*⟩ is the same as that for \DTLforeach. The second optional argument allows an assignment list to be passed to \DTLforeach.

```
\newcommand*{\DTLmeanforkeys}[1][\boolean{true}\and
 \DTLisnumerical{\DTLthisval}]{%
  \def\@dtl@cond{#1}%
  \@dtlmeanforkeys
}
```

\@dtl@elements

Count register to keep track of number of elements

```
\newcount\@dtl@elements
```

\@dtlmeanforkeys

```
\newcommand*{\@dtlmeanforkeys}[4][]{%
  \def#4{0}%
  \@dtl@elements=0\relax
```

Iterate over all the listed data bases

```
  \@for\@dtl@dbname:=#2\do{%
```

Iterate through this database (using read only version)

```
      \@sDTLforeach{\@dtl@dbname}%
      {#1}% assignment list
      {%
```

Iterate through key list.

```
        \@for\@dtl@key:=#3\do{%
          \@sdtl@getcolumnindex{\@dtl@col}{\@dtl@dbname}{\@dtl@key}%
          \dtlcurrentrow=\expandafter{\dtl@thisrow}%
          \dtlgetentryfromrow{\DTLthisval}{\@dtl@col}{\dtlcurrentrow}%
          \expandafter\ifthenelse\expandafter{\@dtl@cond}%
          {%
            \DTLadd{#4}{#4}{\DTLthisval}%
            \advance\@dtl@elements by 1\relax
          }{}%
        }%
      }%
    }%
```

Divide total by number of elements summed.

```
    \ifnum\@dtl@elements=0\relax
      \PackageError{datatool}{Unable to evaluate mean: no data}{}%
    \else
      \edef\@dtl@n{\number\@dtl@elements}%
      \DTLdiv{#4}{#4}{\@dtl@n}%
    \fi
  }
```

\DTLmeanforcolumn

\DTLmeanforcolumn{⟨*db*⟩}{⟨*key*⟩}{⟨*cmd*⟩}

Quicker version of \DTLmeanforkeys that just computes the mean over one column (specified by ⟨*key*⟩) for a single database (specified by ⟨*db*⟩) and stores the result in ⟨*cmd*⟩.

```
  \newcommand*{\DTLmeanforcolumn}[3]{%
    \def#3{0}%
    \@dtl@elements=0\relax
```

Check data base exists

```
    \DTLifdbexists{#1}%
    {%
```

Check column exists

```
      \@sDTLifhaskey{#1}{#2}%
      {%
        \@sdtlforcolumn{\DTLthisval}{#1}{#2}%
        {%
          \DTLadd{#3}{#3}{\DTLthisval}%
          \advance\@dtl@elements by 1\relax
        }%
        \ifnum\@dtl@elements=0\relax
          \PackageError{datatool}{Can't compute mean for
           column `#2' in database `#1': no data}{}%
        \else
```

```
            \edef\@dtl@n{\number\@dtl@elements}%
            \DTLdiv{#3}{#3}{\@dtl@n}%
          \fi
        }%
```

key not defined for this data base

```
        {%
          \PackageError{datatool}{Key ‘#2’ doesn’t
            exist in database ‘#1’}{}%
        }%
    }%
```

data base doesn't exist

```
    {%
      \PackageError{datatool}{Data base ‘#1’ doesn’t
        exist}{}%
    }%
  }
```

\DTLvarianceforkeys

\DTLvarianceforkeys[⟨*condition*⟩][⟨*assign list*⟩]{⟨*db list*⟩}{⟨*key list*⟩}{⟨*cmd*⟩}

Computes the variance of all entries for each key in ⟨*key list*⟩ over all databases in ⟨*db list*⟩, and stores in ⟨*cmd*⟩, which must be a control sequence. The first optional argument ⟨*condition*⟩ is the same as that for \DTLforeach. The second optional argument is an assignment list to pass to \DTLforeach in case it is required for the condition.

```
\newcommand*{\DTLvarianceforkeys}[1][\boolean{true}\and
 \DTLisnumerical{\DTLthisval}]{%
  \def\@dtl@cond{#1}%
  \@dtlvarianceforkeys
}
```

\@dtlmeanforkeys

```
\newcommand*{\@dtlvarianceforkeys}[4][]{%
  \@dtlmeanforkeys[#1]{#2}{#3}{\dtl@mean}%
  \def#4{0}%
  \@dtl@elements=0\relax
```

Iterate over all the listed data bases

```
  \@for\@dtl@dbname:=#2\do{%
```

Iterate through this database (using read only version)

```
    \@sDTLforeach{\@dtl@dbname}%
    {#1}% assignment list
    {%
```

Iterate through key list.

```
      \@for\@dtl@key:=#3\do{%
        \@sdtl@getcolumnindex{\@dtl@col}{\@dtl@dbname}{\@dtl@key}%
        \dtlcurrentrow=\expandafter{\dtl@thisrow}%
        \dtlgetentryfromrow{\DTLthisval}{\@dtl@col}{\dtlcurrentrow}%
        \expandafter\ifthenelse\expandafter{\@dtl@cond}%
        {%
```

compute $(x_i - \mu)^2$

```
            \DTLsub{\dtl@diff}{\DTLthisval}{\dtl@mean}%
            \DTLmul{\dtl@diff}{\dtl@diff}{\dtl@diff}%
            \DTLadd{#4}{#4}{\dtl@diff}%
            \advance\@dtl@elements by 1\relax
          }{}%
        }%
      }%
    }%
```

Divide by number of elements.

```
    \ifnum\@dtl@elements=0\relax
      \PackageError{datatool}{Unable to evaluate variance: no data}{}%
    \else
      \edef\@dtl@n{\number\@dtl@elements}%
      \DTLdiv{#4}{#4}{\@dtl@n}%
    \fi
  }
```

`\DTLvarianceforcolumn`

`\DTLvarianceforcolumn{`⟨*db*⟩`}{`⟨*key*⟩`}{`⟨*cmd*⟩`}`

Quicker version of `\DTLvarianceforkeys` that just computes the variance over one column (specified by ⟨*key*⟩) for a single database (specified by ⟨*db*⟩) and stores the result in ⟨*cmd*⟩.

```
  \newcommand*{\DTLvarianceforcolumn}[3]{%
    \DTLmeanforcolumn{#1}{#2}{\dtl@mean}%
    \def#3{0}%
    \@dtl@elements=0\relax
```

Check data base exists

```
    \DTLifdbexists{#1}%
    {%
```

Check column exists

```
        \@sDTLifhaskey{#1}{#2}%
        {%
          \@sdtlforcolumn{\DTLthisval}{#1}{#2}%
          {%
```

compute $(x_i - \mu)^2$

```
            \DTLsub{\dtl@diff}{\DTLthisval}{\dtl@mean}%
            \DTLmul{\dtl@diff}{\dtl@diff}{\dtl@diff}%
            \DTLadd{#3}{#3}{\dtl@diff}%
            \advance\@dtl@elements by 1\relax
          }%
          \ifnum\@dtl@elements=0\relax
            \PackageError{datatool}{Can't compute variance for
             column `#2' in database `#1': no data}{}%
          \else
            \edef\@dtl@n{\number\@dtl@elements}%
            \DTLdiv{#3}{#3}{\@dtl@n}%
          \fi
        }%
```

key not defined for this data base

```
      {%
        \PackageError{datatool}{Key ‘#2’ doesn’t
          exist in database ‘#1’}{}%
      }%
    }%
```

data base doesn't exist

```
    {%
      \PackageError{datatool}{Data base ‘#1’ doesn’t
        exist}{}%
    }%
  }
```

\DTLsdforkeys

\DTLsdforkeys[⟨*condition*⟩][⟨*assign list*⟩]{⟨*db list*⟩}{⟨*key list*⟩}{⟨*cmd*⟩}

Computes the standard deviation of all entries for each key in ⟨*key list*⟩ over all databases in ⟨*db list*⟩, and stores in ⟨*cmd*⟩, which must be a control sequence. The first optional argument ⟨*condition*⟩ is the same as that for \DTLforeach. The second optional argument is an assignment list for \DTLforeach in case it is needed for the condition.

```
\newcommand*{\DTLsdforkeys}[1][\boolean{true}\and
\DTLisnumerical{\DTLthisval}]{%
  \def\@dtl@cond{#1}%
  \@dtlsdforkeys
}
```

\@dtlsdforkeys

```
\newcommand*{\@dtlsdforkeys}[4][]{%
  \@dtlvarianceforkeys[#1]{#2}{#3}{#4}%
  \DTLsqrt{#4}{#4}%
}
```

\DTLsdforcolumn

\DTLsdforcolumn{⟨*db*⟩}{⟨*key*⟩}{⟨*cmd*⟩}

Quicker version of \DTLsdforkeys that just computes the standard deviation over one column (specified by ⟨*key*⟩) for a single database (specified by ⟨*db*⟩) and stores the result in ⟨*cmd*⟩.

```
\newcommand*{\DTLsdforcolumn}[3]{%
  \DTLvarianceforcolumn{#1}{#2}{#3}%
  \DTLsqrt{#3}{#3}%
}
```

\DTLminforkeys

\DTLminforkeys[⟨*condition*⟩][⟨*assign list*⟩]{⟨*db list*⟩}{⟨*key list*⟩}{⟨*cmd*⟩}

Determines the minimum over all entries for each key in ⟨*key list*⟩ over all databases in ⟨*db list*⟩, and stores in ⟨*cmd*⟩, which must be a control sequence. The first optional argument ⟨*condition*⟩ is the same as that for \DTLforeach. The

second optional argument is an assignment list for \DTLforeach in the event that extra information is need for the condition.

```
\newcommand*{\DTLminforkeys}[1][\boolean{true}\and
 \DTLisnumerical{\DTLthisval}]{%
  \def\@dtl@cond{#1}%
  \@dtlminforkeys
}
```

\@dtlminforkeys

```
\newcommand*{\@dtlminforkeys}[4][]{%
  \def#4{}%
```

Iterate over all the listed data bases

```
  \@for\@dtl@dbname:=#2\do{%
```

Iterate through this database (using read only version)

```
    \@sDTLforeach{\@dtl@dbname}%
    {#1}% assignment list
    {%
```

Iterate through key list.

```
      \@for\@dtl@key:=#3\do{%
        \@sdtl@getcolumnindex{\@dtl@col}{\@dtl@dbname}{\@dtl@key}%
        \dtlcurrentrow=\expandafter{\dtl@thisrow}%
        \dtlgetentryfromrow{\DTLthisval}{\@dtl@col}{\dtlcurrentrow}%
        \expandafter\ifthenelse\expandafter{\@dtl@cond}%
        {%
          \ifx#4\@empty
            \let#4\DTLthisval
          \else
            \DTLmin{#4}{#4}{\DTLthisval}%
          \fi
        }{}%
      }%
    }%
  }%
}
```

\DTLminforcolumn

\DTLminforcolumn{⟨*db*⟩}{⟨*key*⟩}{⟨*cmd*⟩}

Quicker version of \DTLminforkeys that just finds the minimum value in one column (specified by ⟨*key*⟩) for a single database (specified by ⟨*db*⟩) and stores the result in ⟨*cmd*⟩.

```
\newcommand*{\DTLminforcolumn}[3]{%
  \def#3{}%
```

Check data base exists

```
  \DTLifdbexists{#1}%
  {%
```

Check column exists

```
    \@sDTLifhaskey{#1}{#2}%
    {%
      \@sdtlforcolumn{\DTLthisval}{#1}{#2}%
```

```
          {%
            \ifx#3\@empty
              \let#3\DTLthisval
            \else
              \DTLmin{#3}{#3}{\DTLthisval}%
            \fi
          }%
        }%
```

key not defined for this data base

```
        {%
          \PackageError{datatool}{Key ‘#2’ doesn’t
            exist in database ‘#1’}{}%
        }%
    }%
```

data base doesn't exist

```
    {%
      \PackageError{datatool}{Data base ‘#1’ doesn’t
        exist}{}%
    }%
  }
```

\DTLmaxforkeys

\DTLmaxforkeys[⟨*condition*⟩][⟨*assign list*⟩]{⟨*db list*⟩}{⟨*key list*⟩}{⟨*cmd*⟩}

Determines the maximum over all entries for each key in ⟨*key list*⟩ over all databases in ⟨*db list*⟩, and stores in ⟨*cmd*⟩, which must be a control sequence. The first optional argument ⟨*condition*⟩ is the same as that for \DTLforeach. The second optional argument is an assignment list to pass to \DTLforeach in the event that extra information is required in the condition.

```
\newcommand*{\DTLmaxforkeys}[1][\boolean{true}\and
 \DTLisnumerical{\DTLthisval}]{%
  \def\@dtl@cond{#1}%
  \@dtlmaxforkeys
}
```

\@dtlmaxforkeys

```
\newcommand*{\@dtlmaxforkeys}[4][]{%
  \def#4{}%
```

Iterate over all the listed data bases

```
  \@for\@dtl@dbname:=#2\do{%
```

Iterate through this database (using read only version)

```
    \@sDTLforeach{\@dtl@dbname}%
    {#1}% assignment list
    {%
```

Iterate through key list.

```
      \@for\@dtl@key:=#3\do{%
        \@sdtl@getcolumnindex{\@dtl@col}{\@dtl@dbname}{\@dtl@key}%
        \dtlcurrentrow=\expandafter{\dtl@thisrow}%
        \dtlgetentryfromrow{\DTLthisval}{\@dtl@col}{\dtlcurrentrow}%
        \expandafter\ifthenelse\expandafter{\@dtl@cond}%
```

```
          {%
            \ifx#4\@empty
              \let#4\DTLthisval
            \else
              \DTLmax{#4}{#4}{\DTLthisval}%
            \fi
          }{}%
        }%
      }%
    }%
  }
```

\DTLmaxforcolumn

\DTLmaxforcolumn{⟨*db*⟩}{⟨*key*⟩}{⟨*cmd*⟩}

Quicker version of \DTLmaxforkeys that just finds the maximum value in one column (specified by ⟨*key*⟩) for a single database (specified by ⟨*db*⟩) and stores the result in ⟨*cmd*⟩.

```
\newcommand*{\DTLmaxforcolumn}[3]{%
  \def#3{}%
```

Check data base exists

```
  \DTLifdbexists{#1}%
  {%
```

Check column exists

```
    \@sDTLifhaskey{#1}{#2}%
    {%
      \@sdtlforcolumn{\DTLthisval}{#1}{#2}%
      {%
        \ifx#3\@empty
          \let#3\DTLthisval
        \else
          \DTLmax{#3}{#3}{\DTLthisval}%
        \fi
      }%
    }%
```

key not defined for this data base

```
    {%
      \PackageError{datatool}{Key ‘#2’ doesn’t
        exist in database ‘#1’}{}%
    }%
  }%
```

data base doesn't exist

```
  {%
    \PackageError{datatool}{Data base ‘#1’ doesn’t
      exist}{}%
  }%
}
```

\DTLcomputebounds

\DTLcomputebounds[⟨*condition*⟩]{⟨*db list*⟩}{⟨*x key*⟩}{⟨*y key*⟩}{⟨*minX cmd*⟩}{⟨*minY cmd*⟩}{⟨*maxX cmd*⟩}{⟨*maxY cmd*⟩}

Computes the maximum and minimum x and y values over all the databases listed in ⟨*db list*⟩ where the x value is given by ⟨*x key*⟩ and the y value is given by ⟨*y key*⟩. The results are stored in ⟨*minX cmd*⟩, ⟨*minY cmd*⟩, ⟨*maxX cmd*⟩ and ⟨*maxY cmd*⟩ in standard decimal format.

```
\newcommand*{\DTLcomputebounds}[8][\boolean{true}]{%
\let#5=\relax
\let#6=\relax
\let#7=\relax
\let#8=\relax
\@for\dtl@thisdb:=#2\do{%
  \@sDTLforeach[#1]{\dtl@thisdb}{\DTLthisX=#3,\DTLthisY=#4}{%
    \DTLconverttodecimal{\DTLthisX}{\dtl@decx}%
    \DTLconverttodecimal{\DTLthisY}{\dtl@decy}%
    \ifx#5\relax
      \let#5=\dtl@decx
      \let#6=\dtl@decy
      \let#7=\dtl@decx
      \let#8=\dtl@decy
    \else
      \FPmin{#5}{#5}{\dtl@decx}%
      \FPmin{#6}{#6}{\dtl@decy}%
      \FPmax{#7}{#7}{\dtl@decx}%
      \FPmax{#8}{#8}{\dtl@decy}%
    \fi
  }%
}%
}
```

`\DTLgetvalueforkey`

`\DTLgetvalueforkey{`⟨*cmd*⟩`}{`⟨*key*⟩`}{`⟨*db name*⟩`}{`⟨*ref key*⟩`}{`⟨*ref value*⟩`}`

This (globally) sets ⟨*cmd*⟩ (a control sequence) to the value of the key specified by ⟨*key*⟩ in the first row of the database called ⟨*db name*⟩ which contains the key ⟨*ref key*⟩ which has the value ⟨*value*⟩.

```
\newcommand*{\DTLgetvalueforkey}[5]{%
```

Get row containing referenced (key,value) pair

```
  \DTLgetrowforkey{\@dtl@row}{#3}{#4}{#5}%
```

Get column number for ⟨*key*⟩

```
  \@sdtl@getcolumnindex{\@dtl@col}{#3}{#2}%
```

Get value for given column

```
  {%
    \dtlcurrentrow=\expandafter{\@dtl@row}%
    \edef\@dtl@dogetval{\noexpand\dtlgetentryfromcurrentrow
      {\noexpand\@dtl@val}{\@dtl@col}}%
    \@dtl@dogetval
    \global\let#1=\@dtl@val
  }%
}
```

`\DTLgetrowforkey`

`\DTLgetrowforkey{`⟨*cmd*⟩`}{`⟨*db name*⟩`}{`⟨*ref key*⟩`}{`⟨*ref value*⟩`}`

This (globally) sets ⟨*cmd*⟩ (a control sequence) to the first row of the database called ⟨*db name*⟩ which contains the key ⟨*ref key*⟩ that has the value ⟨*value*⟩.

```
\newcommand*{\DTLgetrowforkey}[4]{%
  \global\let#1=\@empty
  \@sDTLforeach{#2}{\dtl@refvalue=#3}{%
    \DTLifnull{\dtl@refvalue}%
    {}%
    {%
      \ifthenelse{\equal{\dtl@refvalue}{#4}}%
      {%
        \xdef#1{\the\dtlcurrentrow}%
        \dtlbreak
      }%
      {}%
    }%
  }%
}
```

`\dtlsplitrow` `\dtlsplitrow{`⟨*row specs*⟩`}{`⟨*col num*⟩`}{`⟨*before cs*⟩`}{`⟨*after cs*⟩`}`

Splits the row around the entry given by ⟨*col num*⟩. The entries before the split are stored in ⟨*before cs*⟩ and the entries after the split are stored in ⟨*after cs*⟩. ⟨*row specs*⟩ and ⟨*col num*⟩ need to be expanded before use.

```
\newcommand*{\dtlsplitrow}[4]{%
  \def\@dtlsplitrow##1%before stuff
    \db@col@id@w #2\db@col@id@end@% column id
      ##2% unwanted stuff
    \db@col@id@w #2\db@col@id@end@% column id
    ##3% after stuff
    \q@nil{\def#3{##1}\def#4{##3}}%
   \@dtlsplitrow#1\q@nil
}
```

10.11 Sorting Databases

`\@dtl@list` Token register to store data when sorting.

```
\newtoks\@dtl@list
```

`\DTLsort` `\DTLsort[`⟨*replacement keys*⟩`]{`⟨*sort criteria*⟩`}{`⟨*db name*⟩`}`

Sorts database ⟨*db name*⟩ according to `{`⟨*sort criteria*⟩`}`, which must be a comma separated list of keys, and optionally `=`⟨*order*⟩, where ⟨*order*⟩ is either `ascending` or `descending`. The optional argument is a list of keys to uses if the given key has a null value. The starred version uses a case insensitive string comparison.

```
\newcommand*{\DTLsort}{\@ifstar\@sDTLsort\@DTLsort}
```

`\@DTLsort` Unstarred (case sensitive) version.

```
\newcommand{\@DTLsort}[3][]{%
```

Check the database exists

```
    \DTLifdbexists{#3}%
    {%
```

Store replacement keys in `\@dtl@replacementkeys`.

```
      \edef\@dtl@replacementkeys{#1}%
```

Store sort order in `\@dtl@sortorder`.

```
      \edef\@dtl@sortorder{#2}%
```

Set `\@dtl@comparecs` to the required string comparison function. (Using case sensitive comparison macro `\dtlcompare`.)

```
      \let\@dtl@comparecs=\dtlcompare
```

Sort the database.

```
      \dtl@sortdata{#3}%
    }%
    {%
      \PackageError{datatool}{Database ‘#3’ doesn’t exist}{}%
    }%
  }
```

`\@sDTLsort` Starred (case insensitive) version.

```
  \newcommand{\@sDTLsort}[3][]{%
```

Check the database exists

```
    \DTLifdbexists{#3}%
    {%
```

Store replacement keys in `\@dtl@replacementkeys`.

```
      \edef\@dtl@replacementkeys{#1}%
```

Store sort order in `\@dtl@sortorder`.

```
      \edef\@dtl@sortorder{#2}%
```

Set `\@dtl@comparecs` to the required string comparison function. (Using case insensitive comparison macro `\dtlicompare`.)

```
      \let\@dtl@comparecs=\dtlicompare
```

Sort the database.

```
      \dtl@sortdata{#3}%
    }%
    {%
      \PackageError{datatool}{Database ‘#3’ doesn’t exist}{}%
    }%
  }
```

`\@dtl@rowa` Token register to store first row when sorting.

```
  \newtoks\@dtl@rowa
```

`\@dtl@rowb` Token register to store comparison row when sorting.

```
  \newtoks\@dtl@rowb
```

`\dtl@sortdata` `\dtl@sortdata{⟨db⟩}`

Sorts the data in named database using an insertion sort algorithm. `\@dtl@replacementkeys`, `\@dtl@sortorder` and `\@dtl@comparecs` must be set prior to use.

```
\newcommand*{\dtl@sortdata}[1]{%
```

Initialise macro containing sorted data.

```
  \def\@dtl@sortedlist{}%
```

Store database name.

```
  \edef\@dtl@dbname{#1}%
```

Iterate through each row and insert into sorted list.

```
  \@dtlforeachrow(\@dtl@rowAnum,\@dtl@rowAcontents)\in{#1}\do{%
    \@dtl@rowa=\expandafter{\@dtl@rowAcontents}%
```

Create a temporary list

```
    \def\@dtl@newlist{}%
```

Initialise the insertion for this iteration. Insertion hasn't been done yet.

```
  \@dtl@insertdonefalse
```

Initialise row index to 0

```
  \dtlrownum=0\relax
```

Iterate through sorted list.

```
  \expandafter\@dtl@foreachrow\@dtl@sortedlist
    \db@row@elt@w%
    \db@row@id@w \@nil\db@row@id@end@%
    \db@row@id@w \@nil\db@row@id@end@%
    \db@row@elt@end@%
    \@@{\@dtl@rowBnum}{\@dtl@rowBcontents}{%
```

Store row B in a token register

```
      \@dtl@rowb=\expandafter{\@dtl@rowBcontents}%
```

Get current row number of sorted list

```
      \dtlrownum=\@dtl@rowBnum
```

Has the insertion been done?

```
      \if@dtl@insertdone
```

New element has already been inserted, so just increment the row number to compensate for the inserted row.

```
        \advance\dtlrownum by 1\relax
      \else
```

Insertion hasn't been done yet. Compare row A and row B.

```
        \@dtl@sortcriteria{\@dtl@rowa}{\@dtl@rowb}%
```

If `\dtl@sortresult` is negative insert A before B.

```
        \ifnum\dtl@sortresult<0\relax
```

Insert row A into new list. First store `\@dtl@newlist` in `\toks@`.

```
          \toks@=\expandafter{\@dtl@newlist}%
```

Update `\@dtl@newlist` to be the old value followed by row A.

```
          \edef\@dtl@newlist{%
```

Old value:

```
            \the\toks@
```

Format row A

```
            \noexpand\db@row@elt@w%
             \noexpand\db@row@id@w \number\dtlrownum
             \noexpand\db@row@id@end@%
             \the\@dtl@rowa
             \noexpand\db@row@id@w \number\dtlrownum
             \noexpand\db@row@id@end@%
            \noexpand\db@row@elt@end@%
          }%
```

Increment row number to compensate for inserted row.

```
          \advance\dtlrownum by 1\relax
```

Mark insertion done.

```
          \@dtl@insertdonetrue
        \fi
      \fi
```

Insert row B

```
      \toks@=\expandafter{\@dtl@newlist}%
      \edef\@dtl@newlist{\the\toks@
```

row B

```
        \noexpand\db@row@elt@w%
         \noexpand\db@row@id@w \number\dtlrownum
         \noexpand\db@row@id@end@%
         \the\@dtl@rowb
         \noexpand\db@row@id@w \number\dtlrownum
         \noexpand\db@row@id@end@%
        \noexpand\db@row@elt@end@%
      }%
```

Repeat loop.

```
    }\q@nil
```

If row A hasn't been inserted, do so now.

```
    \if@dtl@insertdone
    \else
```

`\dtlrownum` contains the index of the last row in new list, So increment it to get the new index for row A.

```
      \advance\dtlrownum by 1\relax
```

Insert row A.

```
      \toks@=\expandafter{\@dtl@newlist}%
      \edef\@dtl@newlist{\the\toks@
```

row A

```
        \noexpand\db@row@elt@w%
         \noexpand\db@row@id@w \number\dtlrownum
         \noexpand\db@row@id@end@%
         \the\@dtl@rowa
         \noexpand\db@row@id@w \number\dtlrownum
         \noexpand\db@row@id@end@%
        \noexpand\db@row@elt@end@%
      }%
    \fi
```

Set sorted list to new list.

```
      \let\@dtl@sortedlist=\@dtl@newlist
    }%
```

Update database.

```
    \expandafter\global\csname dtldb@#1\endcsname=\expandafter
      {\@dtl@sortedlist}%
  }
```

\@dtl@sortcriteria

\@dtl@sortcriteria{⟨*row a toks*⟩}{⟨*row b toks*⟩}

\@dtl@dbname and \@dtl@sortorder must be set before use \@dtl@sortorder is a comma separated list of either just keys or ⟨*key*⟩=⟨*direction*⟩. (Check keys are valid before use.)

```
  \newcommand{\@dtl@sortcriteria}[2]{%
```

Iterate through the sort order.

```
    \@for\@dtl@level:=\@dtl@sortorder\do{%
```

Set \@dtl@sortdirection to −1 (ascending) or +1 (descending). Key is stored in \@dtl@key.

```
      \expandafter\@dtl@getsortdirection\@dtl@level=\relax
```

Initially comparing on the same key

```
      \let\@dtl@keya=\@dtl@key
      \let\@dtl@keyb=\@dtl@key
```

Get values corresponding to key from both rows. First get column index corresponding to key.

```
      \@sdtl@getcolumnindex{\@dtl@col}{\@dtl@dbname}{\@dtl@key}%
```

Get entry for this column from row A and store in \@dtl@a.

```
      \dtlgetentryfromrow{\@dtl@a}{\@dtl@col}{#1}%
```

Get entry for this column from row B and store in \@dtl@b.

```
      \dtlgetentryfromrow{\@dtl@b}{\@dtl@col}{#2}%
```

Has value from row A been defined?

```
      \ifx\@dtl@a\dtlnovalue
```

Value hasn't been defined so set to null

```
        \@dtl@setnull{\@dtl@a}{\@dtl@key}%
      \fi
```

Has value from row B been defined?

```
      \ifx\@dtl@b\dtlnovalue
```

Value hasn't been defined so set to null

```
        \@dtl@setnull{\@dtl@b}{\@dtl@key}%
      \fi
```

Check if value for row A is null.

```
      \DTLifnull{\@dtl@a}%
      {%
```

Value for row A is null, so find the first non null key in list of replacement keys.

```
        \@for\@dtl@keya:=\@dtl@replacementkeys\do{%
```

Get column corresponding to this key.

```
        \@sdtl@getcolumnindex{\@dtl@col}{\@dtl@dbname}{\@dtl@keya}%
        \dtlgetentryfromrow{\@dtl@a}{\@dtl@col}{#1}%
```

Has value for row A been defined?

```
        \ifx\@dtl@a\dtlnovalue
```

Value for row A hasn't been defined so set to null

```
          \@dtl@setnull{\@dtl@a}{\@dtl@key}%
        \fi
```

Is value for row A null? If not null end the loop.

```
        \DTLifnull{\@dtl@a}{}{\@endfortrue}%
      }%
```

No non-null value found.

```
      \ifx\@dtl@keya\@nnil
        \let\@dtl@keya\@dtl@key
        \@dtl@setnull{\@dtl@a}{\@dtl@key}%
      \fi
    }%
    {}%
```

Check if value for row B is null.

```
    \DTLifnull{\@dtl@b}%
    {%
```

Value for row B is null, so find the first non null key in list of replacement keys.

```
      \@for\@dtl@keyb:=\@dtl@replacementkeys\do{%
```

Get column corresponding to this key.

```
        \@sdtl@getcolumnindex{\@dtl@col}{\@dtl@dbname}{\@dtl@keyb}%
        \dtlgetentryfromrow{\@dtl@b}{\@dtl@col}{#2}%
```

Has value for row B been defined?

```
        \ifx\@dtl@b\dtlnovalue
```

Value for row B hasn't been defined so set to null.

```
          \@dtl@setnull{\@dtl@b}{\@dtl@key}%
        \fi
```

Is value for row B null? If not null end the loop.

```
        \DTLifnull{\@dtl@b}{}{\@endfortrue}%
      }%
```

No non-null value found.

```
      \ifx\@dtl@keyb\@nnil
        \let\@dtl@keyb\@dtl@key
        \@dtl@setnull{\@dtl@b}{\@dtl@key}%
      \fi
    }%
    {}%
```

Compare rows A and B. First store the values for row A and B in token registers so that they can be passed to `\dtl@compare@`.

```
    \@dtl@toksA=\expandafter{\@dtl@a}%
    \@dtl@toksB=\expandafter{\@dtl@b}%
```

Do comparison.

```
      \edef\@dtl@docompare{\noexpand\dtl@compare@
        {\@dtl@keya}{\@dtl@keyb}%
        {\noexpand\@dtl@toksA}{\noexpand\@dtl@toksB}}%
      \@dtl@docompare
```

Repeat if the two values are considered identical and there are further sorting options.

```
      \ifnum\dtl@sortresult=0\relax
```

Reset switch to prevent breaking out of outer loop.

```
        \@endforfalse
      \else
```

Break out of loop.

```
        \@endfortrue
      \fi
    }%
```

Apply sort direction

```
    \multiply\dtl@sortresult by -\@dtl@sortdirection\relax
  }
```

`\@dtl@getsortdirection` Get the direction from either ⟨*key*⟩ or ⟨*key*⟩=⟨*direction*⟩. Sets `\@dtl@sortdirection` to either -1 (ascending) or 1 (descending).

```
  \def\@dtl@getsortdirection#1=#2\relax{%
```

Store key in `\@dtl@key`.

```
    \def\@dtl@key{#1}%
```

Store sort direction. This will be empty if no direction was specified.

```
    \def\@dtl@sortdirection{#2}%
```

Check if a direction was specified.

```
    \ifx\@dtl@sortdirection\@empty
```

No direction specified so assume ascending.

```
      \def\@dtl@sortdirection{-1}%
    \else
```

Get the sort direction from the second argument (needs terminating equal sign removed) and store in `\@dtl@sortdirection`.

```
      \@dtl@get@sortdirection#2%
```

Determine the direction.

```
      \def\@dtl@dir{ascending}%
      \ifx\@dtl@sortdirection\@dtl@dir
```

Ascending

```
        \def\@dtl@sortdirection{-1}%
      \else
```

Check if descending.

```
        \def\@dtl@dir{descending}%
        \ifx\@dtl@sortdirection\@dtl@dir
```

Descending

```
          \def\@dtl@sortdirection{1}%
        \else
```

Direction not valid. Generate error message.

```
          \PackageError{datatool}{Invalid sort direction
          `\@dtl@sortdirection'}{The sort direction can only be
           one of `ascending' or `descending'}%
```

Assume ascending.

```
          \def\@dtl@sortdirection{-1}
        \fi
      \fi
    \fi
  }
```

`\@dtl@get@sortdirection` Get direction (trims trailing = sign)

```
\def\@dtl@get@sortdirection#1={\def\@dtl@sortdirection{#1}}
```

`\@dtl@toksA`

```
\newtoks\@dtl@toksA
```

`\@dtl@toksB`

```
\newtoks\@dtl@toksB
%   \end{macrocode}
%\end{macro}
%\begin{macro}{\@dtl@toks}
%    \begin{macrocode}
\newtoks\@dtl@toks
```

`\dtl@compare` `\dtl@compare{⟨key⟩}{⟨a toks⟩}{⟨b toks⟩}`

Compares two values according to ⟨*key*⟩ of database given by `\@dtl@dbname`. Sets `\dtl@sortresult`. `\@dtl@comparecs` must be set to the required comparison macro.

```
\newcommand{\dtl@compare}[3]{%
  \dtl@compare@{#1}{#1}{#2}{#3}%
}
```

`\dtl@compare@` `\dtl@compare@{⟨keyA⟩}{⟨keyB⟩}{⟨A toks⟩}{⟨B toks⟩}`

Compare ⟨*A*⟩ and ⟨*B*⟩ according ⟨*keyA*⟩ and ⟨*keyB*⟩ for database given by `\@dtl@dbname`. Sets `\dtl@sortresult`. `\@dtl@comparecs` must be set before use.

```
\newcommand{\dtl@compare@}[4]{%
```

Get the data type for first key and store in `\@dtl@typeA`.

```
  \DTLgetdatatype{\@dtl@typeA}{\@dtl@dbname}{#1}%
```

Is it unset? If so, assume string

```
  \ifx\@dtl@typeA\DTLunsettype
    \let\@dtl@typeA\DTLstringtype
  \fi
```

Get the data type for the second key and store in `\@dtl@typeB`

```
\DTLgetdatatype{\@dtl@typeB}{\@dtl@dbname}{#2}%
```

Is it unset? If so, assume string

```
\ifx\@dtl@typeB\DTLunsettype
  \let\@dtl@typeB\DTLstringtype
\fi
```

Multiply the two values together

```
\@dtl@tmpcount=\@dtl@typeA\relax
\multiply\@dtl@tmpcount by \@dtl@typeB\relax
```

If either type is 0 (a string) then the product will also be 0 (string) otherwise it will be one of the numerical types.

```
\ifnum\@dtl@tmpcount=0\relax
```

A string, so use comparison function

```
    \edef\@dtl@tmpcmp{%
      \noexpand\@dtl@comparecs{\noexpand\dtl@sortresult}%
        {\the#3}{\the#4}%
      }%
    \@dtl@tmpcmp
    \ifdtlverbose
      \edef\@dtl@a{\the#3}%
      \edef\@dtl@b{\the#4}%
    \fi
  \else
```

Store the first value

```
    \edef\@dtl@a{\the#3}%
```

Store the second value

```
    \edef\@dtl@b{\the#4}%
```

Compare

```
    \DTLifnumlt{\@dtl@a}{\@dtl@b}%
    {%
```

$A < B$

```
      \dtl@sortresult=-1\relax
    }%
    {%
      \DTLifnumgt{\@dtl@a}{\@dtl@b}%
      {%
```

$A > B$

```
        \dtl@sortresult=1\relax
      }%
      {%
```

$A = B$

```
        \dtl@sortresult=0\relax
      }%
    }%
  \fi
```

Write comparison result to terminal/log if verbose mode.

```
    \ifdtlverbose
      \@onelevel@sanitize\@dtl@a
      \@onelevel@sanitize\@dtl@b
      \dtl@message{`\@dtl@a' <=> `\@dtl@b' = \number\dtl@sortresult}%
    \fi
  }
```

10.12 General List Utilities

\dtl@choplast

\dtl@choplast{⟨*list*⟩}{⟨*rest*⟩}{⟨*last*⟩}

Chops the last element off a comma separated list, putting the last element in the control sequence ⟨*last*⟩ and putting the rest in the control sequence ⟨*rest*⟩. The control sequence ⟨*list*⟩ is unchanged. If the list is empty, both ⟨*last*⟩ and ⟨*rest*⟩ will be empty.

```
\newcommand*{\dtl@choplast}[3]{%
```

Set ⟨*rest*⟩ to empty:

```
\let#2\@empty
```

Set ⟨*last*⟩ to empty:

```
\let#3\@empty
```

Iterate through ⟨*list*⟩:

```
\@for\@dtl@element:=#1\do{%
\ifx#3\@empty
```

First iteration, don't set ⟨*rest*⟩.

```
\else
 \ifx#2\@empty
```

Second iteration, set ⟨*rest*⟩ to ⟨*last*⟩ (which is currently set to the previous value:

```
    \expandafter\toks@\expandafter{#3}%
    \edef#2{{\the\toks@}}%
  \else
```

Subsequent iterations, set ⟨*rest*⟩ to ⟨*rest*⟩,⟨*last*⟩ (⟨*last*⟩ is currently set to the previous value):

```
    \expandafter\toks@\expandafter{#3}%
    \expandafter\@dtl@toks\expandafter{#2}%
    \edef#2{\the\@dtl@toks,{\the\toks@}}%
  \fi
\fi
```

Now set ⟨*last*⟩ to current element.

```
\let#3=\@dtl@element%
}%
}
```

\dtl@chopfirst

\dtl@chopfirst{⟨*list*⟩}{⟨*first*⟩}{⟨*rest*⟩}

Chops first element off ⟨*list*⟩ and store in ⟨*first*⟩. The remainder of the list is stored in ⟨*rest*⟩. (⟨*list*⟩ remains unchanged.)

```
\newcommand*{\dtl@chopfirst}[3]{%
\let#2=\@empty
\let#3=\@empty
\@for\@dtl@element:=#1\do{%
 \let#2=\@dtl@element
 \@endfortrue
}%
\if@endfor
 \let#3=\@forremainder
\fi
\@endforfalse
}
```

`\dtl@sortlist` `\dtl@sortlist{⟨list⟩}{⟨criteria cmd⟩}`

Performs an insertion sort on ⟨*list*⟩, where ⟨*criteria cmd*⟩ is a macro which takes two arguments ⟨*a*⟩ and ⟨*b*⟩. ⟨*criteria cmd*⟩ must set the count register `\dtl@sortresult` to either -1 (⟨*a*⟩ less than ⟨*b*⟩), 0 (⟨*a*⟩ is equal to ⟨*b*⟩) or 1 (⟨*a*⟩ is greater than ⟨*b*⟩.)

```
\newcommand{\dtl@sortlist}[2]{%
\def\@dtl@sortedlist{}%
\@for\@dtl@currentrow:=#1\do{%
\expandafter\dtl@insertinto\expandafter
  {\@dtl@currentrow}{\@dtl@sortedlist}{#2}%
\@endforfalse}%
\let#1=\@dtl@sortedlist
}
```

`\dtl@insertinto` `\dtl@insertinto{⟨element⟩}{⟨sorted-list⟩}{⟨criteria cmd⟩}`

Inserts ⟨*element*⟩ into the sorted list ⟨*sorted-list*⟩ according to the criteria given by ⟨*criteria cmd*⟩ (see above.)

```
\newcommand{\dtl@insertinto}[3]{%
\def\@dtl@newsortedlist{}%
\@dtl@insertdonefalse
\@for\dtl@srtelement:=#2\do{%
\if@dtl@insertdone
  \expandafter\toks@\expandafter{\dtl@srtelement}%
  \edef\@dtl@newstuff{{\the\toks@}}%
\else
  \expandafter#3\expandafter{\dtl@srtelement}{#1}%
  \ifnum\dtl@sortresult<0\relax
    \expandafter\toks@\expandafter{\dtl@srtelement}%
    \@dtl@toks{#1}%
    \edef\@dtl@newstuff{{\the\@dtl@toks},{\the\toks@}}%
    \@dtl@insertdonetrue
  \else
    \expandafter\toks@\expandafter{\dtl@srtelement}%
```

```
    \edef\@dtl@newstuff{{\the\toks@}}%
  \fi
\fi
\ifx\@dtl@newsortedlist\@empty
  \expandafter\toks@\expandafter{\@dtl@newstuff}%
  \edef\@dtl@newsortedlist{\the\toks@}%
\else
  \expandafter\toks@\expandafter{\@dtl@newsortedlist}%
  \expandafter\@dtl@toks\expandafter{\@dtl@newstuff}%
  \edef\@dtl@newsortedlist{\the\toks@,\the\@dtl@toks}%
\fi
\@endforfalse
}%
 \ifx\@dtl@newsortedlist\@empty
   \@dtl@toks{#1}%
   \edef\@dtl@newsortedlist{{\the\@dtl@toks}}%
 \else
   \if@dtl@insertdone
   \else
     \expandafter\toks@\expandafter{\@dtl@newsortedlist}%
     \@dtl@toks{#1}%
     \edef\@dtl@newsortedlist{\the\toks@,{\the\@dtl@toks}}%
   \fi
 \fi
\global\let#2=\@dtl@newsortedlist
}
```

`\if@dtl@insertdone` Define conditional to indicate whether the new entry has been inserted into the sorted list.

```
\newif\if@dtl@insertdone
```

`\dtl@sortresult` Define `\dtl@sortresult` to be set by comparision macro.

```
\newcount\dtl@sortresult
```

10.13 General Token Utilities

`\toks@gput@right@cx`

`\toks@gput@right@cx{`⟨*toks name*⟩`}{`⟨*stuff*⟩`}`

Globally appends stuff to token register \⟨*toks name*⟩

```
\newcommand{\toks@gput@right@cx}[2]{%
  \def\@toks@name{#1}%
  \edef\@dtl@stuff{#2}%
  \global\csname\@toks@name\endcsname\expandafter
    \expandafter\expandafter{\expandafter\the
    \csname\expandafter\@toks@name\expandafter\endcsname\@dtl@stuff}%
}
```

`\toks@gconcat@middle@cx`

`\toks@gconcat@middle@cx{`⟨*toks name*⟩`}{`⟨*before toks*⟩`}{`⟨*stuff*⟩`}{`⟨*after toks*⟩`}`

Globally sets token register \⟨*toks name*⟩ to the contents of ⟨*before toks*⟩ concatenated with ⟨*stuff*⟩ (expanded) and the contents of ⟨*after toks*⟩

```
\newcommand{\toks@gconcat@middle@cx}[4]{%
  \def\@toks@name{#1}%
  \edef\@dtl@stuff{#3}%
  \global\csname\@toks@name\endcsname\expandafter\expandafter
    \expandafter\expandafter\expandafter
    \expandafter\expandafter{\expandafter\expandafter\expandafter
    \the\expandafter\expandafter\expandafter#2%
    \expandafter\@dtl@stuff\the#4}%
}
```

10.14 Floating Point Arithmetic

The commands defined in this section all use the equivalent commands provided by the fp package, but first convert the decimal number into the required format.

\DTLadd

\DTLadd{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

Sets ⟨*cmd*⟩ = ⟨*num1*⟩ + ⟨*num2*⟩

```
\newcommand*{\DTLadd}[3]{%
\DTLconverttodecimal{#2}{\@dtl@numi}%
\DTLconverttodecimal{#3}{\@dtl@numii}%
\FPadd{\@dtl@tmp}{\@dtl@numi}{\@dtl@numii}%
\ifx\@dtl@replaced\@empty
  \DTLdecimaltolocale{\@dtl@tmp}{#1}%
\else
  \DTLdecimaltocurrency{\@dtl@tmp}{#1}%
\fi
}
```

\DTLgadd

Global version

```
\newcommand*{\DTLgadd}[3]{%
\DTLadd{\@dtl@tmpii}{#2}{#3}%
\global\let#1=\@dtl@tmpii
}
```

\DTLaddall

\DTLaddall{⟨*cmd*⟩}{⟨*num list*⟩}

Sums all the values in ⟨*num list*⟩ and stores in ⟨*cmd*⟩ which must be a control sequence.

```
\newcommand*{\DTLaddall}[2]{%
\def\@dtl@sum{0}%
\@for\dtl@thisval:=#2\do{%
  \DTLconverttodecimal{\dtl@thisval}{\@dtl@num}%
  \FPadd{\@dtl@sum}{\@dtl@sum}{\@dtl@num}%
}%
\ifx\@dtl@replaced\@empty
  \DTLdecimaltolocale{\@dtl@sum}{#1}%
```

```
\else
  \DTLdecimaltocurrency{\@dtl@sum}{#1}%
\fi
}
```

\DTLgaddall

\DTLgaddall{⟨*cmd*⟩}{⟨*num list*⟩}

Global version

```
\newcommand*{\DTLgaddall}[2]{%
\DTLaddall{\@dtl@tmpi}{#2}%
\global\let#1=\@dtl@tmpi
}
```

\DTLsub

\DTLsub{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

Sets ⟨*cmd*⟩ = ⟨*num1*⟩ - ⟨*num2*⟩

```
\newcommand*{\DTLsub}[3]{%
\DTLconverttodecimal{#2}{\@dtl@numi}%
\DTLconverttodecimal{#3}{\@dtl@numii}%
\FPsub{\@dtl@tmp}{\@dtl@numi}{\@dtl@numii}%
\ifx\@dtl@replaced\@empty
  \DTLdecimaltolocale{\@dtl@tmp}{#1}%
\else
  \DTLdecimaltocurrency{\@dtl@tmp}{#1}%
\fi
}
```

\DTLgsub Global version

```
\newcommand*{\DTLgsub}[3]{%
\DTLsub{\@dtl@tmpii}{#2}{#3}%
\global\let#1=\@dtl@tmpii
}
```

\DTLmul

\DTLmul{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

Sets ⟨*cmd*⟩ = ⟨*num1*⟩ × ⟨*num2*⟩

```
\newcommand*{\DTLmul}[3]{%
\let\@dtl@thisreplaced=\@empty
\DTLconverttodecimal{#2}{\@dtl@numi}%
\ifx\@dtl@replaced\@empty
\else
  \let\@dtl@thisreplaced=\@dtl@replaced
\fi
\DTLconverttodecimal{#3}{\@dtl@numii}%
\ifx\@dtl@replaced\@empty
\else
  \let\@dtl@thisreplaced=\@dtl@replaced
\fi
\FPmul{\@dtl@tmp}{\@dtl@numi}{\@dtl@numii}%
```

```
\ifx\@dtl@thisreplaced\@empty
  \DTLdecimaltolocale{\@dtl@tmp}{#1}%
\else
  \DTLdecimaltocurrency{\@dtl@tmp}{#1}%
\fi
}
```

\DTLgmul Global version

```
\newcommand*{\DTLgmul}[3]{%
\DTLmul{\@dtl@tmpii}{#2}{#3}%
\global\let#1=\@dtl@tmpii
}
```

\DTLdiv **\DTLdiv{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}**

Sets ⟨*cmd*⟩ = ⟨*num1*⟩ / ⟨*num2*⟩

```
\newcommand*{\DTLdiv}[3]{%
\let\@dtl@thisreplaced=\@empty
\DTLconverttodecimal{#2}{\@dtl@numi}%
\ifx\@dtl@replaced\@empty
\else
  \let\@dtl@thisreplaced=\@dtl@replaced
\fi
\DTLconverttodecimal{#3}{\@dtl@numii}%
\FPdiv{\@dtl@tmp}{\@dtl@numi}{\@dtl@numii}%
\ifx\@dtl@thisreplaced\@empty
  \DTLdecimaltolocale{\@dtl@tmp}{#1}%
\else
  \ifx\@dtl@thisreplaced\@dtl@replaced
    \DTLdecimaltolocale{\@dtl@tmp}{#1}%
  \else
    \DTLdecimaltocurrency{\@dtl@tmp}{#1}%
  \fi
\fi
}
```

\DTLgdiv Global version

```
\newcommand*{\DTLgdiv}[3]{%
\DTLdiv{\@dtl@tmpii}{#2}{#3}%
\global\let#1=\@dtl@tmpii
}
```

\DTLabs **\DTLabs{⟨*cmd*⟩}{⟨*num*⟩}**

Sets ⟨*cmd*⟩ = abs(⟨*num*⟩)

```
\newcommand*{\DTLabs}[2]{%
\DTLconverttodecimal{#2}{\@dtl@numi}%
\FPabs{\@dtl@tmp}{\@dtl@numi}%
\ifx\@dtl@replaced\@empty
  \DTLdecimaltolocale{\@dtl@tmp}{#1}%
```

```
\else
  \DTLdecimaltocurrency{\@dtl@tmp}{#1}%
\fi
}
```

\DTLgabs Global version

```
\newcommand*{\DTLgabs}[2]{%
\DTLabs{\@dtl@tmpii}{#2}%
\global\let#1=\@dtl@tmpii
}
```

\DTLneg **\DTLneg{⟨*cmd*⟩}{⟨*num*⟩}**

Sets ⟨*cmd*⟩ = -⟨*num*⟩

```
\newcommand*{\DTLneg}[2]{%
\DTLconverttodecimal{#2}{\@dtl@numi}%
\FPneg{\@dtl@tmp}{\@dtl@numi}%
\ifx\@dtl@replaced\@empty
  \DTLdecimaltolocale{\@dtl@tmp}{#1}%
\else
  \DTLdecimaltocurrency{\@dtl@tmp}{#1}%
\fi
}
```

\DTLgneg Global version

```
\newcommand*{\DTLgneg}[2]{%
\DTLneg{\@dtl@tmpii}{#2}%
\global\let#1=\@dtl@tmpii
}
```

\DTLsqrt **\DTLsqrt{⟨*cmd*⟩}{⟨*num*⟩}**

Sets ⟨*cmd*⟩ = sqrt(⟨*num*⟩)

```
\newcommand*{\DTLsqrt}[2]{%
\DTLconverttodecimal{#2}{\@dtl@numi}%
\FProot{\@dtl@tmpi}{\@dtl@numi}{2}%
\ifx\@dtl@replaced\@empty
  \DTLdecimaltolocale{\@dtl@tmpi}{#1}%
\else
  \DTLdecimaltocurrency{\@dtl@tmpi}{#1}%
\fi
}
```

\DTLgsqrt Global version

```
\newcommand*{\DTLgsqrt}[2]{%
\DTLsqrt{\@dtl@tmpii}{#2}%
\global\let#1=\@dtl@tmpii
}
```

\DTLmin

\DTLmin{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

Sets ⟨*cmd*⟩ = min(⟨*num1*⟩, ⟨*num2*⟩)

```
\newcommand*{\DTLmin}[3]{%
\DTLconverttodecimal{#2}{\@dtl@numi}%
\DTLconverttodecimal{#3}{\@dtl@numii}%
\FPiflt{\@dtl@numi}{\@dtl@numii}%
 \dtl@ifsingle{#2}{%
  \let#1=#2}{%
  \def#1{#2}}%
\else
 \dtl@ifsingle{#3}{%
  \let#1=#3}{%
  \def#1{#3}}%
\fi
}
```

\DTLgmin Global version

```
\newcommand*{\DTLgmin}[3]{%
\DTLmin{\@dtl@tmpii}{#2}{#3}%
\global\let#1=\@dtl@tmpii
}
```

\DTLminall

\DTLminall{⟨*cmd*⟩}{⟨*num list*⟩}

Finds the minimum value in ⟨*num list*⟩ and stores in ⟨*cmd*⟩ which must be a control sequence.

```
\newcommand*{\DTLminall}[2]{%
\let\@dtl@min=\@empty
\@for\dtl@thisval:=#2\do{%
  \DTLconverttodecimal{\dtl@thisval}{\@dtl@num}%
  \ifx\@dtl@min\@empty
    \let\@dtl@min=\@dtl@num
  \else
    \FPmin{\@dtl@min}{\@dtl@min}{\@dtl@num}%
  \fi
}%
\ifx\@dtl@replaced\@empty
  \DTLdecimaltolocale{\@dtl@min}{#1}%
\else
  \DTLdecimaltocurrency{\@dtl@min}{#1}%
\fi
}
```

\DTLgminall

\DTLgminall{⟨*cmd*⟩}{⟨*num list*⟩}

Global version

```
\newcommand*{\DTLgminall}[2]{%
\DTLminall{\@dtl@tmpi}{#2}%
```

```
\global\let#1=\@dtl@tmpi
}
```

\DTLmax

\DTLmax{⟨*cmd*⟩}{⟨*num1*⟩}{⟨*num2*⟩}

Sets ⟨*cmd*⟩ = max(⟨*num1*⟩, ⟨*num2*⟩)

```
\newcommand*{\DTLmax}[3]{%
\DTLconverttodecimal{#2}{\@dtl@numi}%
\DTLconverttodecimal{#3}{\@dtl@numii}%
\FPmax{\@dtl@tmp}{\@dtl@numi}{\@dtl@numii}%
\FPifgt{\@dtl@numi}{\@dtl@numii}%
 \dtl@ifsingle{#2}{%
  \let#1=#2}{%
  \def#1{#2}}%
\else
 \dtl@ifsingle{#3}{%
  \let#1=#3}{%
  \def#1{#3}}%
\fi
}
```

\DTLgmax Global version

```
\newcommand*{\DTLgmax}[3]{%
\DTLmax{\@dtl@tmpii}{#2}{#3}%
\global\let#1=\@dtl@tmpii
}
```

\DTLmaxall

\DTLmaxall{⟨*cmd*⟩}{⟨*num list*⟩}

Finds the maximum value in ⟨*num list*⟩ and stores in ⟨*cmd*⟩ which must be a control sequence.

```
\newcommand*{\DTLmaxall}[2]{%
\let\@dtl@max=\@empty
\@for\dtl@thisval:=#2\do{%
  \DTLconverttodecimal{\dtl@thisval}{\@dtl@num}%
  \ifx\@dtl@max\@empty
    \let\@dtl@max\@dtl@num
  \else
    \FPmax{\@dtl@max}{\@dtl@max}{\@dtl@num}%
  \fi
}%
\ifx\@dtl@replaced\@empty
  \DTLdecimaltolocale{\@dtl@max}{#1}%
\else
  \DTLdecimaltocurrency{\@dtl@max}{#1}%
\fi
}
```

\DTLgmaxall

\DTLgmaxall{⟨*cmd*⟩}{⟨*num list*⟩}

Global version

```
\newcommand*{\DTLgmaxall}[2]{%
\DTLmaxall{\@dtl@tmpi}{#2}%
\global\let#1=\@dtl@tmpi
}
```

\DTLmeanforall

\DTLmeanforall{⟨*cmd*⟩}{⟨*num list*⟩}

Computes the arithmetic mean of all the values in ⟨*num list*⟩ and stores in ⟨*cmd*⟩ which must be a control sequence.

```
\newcommand*{\DTLmeanforall}[2]{%
\def\@dtl@mean{0}%
\def\@dtl@n{0}%
\@for\dtl@thisval:=#2\do{%
  \DTLconverttodecimal{\dtl@thisval}{\@dtl@num}%
  \FPadd{\@dtl@mean}{\@dtl@mean}{\@dtl@num}%
  \FPadd{\@dtl@n}{\@dtl@n}{1}%
}%
\FPdiv{\@dtl@mean}{\@dtl@mean}{\@dtl@n}%
\ifx\@dtl@replaced\@empty
  \DTLdecimaltolocale{\@dtl@mean}{#1}%
\else
  \DTLdecimaltocurrency{\@dtl@mean}{#1}%
\fi
}
```

\DTLgmeanforall

\DTLgmeanforall{⟨*cmd*⟩}{⟨*num list*⟩}

Global version

```
\newcommand*{\DTLgmeanforall}[2]{%
\DTLmeanforall{\@dtl@tmpi}{#2}%
\global\let#1=\@dtl@tmpi
}
```

\DTLvarianceforall

\DTLvarianceforall{⟨*cmd*⟩}{⟨*num list*⟩}

Computes the variance of all the values in ⟨*num list*⟩ and stores in ⟨*cmd*⟩ which must be a control sequence.

```
\newcommand*{\DTLvarianceforall}[2]{%
\def\@dtl@mean{0}%
\def\@dtl@n{0}%
\let\@dtl@decvals=\@empty
\@for\dtl@thisval:=#2\do{%
  \DTLconverttodecimal{\dtl@thisval}{\@dtl@num}%
  \ifx\@dtl@decvals\@empty
   \let\@dtl@decvals=\@dtl@num
  \else
   \expandafter\toks@\expandafter{\@dtl@decvals}%
```

```
    \edef\@dtl@decvals{\the\toks@,\@dtl@num}%
   \fi
   \FPadd{\@dtl@mean}{\@dtl@mean}{\@dtl@num}%
   \FPadd{\@dtl@n}{\@dtl@n}{1}%
}%
\FPdiv{\@dtl@mean}{\@dtl@mean}{\@dtl@n}%
\def\@dtl@var{0}%
\@for\@dtl@num:=\@dtl@decvals\do{%
   \FPsub{\@dtl@diff}{\@dtl@num}{\@dtl@mean}%
   \FPmul{\@dtl@diff}{\@dtl@diff}{\@dtl@diff}%
   \FPadd{\@dtl@var}{\@dtl@var}{\@dtl@diff}%
}%
\FPdiv{\@dtl@var}{\@dtl@var}{\@dtl@n}%
\ifx\@dtl@replaced\@empty
   \DTLdecimaltolocale{\@dtl@var}{#1}%
\else
   \DTLdecimaltocurrency{\@dtl@var}{#1}%
\fi
}
```

\DTLgvarianceforall

\DTLgvarianceforall{⟨*cmd*⟩}{⟨*num list*⟩}

Global version

```
\newcommand*{\DTLgvarianceforall}[2]{%
\DTLvarianceforall{\@dtl@tmpi}{#2}%
\global\let#1=\@dtl@tmpi
}
```

\DTLsdforall

\DTLsdforall{⟨*cmd*⟩}{⟨*num list*⟩}

Computes the standard deviation of all the values in ⟨*num list*⟩ and stores in ⟨*cmd*⟩ which must be a control sequence.

```
\newcommand*{\DTLsdforall}[2]{%
\def\@dtl@mean{0}%
\def\@dtl@n{0}%
\let\@dtl@decvals=\@empty
\@for\dtl@thisval:=#2\do{%
   \DTLconverttodecimal{\dtl@thisval}{\@dtl@num}%
   \ifx\@dtl@decvals\@empty
    \let\@dtl@decvals=\@dtl@num
   \else
    \expandafter\toks@\expandafter{\@dtl@decvals}%
    \edef\@dtl@decvals{\the\toks@,\@dtl@num}%
   \fi
   \FPadd{\@dtl@mean}{\@dtl@mean}{\@dtl@num}%
   \FPadd{\@dtl@n}{\@dtl@n}{1}%
}%
\FPdiv{\@dtl@mean}{\@dtl@mean}{\@dtl@n}%
\def\@dtl@sd{0}%
\@for\@dtl@num:=\@dtl@decvals\do{%
   \FPsub{\@dtl@diff}{\@dtl@num}{\@dtl@mean}%
```

```
  \FPmul{\@dtl@diff}{\@dtl@diff}{\@dtl@diff}%
  \FPadd{\@dtl@sd}{\@dtl@sd}{\@dtl@diff}%
}%
\FPdiv{\@dtl@sd}{\@dtl@sd}{\@dtl@n}%
\FProot{\@dtl@sd}{\@dtl@sd}{2}%
\ifx\@dtl@replaced\@empty
  \DTLdecimaltolocale{\@dtl@sd}{#1}%
\else
  \DTLdecimaltocurrency{\@dtl@sd}{#1}%
\fi
}
```

\DTLgsdforall

\DTLgsdforall{⟨*cmd*⟩}{⟨*num list*⟩}

Global version

```
\newcommand*{\DTLgsdforall}[2]{%
\DTLsdforall{\@dtl@tmpi}{#2}%
\global\let#1=\@dtl@tmpi
}
```

\DTLround

\DTLround{⟨*cmd*⟩}{⟨*num*⟩}{⟨*num digits*⟩}

Sets ⟨*cmd*⟩ to ⟨*num*⟩ rounded to ⟨*num digits*⟩ digits after the decimal character.

```
\newcommand*{\DTLround}[3]{%
\DTLconverttodecimal{#2}{\@dtl@numi}%
\FPround{\@dtl@tmp}{\@dtl@numi}{#3}%
\ifx\@dtl@replaced\@empty
  \DTLdecimaltolocale{\@dtl@tmp}{#1}%
\else
  \DTLdecimaltocurrency{\@dtl@tmp}{#1}%
\fi
}
```

\DTLground

Global version

```
\newcommand*{\DTLground}[3]{%
\DTLround{\@dtl@tmpii}{#2}{#3}%
\global\let#1=\@dtl@tmpii
}
```

\DTLtrunc

\DTLtrunc{⟨*cmd*⟩}{⟨*num*⟩}{⟨*num digits*⟩}

Sets ⟨*cmd*⟩ to ⟨*num*⟩ truncated to ⟨*num digits*⟩ digits after the decimal character.

```
\newcommand*{\DTLtrunc}[3]{%
\DTLconverttodecimal{#2}{\@dtl@numi}%
\FPtrunc{\@dtl@tmp}{\@dtl@numi}{#3}%
\ifx\@dtl@replaced\@empty
  \DTLdecimaltolocale{\@dtl@tmp}{#1}%
```

```
\else
  \DTLdecimaltocurrency{\@dtl@tmp}{#1}%
\fi
}
```

\DTLgtrunc Global version

```
\newcommand*{\DTLgtrunc}[3]{%
\DTLtrunc{\@dtl@tmpii}{#2}{#3}%
\global\let#1=\@dtl@tmpii
}
```

\DTLclip

\DTLclip{⟨*cmd*⟩}{⟨*num*⟩}

Sets ⟨*cmd*⟩ to ⟨*num*⟩ with all unnecessary 0's removed.

```
\newcommand*{\DTLclip}[2]{%
\DTLconverttodecimal{#2}{\@dtl@numi}%
\FPclip{\@dtl@tmp}{\@dtl@numi}%
\ifx\@dtl@replaced\@empty
  \DTLdecimaltolocale{\@dtl@tmp}{#1}%
\else
  \DTLdecimaltocurrency{\@dtl@tmp}{#1}%
\fi
}
```

\DTLgclip Global version

```
\newcommand*{\DTLgclip}[3]{%
\DTLclip{\@dtl@tmpii}{#2}%
\global\let#1=\@dtl@tmpii
}
```

10.15 String Macros

\DTLinitials

\DTLinitials{⟨*string*⟩}

Convert a string into initials. (Any ~ character found is first converted into a space.)

```
\newcommand*\DTLinitials[1]{%
\def\dtl@initialscmd{}%
\dtl@subnobrsp{#1}{\dtl@string}%
\DTLsubstituteall{\dtl@string}{~}{ }%
\DTLsubstituteall{\dtl@string}{\ }{ }%
\DTLsubstituteall{\dtl@string}{\space}{ }%
\expandafter\dtl@initials\dtl@string{} \@nil%
\dtl@initialscmd}%
```

The following substitutes `\protect \nobreakspace {}` with a space. (Note that in this case the space following `\nobreakspace` forms part of the command.)

```
\edef\dtl@construct@subnobrsp{%
\noexpand\def\noexpand\@dtl@subnobrsp##1\noexpand\protect
```

```
\expandafter\noexpand\csname nobreakspace \endcsname ##2{%
\noexpand\toks@{##1}%
\noexpand\expandafter\noexpand\@dtl@toks\noexpand\expandafter{%
\noexpand\@dtl@string}%
\noexpand\edef\noexpand\@dtl@string{\noexpand\the\noexpand\@dtl@toks
\noexpand\the\noexpand\toks@}%
\noexpand\def\noexpand\@dtl@tmp{##2}%
\noexpand\ifx\noexpand\@dtl@tmp\noexpand\@nnil
  \noexpand\let\noexpand\@dtl@subnobrspnext=\noexpand\relax
\noexpand\else
  \noexpand\toks@{ }%
  \noexpand\expandafter\noexpand\@dtl@toks\noexpand\expandafter{%
  \noexpand\@dtl@string}%
  \noexpand\edef\noexpand\@dtl@string{\noexpand\the\noexpand\@dtl@toks
  \noexpand\the\noexpand\toks@}%
  \noexpand\let\noexpand\@dtl@subnobrspnext=\noexpand\@dtl@subnobrsp
\noexpand\fi
\noexpand\@dtl@subnobrspnext
}%
\noexpand\def\noexpand\dtl@subnobrsp##1##2{%
\noexpand\def\noexpand\@dtl@string{}%
\noexpand\@dtl@subnobrsp ##1\noexpand\protect\expandafter\noexpand
\csname nobreakspace \endcsname \noexpand\@nil
\noexpand\let##2=\noexpand\@dtl@string
}}
\dtl@construct@subnobrsp
```

\DTLstoreinitials

\DTLstoreinitials{⟨*string*⟩}{⟨*cmd*⟩}

Convert a string into initials and store in ⟨*cmd*⟩. (Any ~ character found is first converted into a space.)

```
\newcommand*{\DTLstoreinitials}[2]{%
\def\dtl@initialscmd{}%
\dtl@subnobrsp{#1}{\dtl@string}%
\DTLsubstituteall{\dtl@string}{~}{ }%
\DTLsubstituteall{\dtl@string}{\ }{ }%
\DTLsubstituteall{\dtl@string}{\space}{ }%
\expandafter\dtl@initials\dtl@string{} \@nil
\let#2=\dtl@initialscmd
}
```

\dtl@initials

```
\def\dtl@initials#1#2 #3{%
\dtl@ifsingle{#1}{%
\ifcat\noexpand#1\relax\relax
   \def\@dtl@donextinitials{\@dtl@initials#2 {#3}}%
\else
   \def\@dtl@donextinitials{\@dtl@initials#1#2 {#3}}%
\fi
}{%
 \def\@dtl@donextinitials{\@dtl@initials{#1}#2 {#3}}%
}%
```

```
\@dtl@donextinitials
}
```

\@dtl@initials

```
\def\@dtl@initials#1#2 #3{%
\dtl@initialshyphen#2-{}\dtl@endhyp
\expandafter\@dtl@toks\expandafter{\dtl@initialscmd}%
\toks@{#1}%
\ifx\dtl@inithyphen\@empty
\else
  \edef\dtl@initialscmd{\the\@dtl@toks\the\toks@}%
  \expandafter\@dtl@toks\expandafter{\dtl@initialscmd}%
  \expandafter\toks@\expandafter{\dtl@inithyphen}%
\fi
\def\dtl@tmp{#3}%
\ifx\@nnil\dtl@tmp
 \edef\dtl@initialscmd{\the\@dtl@toks\the\toks@\DTLafterinitials}%
 \let\dtl@initialsnext=\@gobble
\else
 \edef\dtl@initialscmd{\the\@dtl@toks\the\toks@\DTLbetweeninitials}%
 \let\dtl@initialsnext=\dtl@initials
\fi
\dtl@initialsnext{#3}}
```

\dtl@initialshyphen

```
\def\dtl@initialshyphen#1-#2#3\dtl@endhyp{%
\def\dtl@inithyphen{#2}%
\ifx\dtl@inithyphen\@empty
\else
 \edef\dtl@inithyphen{%
   \DTLafterinitialbeforehyphen\DTLinitialhyphen#2}%
\fi
}
```

\DTLafterinitials Defines what to do after the final initial.

```
\newcommand*{\DTLafterinitials}{.}
```

\DTLbetweeninitials Defines what to do between initials.

```
\newcommand*{\DTLbetweeninitials}{.}
```

\DTLafterinitialbeforehyphen Defines what to do before a hyphen.

```
\newcommand*{\DTLafterinitialbeforehyphen}{.}
```

\DTLinitialhyphen Defines what to do at the hyphen

```
\newcommand*{\DTLinitialhyphen}{-}
```

\DTLifAllUpperCase

\DTLifAllUpperCase{⟨*string*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

If ⟨*string*⟩ only contains uppercase characters do ⟨*true part*⟩, otherwise do ⟨*false part*⟩.

```
\newcommand*{\DTLifAllUpperCase}[3]{%
```

```
\protected@edef\dtl@tuc{#1}%
\expandafter\dtl@testifuppercase\dtl@tuc\@nil\relax
\if@dtl@condition#2\else#3\fi
}
```

\dtl@testifalluppercase

```
\def\dtl@testifuppercase#1#2{%
\def\dtl@argi{#1}%
\def\dtl@argii{#2}%
\def\dtl@tc@rest{}%
\ifx\dtl@argi\@nnil
  \let\dtl@testifuppernext=\@nnil
\else
  \ifx#1\protect
    \let\dtl@testifuppernext=\dtl@testifuppercase
  \else
    \ifx\uppercase#1\relax
      \@dtl@conditiontrue
      \def\dtl@tc@rest{}%
      \let\dtl@testifuppernext=\relax
    \else
      \edef\dtl@tc@arg{\string#1}%
      \expandafter\dtl@test@ifuppercase\dtl@tc@arg\end
      \ifx\dtl@argii\@nnil
        \let\dtl@testifuppernext=\@dtl@gobbletonil
      \fi
    \fi
  \fi
\fi
\ifx\dtl@testifuppernext\relax
 \edef\dtl@dotestifuppernext{%
   \noexpand\dtl@testifuppercase}%
\else
 \ifx\dtl@testifuppernext\@nnil
   \edef\dtl@dotestifuppernext{#2}%
 \else
   \expandafter\toks@\expandafter{\dtl@tc@rest}%
   \@dtl@toks{#2}%
   \edef\dtl@dotestifuppernext{%
     \noexpand\dtl@testifuppernext\the\toks@\the\@dtl@toks}%
 \fi
\fi
\dtl@dotestifuppernext
}
```

\dtl@test@ifalluppercase

```
\def\dtl@test@ifuppercase#1#2\end{%
\def\dtl@tc@rest{#2}%
\IfSubStringInString{\string\MakeUppercase}{#1#2}{%
  \@dtl@conditiontrue
  \def\dtl@tc@rest{}%
  \let\dtl@testifuppernext=\relax
}{%
 \IfSubStringInString{\string\MakeTextUppercase}{#1#2}{%
```

```
      \@dtl@conditiontrue
      \def\dtl@tc@rest{}%
      \let\dtl@testifuppernext=\relax
    }{%
     \edef\dtl@uccode{\the\uccode`#1}%
     \edef\dtl@code{\number`#1}%
     \ifnum\dtl@code=\dtl@uccode\relax
       \@dtl@conditiontrue
       \let\dtl@testifuppernext=\dtl@testifuppercase
     \else
       \ifnum\dtl@uccode=0\relax
         \@dtl@conditiontrue
         \let\dtl@testifuppernext=\dtl@testifuppercase
       \else
         \@dtl@conditionfalse
         \let\dtl@testifuppernext=\@dtl@gobbletonil
       \fi
     \fi
   }}}
```

`\DTLifAllLowerCase`

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

If ⟨*string*⟩ only contains lowercase characters do ⟨*true part*⟩, otherwise do ⟨*false part*⟩.

```
\newcommand*{\DTLifAllLowerCase}[3]{%
\protected@edef\dtl@tlc{#1}%
\expandafter\dtl@testiflowercase\dtl@tlc\@nil\relax
\if@dtl@condition#2\else#3\fi
}
```

`\dtl@testifalllowercase`

```
\def\dtl@testiflowercase#1#2{%
\def\dtl@argi{#1}%
\def\dtl@argii{#2}%
\ifx\dtl@argi\@nnil
  \let\dtl@testiflowernext=\@nnil
\else
  \ifx#1\protect
    \let\dtl@testiflowernext=\dtl@testiflowercase
  \else
    \ifx\lowercase#1\relax
      \@dtl@conditiontrue
      \def\dtl@tc@rest{}%
      \let\dtl@testiflowernext=\relax
    \else
      \edef\dtl@tc@arg{\string#1}%
      \expandafter\dtl@test@iflowercase\dtl@tc@arg\end
      \ifx\dtl@argii\@nnil
        \let\dtl@testiflowernext=\@dtl@gobbletonil
      \fi
    \fi
  \fi
```

```
\fi
\ifx\dtl@testiflowernext\relax
 \edef\dtl@dotestiflowernext{%
   \noexpand\dtl@testiflowercase}%
\else
 \ifx\dtl@testiflowernext\@nnil
   \edef\dtl@dotestiflowernext{#2}%
 \else
   \expandafter\toks@\expandafter{\dtl@tc@rest}%
   \@dtl@toks{#2}%
   \edef\dtl@dotestiflowernext{%
     \noexpand\dtl@testiflowernext\the\toks@\the\@dtl@toks}%
 \fi
\fi
\dtl@dotestiflowernext
}
```

\dtl@test@ifalllowercase

```
\def\dtl@test@iflowercase#1#2\end{%
\def\dtl@tc@rest{#2}%
\IfSubStringInString{\string\MakeLowercase}{#1#2}{%
  \@dtl@conditiontrue
  \def\dtl@tc@rest{}%
  \let\dtl@testiflowernext=\relax
}{%
 \IfSubStringInString{\string\MakeTextLowercase}{#1#2}{%
    \@dtl@conditiontrue
    \def\dtl@tc@rest{}%
    \let\dtl@testiflowernext=\relax
 }{%
  \edef\dtl@lccode{\the\lccode`#1}%
  \edef\dtl@code{\number`#1}%
  \ifnum\dtl@code=\dtl@lccode\relax
    \@dtl@conditiontrue
    \let\dtl@testiflowernext=\dtl@testiflowercase
  \else
    \ifnum\dtl@lccode=0\relax
      \@dtl@conditiontrue
      \let\dtl@testiflowernext=\dtl@testiflowercase
    \else
      \@dtl@conditionfalse
      \let\dtl@testiflowernext=\@dtl@gobbletonil
    \fi
  \fi
}}}
```

10.16 Saving a database to an external file

\@dtl@write

```
\newwrite\@dtl@write
```

\DTLsavedb

\DTLsavedb{⟨*db name*⟩}{⟨*filename*⟩}

Save a database as an ASCII data file using the separator and delimiter given by `\@dtl@separator` and `\@dtl@delimiter`.

```
\newcommand*{\DTLsavedb}[2]{%
  \DTLifdbexists{#1}%
  {%
```

Open output file

```
    \openout\@dtl@write=#2\relax
```

Initialise header row

```
    \def\@dtl@header{}%
```

Construct the header row

```
    \dtlforeachkey(\@dtl@key,\@dtl@col,\@dtl@type,\@dtl@head)%
      \in{#1}\do
    {%
      \IfSubStringInString{\@dtl@separator}{\@dtl@key}%
      {%
        \ifx\@dtl@header\@empty
          \protected@edef\@dtl@header{%
            \@dtl@delimiter\@dtl@key\@dtl@delimiter}%
        \else
          \toks@=\expandafter{\@dtl@header}%
          \protected@edef\@dtl@header{%
            \the\toks@\@dtl@separator
            \@dtl@delimiter\@dtl@key\@dtl@delimiter}%
        \fi
      }%
      {%
        \ifx\@dtl@header\@empty
          \protected@edef\@dtl@header{\@dtl@key}%
        \else
          \toks@=\expandafter{\@dtl@header}%
          \protected@edef\@dtl@header{\the\toks@
            \@dtl@separator\@dtl@key}%
        \fi
      }%
    }%
```

Print header

```
    \protected@write\@dtl@write{}{\@dtl@header}%
```

Iterate through each row

```
    \@sDTLforeach{#1}{}%
    {%
```

Initialise row

```
      \def\@dtl@row{}%
```

Iterate through each key

```
      \DTLforeachkeyinrow{\@dtl@val}%
      {%
        \IfSubStringInString{\@dtl@separator}{\@dtl@val}%
        {%
          \ifx\@dtl@row\@empty
            \protected@edef\@dtl@row{%
```

```
            \@dtl@delimiter\@dtl@val\@dtl@delimiter}%
          \else
            \toks@=\expandafter{\@dtl@row}%
            \protected@edef\@dtl@row{\the\toks@\@dtl@separator
              \@dtl@delimiter\@dtl@val\@dtl@delimiter}%
          \fi
        }%
        {%
          \ifx\@dtl@row\@empty
            \protected@edef\@dtl@row{\@dtl@val}%
          \else
            \toks@=\expandafter{\@dtl@row}%
            \protected@edef\@dtl@row{\the\toks@\@dtl@separator
              \@dtl@val}%
          \fi
        }%
```

Print row

```
        \protected@write\@dtl@write{}{\@dtl@row}%
      }%
    }%
```

Close output file

```
    \closeout\@dtl@write
  }%
  {%
    \PackageError{datatool}{Can't save database `#1': no such
      database}{}%
  }%
}
```

\DTLsavetexdb

\DTLsavetexdb{⟨*db name*⟩}{⟨*filename*⟩}

Save a database as a LaTeX file.

```
\newcommand*{\DTLsavetexdb}[2]{%
  \DTLifdbexists{#1}%
  {%
```

Open output file

```
    \openout\@dtl@write=#2\relax
```

Write new data base definition

```
    \protected@write\@dtl@write{}{\string\DTLnewdb{#1}}%
```

Iterate through each row

```
    \@sDTLforeach{#1}{}%
    {%
```

Start new row

```
      \protected@write\@dtl@write{}{\string\DTLnewrow*{#1}}%
```

Iterate through each column

```
      \DTLforeachkeyinrow{\@dtl@val}%
      {%
```

Is this entry null?

```
          \DTLifnull{\@dtl@val}%
          {\def\@dtl@val{}}%
          {}%
```

Add entry

```
          \protected@write\@dtl@write{}{%
            \string\DTLnewdbentry*{#1}{\dtlkey}{\@dtl@val}}%
        }%
      }%
```

Save the column headers.

```
      \dtlforeachkey(\@dtl@k,\@dtl@c,\@dtl@t,\@dtl@h)\in{#1}\do
      {%
        \@onelevel@sanitize\@dtl@h
        \protected@write\@dtl@write{}{%
          \string\DTLsetheader*{#1}{\@dtl@k}{\@dtl@h}}%
      }%
```

Close output file

```
      \closeout\@dtl@write
    }%
    {%
      \PackageError{datatool}{Can't save database `#1': no such
        database}{}%
    }%
  }
```

10.17 Loading a database from an external file

```
\newcommand{\@longempty}{}
```

`\@dtl@read`

```
\newread\@dtl@read
```

`\dtl@entrycr` Keep track of current column in data file

```
\newcount\dtl@entrycr
```

`\ifdtlnoheader` The `noheader` option indicates that the file doesn't have a header row.

```
\define@boolkey{loaddb}[dtl]{noheader}[true]{}
```

The `keys` option specifies the list of keys in the same order as the columns in the data file. Each key is stored in `\@dtl@inky@`$\langle n\rangle$ where $\langle n\rangle$ is the roman numeral representation of the current column.

```
\define@key{loaddb}{keys}{%
  \dtl@entrycr=0\relax
  \@for\@dtl@key:=#1\do
  {%
    \advance\dtl@entrycr by 1\relax
    \expandafter
      \edef\csname @dtl@inky@\romannumeral\dtl@entrycr\endcsname{%
        \@dtl@key}%
  }%
}
```

The headers option specifies the list of headers in the same order as the columns in the data file.

```
\define@key{loaddb}{headers}{%
  \dtl@entrycr=0\relax
  \@for\@dtl@head:=#1\do
  {%
    \advance\dtl@entrycr by 1\relax
    \toks@=\expandafter{\@dtl@head}%
    \expandafter
      \edef\csname @dtl@inhd@\romannumeral\dtl@entrycr\endcsname{%
        \the\toks@}%
  }%
}
```

\dtldefaultkey Default key to use if none specified (column index will be appended).

```
\newcommand*{\dtldefaultkey}{Column}
```

\@dtl@readline

\@dtl@readline{⟨*file reg*⟩}{⟨*cs*⟩}

Reads line from ⟨*file reg*⟩, trims end of line character and stores in ⟨*cs*⟩.

```
\newcommand*{\@dtl@readline}[2]{%
% Read a line from "#1" and store in "#2"
%    \begin{macrocode}
  \read#1 to #2%
```

Trim the end of line character

```
  \dtl@trim#2%
}
```

\@dtl@readrawline

\@dtl@readrawline{⟨*file register*⟩}{⟨*cs*⟩}

Reads line from ⟨*file register*⟩, trims end of line character, applies mappings and stores in ⟨*cs*⟩.

```
\newcommand*{\@dtl@readrawline}[2]{%
% Read a line from "#1" and store in "#2"
%    \begin{macrocode}
  \@dtl@rawread#1 to #2%
```

Trim the end of line character

```
  \dtl@trim#2%
```

Apply mappings

```
  \dtl@domappings\@dtl@line
}
```

\DTLloaddb

\DTLloaddb[⟨*options*⟩]{⟨*db name*⟩}{⟨*filename*⟩}

Creates a new database called ⟨*db name*⟩, and loads the data in ⟨*filename*⟩ into it. The separator and delimiter used in the file must match \@dtl@separator and \@dtl@delimiter. The optional argument is a comma-separated list.

```
\newcommand*{\DTLloaddb}{%
  \let\@dtl@doreadline\@dtl@readline
  \@dtlloaddb
}
```

`\@dtlloaddb` Loads database using `\@dtl@doreadline` to read and trim line from file. (`\@dtl@doreadline` must be set before use.)

```
\newcommand*{\@dtlloaddb}[3][]{%
```

Check if file exists

```
  \IfFileExists{#3}{%
```

File exists. Locally change catcode of double quote character in case it has been made active.

```
  \begingroup
    \catcode`\"12\relax
```

Initialise default options

```
      \dtlnoheaderfalse
```

Get the options

```
      \setkeys{loaddb}{#1}%
```

Open the file for reading.

```
    \openin\@dtl@read=#3%
    \dtl@message{Reading `#3'}%
```

Create the new database.

```
    \DTLnewdb{#2}%
```

Check if the file is empty.

```
    \ifeof\@dtl@read
```

File is empty, so just issue a warning.

```
      \PackageWarning{datatool}{File `#3' has no data}%
    \else
```

Does the file have a header row?

```
      \ifdtlnoheader
      \else
```

Remove initial blank rows

```
        \loop
```

Set repeat condition to false

```
          \@dtl@conditionfalse
```

Do nothing if reached the end of file

```
          \ifeof\@dtl@read
          \else
```

Read a line from the file and store in `\@dtl@line`

```
            \@dtl@doreadline\@dtl@read\@dtl@line
```

If this is a blank row, set repeat condition to true

```
            \ifx\@dtl@line\@longempty
              \@dtl@conditiontrue
            \fi
          \fi
```

Repeat loop if necessary

```
\if@dtl@condition
\repeat
```

Parse the header row. Store the row as ⟨*sep*⟩⟨*row*⟩⟨*sep*⟩ in `\@dtl@lin@`.

```
\protected@edef\@dtl@lin@{%
   \@dtl@separator\@dtl@line\@dtl@separator}%
```

Keep track of columns:

```
\dtl@entrycr=0\relax
```

Keep lopping off elements until the end of the row is reached. (That is, until `\@dtl@lin@` is `\@dtl@separator`.)

```
\loop
```

Lopoff the first element and store in `\@dtl@key`

```
\expandafter\@dtl@lopoff\@dtl@lin@\to\@dtl@lin@\@dtl@key
```

Increment column count.

```
\advance\dtl@entrycr by 1\relax
```

Store key in `\@dtl@toks`

```
\expandafter\@dtl@toks\expandafter{\@dtl@key}%
```

Store the key in `\@dtl@inky@`⟨*n*⟩ where ⟨*n*⟩ is the roman numeral representation of the current column, unless already defined.

```
\@ifundefined{@dtl@inky@\romannumeral\dtl@entrycr}%
{%
  \expandafter
    \edef\csname @dtl@inky@\romannumeral
      \dtl@entrycr\endcsname{\the\@dtl@toks}%
}%
{%
```

If key has been specified in `#1`, then use the header found in the file, unless a header has also been specified in `#1`

```
  \@ifundefined{@dtl@inhd@\romannumeral\dtl@entrycr}%
  {%
    \expandafter
      \edef\csname @dtl@inhd@\romannumeral
        \dtl@entrycr\endcsname{\the\@dtl@toks}%
  }%
  {}%
}%
```

Check if the loop should be repeated

```
\ifx\@dtl@lin@\@dtl@separator
  \@dtl@conditionfalse
\else
  \@dtl@conditiontrue
\fi
```

Repeat loop if necessary.

```
\if@dtl@condition
\repeat
```

End if no header

```
\fi
```

Now for the rest of the data. If the end of file has been reached, then only the header row is available or file is empty.

```
\ifeof\@dtl@read
   \ifdtlnoheader
     \PackageWarning{datatool}{No data in ‘#3’}%
   \else
     \PackageWarning{datatool}{Only header row found in ‘#3’}%
   \fi
\else
```

Iterate through the rest of the file. First set the repeat condition to true:

```
  \@dtl@conditiontrue
  \loop
```

Read in a line

```
   \@dtl@doreadline\@dtl@read\@dtl@line
```

Check if the line is empty.

```
    \ifx\@dtl@line\@longempty
```

Do nothing if the row is empty.

```
    \else
```

Add a new row to the database. (Don't need to check if the database exists, since it's just been created.)

```
      \@sDTLnewrow{#2}%
```

Store the row as ⟨*sep*⟩⟨*row*⟩⟨*sep*⟩ to make the lopping off easier

```
      \expandafter\@dtl@toks\expandafter{\@dtl@line}%
      \edef\@dtl@lin@{\@dtl@separator\the\@dtl@toks
        \@dtl@separator}%
```

Reset the column counter.

```
      \dtl@entrycr=0\relax
```

Iterate through each element in the row. Needs to be grouped since we're already inside a loop.

```
      {%
```

Initialise repeat condition

```
        \@dtl@conditiontrue
```

Iterate through the list

```
        \loop
```

lop off first element and store in `\@dtl@thisentry`

```
          \expandafter\@dtl@lopoff\@dtl@lin@\to
            \@dtl@lin@\@dtl@thisentry
```

Increment the column count.

```
          \advance\dtl@entrycr by 1\relax
```

Get the key for this column and store in `\@dtl@thiskey`. Use default value if not defined.

```
          \@ifundefined{@dtl@inky@\romannumeral\dtl@entrycr}%
          {%
            \edef\@dtl@thiskey{\dtldefaultkey
              \number\dtl@entrycr}%
```

```
      \expandafter\let
        \csname @dtl@inky@\romannumeral
          \dtl@entrycr\endcsname\@dtl@thiskey
    }%
    {%
      \edef\@dtl@thiskey{%
        \csname @dtl@inky@\romannumeral
          \dtl@entrycr\endcsname}%
    }%
```

Store this entry in `\@dtl@toks`

```
    \expandafter\@dtl@toks\expandafter{\@dtl@thisentry}%
```

Add this entry to the database

```
    \edef\@do@dtlnewentry{\noexpand\@sDTLnewdbentry
      {#2}{\@dtl@thiskey}{\the\@dtl@toks}}%
    \@do@dtlnewentry
```

Check if loop should be terminated

```
    \ifx\@dtl@lin@\@dtl@separator
      \@dtl@conditionfalse
    \fi
```

Repeat loop if necessary

```
      \if@dtl@condition
      \repeat
  }%
```

End of parsing this row

```
  \fi
```

If the end of file has been reached, set the repeat condition to false.

```
  \ifeof\@dtl@read \@dtl@conditionfalse\fi
```

Repeat if necessary

```
  \if@dtl@condition
  \repeat
\fi
```

End of first `\ifeof`

```
\fi
```

Close the input file

```
\closein\@dtl@read
```

Set the headers if required

```
\edef\@dtl@maxcols{\expandafter
  \number\csname dtlcols@#2\endcsname}%
\dtlgforint\dtl@entrycr=1\to\@dtl@maxcols\step1\do
{%
  \@ifundefined{@dtl@inhd@\romannumeral\dtl@entrycr}%
  {}%
  {%
     \expandafter\let\expandafter\@dtl@head
       \csname @dtl@inhd@\romannumeral\dtl@entrycr\endcsname
     \@dtl@toks=\expandafter{\@dtl@head}%
     \edef\@dtl@dosetheader{\noexpand\@dtl@setheaderforindex
       {#2}{\number\dtl@entrycr}{\the\@dtl@toks}}%
```

```
            \@dtl@dosetheader
          }%
        }%
```

End current scope

```
    \endgroup
```

End true part of if file exists

```
    }{%
```

Requested file not found on TeX's path

```
      \PackageError{datatool}{Can't load database `#2' (file `#3'
      doesn't exist)}{}%
    }%
  }
```

`\dtl@trim` `\dtl@trim{⟨line⟩}`

Trims the trailing space from ⟨*line*⟩.

```
\newcommand{\dtl@trim}[1]{%
\def\@dtl@trmstr{}%
\expandafter\@dtl@starttrim#1\@nil%
\let#1=\@dtl@trmstr
}
```

`\@dtl@starttrim` Start trimming

```
\long\def\@dtl@starttrim#1#2{%
\ifx\par#1%
 \def\@dtl@dotrim{\@dtl@trim{} #2}%
\else
 \def\@dtl@dotrim{\@dtl@trim#1#2}%
\fi
\@dtl@dotrim%
}
```

`\@dtl@trim`

```
\long\def\@dtl@trim#1 \@nil{\long\def\@dtl@trmstr{#1}}
```

`\DTLloadrawdb` `\DTLloadrawdb{⟨db name⟩}{⟨filename⟩}`

Loads a raw database (substitutes `%` → `\%`, `$` → `\$`, `&` → `\&`, `#` → `\#`, `~` → `\textasciitilde`, `_` → `\_` and `^` → `\textasciicircum`.)

```
\newcommand*\DTLloadrawdb{%
  \let\@dtl@doreadline\@dtl@readrawline
  \@dtlloaddb
}
```

`\@dtl@rawread` `\@dtl@rawread⟨number⟩to⟨cmd⟩`

Reads in a raw line from file given by ⟨*number*⟩ converts special characters and stores in ⟨*cmd*⟩

```
\begingroup
\catcode`\%=\active
\catcode`$=\active
\catcode`&=\active
\catcode`~=\active
\catcode`_=\active
\catcode`^=\active
\catcode`#=\active
\catcode`?=6\relax
\catcode`<=1\relax
\catcode`>=2\relax
\catcode`\{=\active
\catcode`\}=\active
\gdef\@dtl@rawread?1to?2<\relax
<<\catcode`\%=\active
\catcode`$=\active
\catcode`&=\active
\catcode`~=\active
\catcode`_=\active
\catcode`^=\active
\catcode`#=\active
\catcode`\{=\active
\catcode`\}=\active
\def%<\noexpand\%>\relax
\def$<\noexpand\$>\relax
\def&<\&>\relax
\def#<\#>\relax
\def~<\noexpand\textasciitilde>\relax
\def_<\noexpand\_>\relax
\def^<\noexpand\textasciicircum>\relax
\@dtl@activatebraces
\@dtl@doreadraw?1?2>>>
\gdef\@dtl@doreadraw?1?2<\relax
\read?1 to \tmp
\xdef?2<\tmp>\relax
>
\endgroup
```

`\@dtl@activatebraces` `\@dtl@activatebraces` resets braces for `\@dtl@rawread`

```
\begingroup
\catcode`\{=\active
\catcode`\}=\active
\catcode`<=1\relax
\catcode`>=2\relax
\gdef\@dtl@activatebraces<%
 \catcode`\{=\active
 \catcode`\}=\active
 \def{<\noexpand\{>%
 \def}<\noexpand\}>%
>%
\endgroup
```

\DTLrawmap

\DTLrawmap{⟨*string*⟩}{⟨*replacement*⟩}

Additional mappings to perform when reading a raw data file

```
\newcommand*{\DTLrawmap}[2]{%
\expandafter\@dtl@toks\expandafter{\@dtl@rawmappings}%
\ifx\@dtl@rawmappings\@empty
  \def\@dtl@rawmappings{{#1}{#2}}%
\else
  \def\@dtl@tmp{{#1}{#2}}
  \protected@edef\@dtl@rawmappings{\the\@dtl@toks,\@dtl@tmp}
\fi
}
```

\@dtl@rawmappings

List of mappings.

```
\newcommand*{\@dtl@rawmappings}{}
```

\dtl@domappings

\dtl@domappings{⟨*cmd*⟩}

Do all mappings in string given by ⟨*cmd*⟩.

```
\newcommand*{\dtl@domappings}[1]{%
\@for\@dtl@map:=\@dtl@rawmappings\do{%
  \expandafter\DTLsubstitute\expandafter#1\@dtl@map
}}
```

\DTLsubstitute

\DTLsubstitute{⟨*cmd*⟩}{⟨*original*⟩}{⟨*replacement*⟩}

Substitutes first occurence of ⟨*original*⟩ with {⟨*replacement*⟩} within the string given by ⟨*cmd*⟩

```
\newcommand{\DTLsubstitute}[3]{%
\expandafter\DTLsplitstring\expandafter
  {#1}{#2}{\@dtl@beforepart}{\@dtl@afterpart}%
\ifx\@dtl@replaced\@empty
\else
  \def#1{}%
  \expandafter\@dtl@toks\expandafter{\@dtl@beforepart}%
  \expandafter\toks@\expandafter{#1}%
  \protected@edef#1{\the\toks@\the\@dtl@toks#3}%
  \expandafter\@dtl@toks\expandafter{\@dtl@afterpart}%
  \expandafter\toks@\expandafter{#1}%
  \edef#1{\the\toks@\the\@dtl@toks}%
\fi
}
```

\DTLsplitstring

\DTLsplitstring{⟨*string*⟩}{⟨*split text*⟩}{⟨*before cmd*⟩}{⟨*after cmd*⟩}

Splits string at ⟨*split text*⟩ stores the pre split text in ⟨*before cmd*⟩ and the post split text in ⟨*after cmd*⟩.

```
\newcommand*{\DTLsplitstring}[4]{%
\def\dtl@splitstr##1#2##2\@nil{%
\def#3{##1}%
\def#4{##2}%
\ifx#4\@empty
  \let\@dtl@replaced=\@empty
\else
  \def\@dtl@replaced{#2}%
  \dtl@split@str##2\@nil
\fi
}%
\def\dtl@split@str##1#2\@nil{%
\def#4{##1}}%
\dtl@splitstr#1#2\@nil
}
```

`\DTLsubstituteall` **`\DTLsubstituteall{`⟨*cmd*⟩`}{`⟨*original*⟩`}{`⟨*replacement*⟩`}`**

Substitutes all occurences of ⟨*original*⟩ with `{`⟨*replacement*⟩`}` within the string given by ⟨*cmd*⟩

```
\newcommand{\DTLsubstituteall}[3]{%
\def\@dtl@splitsubstr{}%
\let\@dtl@afterpart=#1\relax
\@dtl@dosubstitute{#2}{#3}%
\expandafter\toks@\expandafter{\@dtl@splitsubstr}%
\expandafter\@dtl@toks\expandafter{\@dtl@afterpart}%
\edef#1{\the\toks@\the\@dtl@toks}%
}
```

`\@dtl@dosubstitute` Recursive substitution macro.

```
\def\@dtl@dosubstitute#1#2{%
\expandafter\DTLsplitstring\expandafter
 {\@dtl@afterpart}{#1}{\@dtl@beforepart}{\@dtl@afterpart}%
\expandafter\toks@\expandafter{\@dtl@splitsubstr}%
\expandafter\@dtl@toks\expandafter{\@dtl@beforepart}%
\edef\@dtl@splitsubstr{\the\toks@\the\@dtl@toks}%
\ifx\@dtl@replaced\@empty
  \let\@dtl@dosubstnext=\@dtl@dosubstitutenoop
\else
  \expandafter\toks@\expandafter{\@dtl@splitsubstr}%
  \@dtl@toks{#2}%
  \edef\@dtl@splitsubstr{\the\toks@\the\@dtl@toks}%
  \let\@dtl@dosubstnext=\@dtl@dosubstitute
\fi
\@dtl@dosubstnext{#1}{#2}%
}
```

`\@dtl@dosubstitutenoop` Terminates recursive substitution macro.

```
\def\@dtl@dosubstitutenoop#1#2{}
```

`\DTLifinlist` `\DTLifinlist{`⟨*element*⟩`}{`⟨*list*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}` If ⟨*element*⟩ is contained in the comma-separated list given by ⟨*list*⟩, then do ⟨*true part*⟩ otherwise

do false part. (Does a one level expansion on ⟨*list*⟩)

```
\newcommand*{\DTLifinlist}[4]{%
  \def\@dtl@doifinlist##1,#1,##2\end@dtl@doifinlist{%
     \def\@before{##1}%
     \def\@after{##2}%
  }%
  \expandafter\@dtl@doifinlist\expandafter,#2,#1,\@nil
    \end@dtl@doifinlist
  \ifx\@after\@nnil
% not found
    #4%
  \else
% found
    #3%
  \fi
}
```

10.18 Currencies

`\@dtl@currencies` `\@dtl@currencies` stores all known currencies.

```
\newcommand*{\@dtl@currencies}{\$,\pounds}
```

`\DTLnewcurrencysymbol`

`\DTLaddcurrency{`⟨*symbol*⟩`}`

Adds ⟨*symbol*⟩ to the list of known currencies

```
\newcommand*{\DTLnewcurrencysymbol}[1]{%
\expandafter\toks@\expandafter{\@dtl@currencies}%
\@dtl@toks{#1}%
\edef\@dtl@currencies{\the\@dtl@toks,\the\toks@}%
}
```

If any of the following currency commands have been defined, add them to the list:

```
\AtBeginDocument{%
\@ifundefined{texteuro}{}{\DTLnewcurrencysymbol{\texteuro}}%
\@ifundefined{textdollar}{}{\DTLnewcurrencysymbol{\textdollar}}%
\@ifundefined{textstirling}{}{\DTLnewcurrencysymbol{\textstirling}}%
\@ifundefined{textyen}{}{\DTLnewcurrencysymbol{\textyen}}%
\@ifundefined{textwon}{}{\DTLnewcurrencysymbol{\textwon}}%
\@ifundefined{textcurrency}{}{\DTLnewcurrencysymbol{\textcurrency}}%
\@ifundefined{euro}{}{\DTLnewcurrencysymbol{\euro}}%
\@ifundefined{yen}{}{\DTLnewcurrencysymbol{\yen}}%
}
```

`\@dtl@standardize@currency`

`\@dtl@standardize@currency{`⟨*cmd*⟩`}`

Substitutes the first currency symbol found in ⟨*cmd*⟩ with `\$`. This is used when testing text to determine if it is currency. The original currency symbol is

stored in \@dtl@org@currency, so that it can be replaced later. If no currency symbol is found, \@dtl@org@currency will be empty.

```
\newcommand{\@dtl@standardize@currency}[1]{%
\def\@dtl@org@currency{}%
\@for\@dtl@thiscurrency:=\@dtl@currencies\do{%
  \expandafter\toks@\expandafter{\@dtl@thiscurrency}%
  \edef\@dtl@dosubs{\noexpand\DTLsubstitute{\noexpand#1}%
   {\the\toks@}{\noexpand\$}}%
  \@dtl@dosubs
  \ifx\@dtl@replaced\@empty
  \else
    \let\@dtl@org@currency=\@dtl@replaced
    \@endfortrue
  \fi
}%
\@endforfalse}
```

\@dtl@currency

\@dtl@currency is set by \DTLlocaltodecimal and \@dtl@checknumerical. It is used by \DTLdecimaltocurrency. Set to \$ by default.

```
\newcommand*{\@dtl@currency}{\$}
```

\DTLsetdefaultcurrency

\DTLsetdefaultcurrency{⟨*symbol*⟩} sets the default currency.

```
\newcommand*{\DTLsetdefaultcurrency}[1]{%
\renewcommand*{\@dtl@currency}{#1}}
```

\dtl@ifsingle

\dtl@ifsingle{⟨*arg*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

If there is only one object in ⟨*arg*⟩ (without expansion) do ⟨*true part*⟩, otherwise do false part.

```
\newcommand{\dtl@ifsingle}[3]{%
\def\@dtl@arg{#1}%
\ifx\@dtl@arg\@empty
  #3%
\else
  \@dtl@ifsingle#1\@nil{#2}{#3}%
\fi
}
```

\@dtl@ifsingle

```
\def\@dtl@ifsingle#1#2\@nil#3#4{%
\def\dtl@sg@arg{#2}%
\ifx\dtl@sg@arg\@empty
 #3%
\else
 #4%
\fi
}
```

10.19 Debugging commands

These commands are provided to assist debugging

\dtlshowdb

\dtlshowdb{⟨*db name*⟩}

Shows the database.

```
\newcommand*{\dtlshowdb}[1]{%
\expandafter\showthe\csname dtldb@#1\endcsname}
```

\dtlshowdbkeys

\dtlshowdbkeys{⟨*db name*⟩}

Shows the key list for the named database.

```
\newcommand*{\dtlshowdbkeys}[1]{%
\expandafter\showthe\csname dtlkeys@#1\endcsname}
```

\dtlshowtype

\dtlshowtype{⟨*db name*⟩}{⟨*key*⟩}

Show the data type for given key in the named database. This should be an integer from 0 to 3.

```
\newcommand*{\dtlshowtype}[2]{%
  \DTLgetdatatype{\@dtl@type}{#1}{#2}\show\@dtl@type
}
```

11 datapie.sty

Declare package:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{datapie}[2007/02/27 v2.0 (NLCT)]
```

Require xkeyval package

```
\RequirePackage{xkeyval}
```

\ifDTLcolorpiechart

The conditional \ifDTLcolorpiechart is to determine whether to use colour or grey scale.

```
\newif\ifDTLcolorpiechart
\DTLcolorpiecharttrue
```

Package options to change the conditional:

```
\DeclareOption{color}{\DTLcolorpiecharttrue}
\DeclareOption{gray}{\DTLcolorpiechartfalse}
```

\ifDTLrotateinner

Define boolean keys to govern label rotations.

```
\define@boolkey{datapie}[DTL]{rotateinner}[true]{}
```

\ifDTLrotateouter

```
\define@boolkey{datapie}[DTL]{rotateouter}[true]{}
```

Set defaults:

```
\DTLrotateinnerfalse
\DTLrotateouterfalse
```

Package options to change \DTLrotateinner

```
\DeclareOption{rotateinner}{\DTLrotateinnertrue}
\DeclareOption{norotateinner}{\DTLrotateinnerfalse}
```

Package options to change \DTLrotateouter

```
\DeclareOption{rotateouter}{\DTLrotateoutertrue}
\DeclareOption{norotateouter}{\DTLrotateouterfalse}
```

Process options:

```
\ProcessOptions
```

Required packages:

```
\RequirePackage{datatool}
\RequirePackage{tikz}
```

Define some variables that govern the appearance of the pie chart.

\DTLradius The radius of the pie chart is given by \DTLradius.

```
\newlength\DTLradius
\DTLradius=2cm
```

\DTLinnerratio The inner label offset ratio is given by \DTLinnerratio

```
\newcommand*{\DTLinnerratio}{0.5}
```

\DTLouterratio The outer label offset ratio is given by \DTLouterratio.

```
\newcommand*{\DTLouterratio}{1.25}
```

\DTLcutawayratio The cutaway offset ratio is given by \DTLcutawayratio.

```
\newcommand*\DTLcutawayratio{0.2}
```

\DTLstartangle The angle of the first segment is given by \DTLstartangle.

```
\newcommand*{\DTLstartangle}{0}
```

\dtl@inneroffset

```
\newlength\dtl@inneroffset
\dtl@inneroffset=\DTLinnerratio\DTLradius
```

\dtl@outeroffset

```
\newlength\dtl@outeroffset
\dtl@outeroffset=\DTLouterratio\DTLradius
```

\dtl@cutawayoffset

```
\newlength\dtl@cutawayoffset
\dtl@cutawayoffset=\DTLcutawayratio\DTLradius
```

\dtl@piecutaways \dtl@piecutaways is a comma separated list of segments that need to be cut away from the pie chart.

```
\newcommand*{\dtl@piecutaways}{}
```

\dtl@innerlabel \dtl@innerlabel specifies the label to appear inside the segment. By default this is the variable used to create the pie chart.

```
\def\dtl@innerlabel{\DTLpievariable}%
```

\dtl@outerlabel

```
\def\dtl@outerlabel{}%
```

DTLpieroundvar is a counter governing the number of digits to round to when using \FPround.

```
\newcounter{DTLpieroundvar}
\setcounter{DTLpieroundvar}{1}
```

\DTLdisplayinnerlabel

\DTLdisplayinnerlabel{⟨*label*⟩}

This is used to format the inner label. This just does the label by default.

```
\newcommand*{\DTLdisplayinnerlabel}[1]{#1}
```

\DTLdisplayouterlabel

\DTLdisplayouterlabel{⟨*label*⟩}

This is used to format the outer label. This just does the label by default.

```
\newcommand*{\DTLdisplayouterlabel}[1]{#1}
```

\DTLpiepercent

\DTLpiepercent returns the percentage value of the current segment.

```
\newcommand*{\DTLpiepercent}{%
\ifnum\dtlforeachlevel=0\relax
  \PackageError{datapie}{Can't use
  \string\DTLpiepercent\space outside
  \string\DTLpiechart}{}%
\else
  \csname dtl@piepercent@\romannumeral\@dtl@seg\endcsname
\fi}
```

\DTLpieatbegintikz

\DTLpieatbegintikz specifies any commands to apply at the start of the tikzpicture environment. By default it does nothing.

```
\newcommand*{\DTLpieatbegintikz}{}
```

\DTLpieatendtikz

\DTLpieatendtikz specifies any commands to apply at the end of the tikzpicture environment. By default it does nothing.

```
\newcommand*{\DTLpieatendtikz}{}
```

\DTLsetpiesegmentcolor

\DTLsetpiesegmentcolor{⟨*n*⟩}{⟨*color*⟩}

Assign colour name ⟨*color*⟩ to the ⟨*n*⟩th segment.

```
\newcommand*{\DTLsetpiesegmentcolor}[2]{%
\expandafter\def\csname dtlpie@segcol\romannumeral#1\endcsname{#2}%
}
```

\DTLgetpiesegmentcolor

\DTLgetpiesegmentcolor{⟨*n*⟩}

Get the colour specification for segment ⟨*n*⟩

```
\newcommand*{\DTLgetpiesegmentcolor}[1]{%
\csname dtlpie@segcol\romannumeral#1\endcsname}
```

\DTLdopiesegmentcolor

\DTLdopiesegmentcolor{⟨n⟩}

Set the colour to that for segment ⟨n⟩

```
\newcommand*{\DTLdopiesegmentcolor}[1]{%
\expandafter\color\expandafter
{\csname dtlpie@segcol\romannumeral#1\endcsname}}
```

\DTLdocurrentpiesegmentcolor

\DTLdocurrentpiesegmentcolor sets the colour to that of the current segment.

```
\newcommand*{\DTLdocurrentpiesegmentcolor}{%
\ifnum\dtlforeachlevel=0\relax
  \PackageError{datapie}{Can't use
  \string\DTLdocurrentpiesegmentcolor\space outside
  \string\DTLpiechart}{}%
\else
  \expandafter\DTLdopiesegmentcolor\expandafter{%
  \csname c@DTLrow\romannumeral\dtlforeachlevel\endcsname}%
\fi}
```

\DTLpieoutlinecolor

\DTLpieoutlinecolor specifies what colour to draw the outline.

```
\newcommand*{\DTLpieoutlinecolor}{black}
```

\DTLpieoutlinewidth

\DTLpieoutlinewidth specifies the line width of the outline: Outline is only drawn if the linewidth is greater than 0pt.

```
\newlength\DTLpieoutlinewidth
\DTLpieoutlinewidth=0pt
```

Set the default colours. If there are more than eight segments, more colours will need to be defined.

```
\ifDTLcolorpiechart
 \DTLsetpiesegmentcolor{1}{red}
 \DTLsetpiesegmentcolor{2}{green}
 \DTLsetpiesegmentcolor{3}{blue}
 \DTLsetpiesegmentcolor{4}{yellow}
 \DTLsetpiesegmentcolor{5}{magenta}
 \DTLsetpiesegmentcolor{6}{cyan}
 \DTLsetpiesegmentcolor{7}{orange}
 \DTLsetpiesegmentcolor{8}{white}
\else
 \DTLsetpiesegmentcolor{1}{black!15}
 \DTLsetpiesegmentcolor{2}{black!25}
 \DTLsetpiesegmentcolor{3}{black!35}
 \DTLsetpiesegmentcolor{4}{black!45}
 \DTLsetpiesegmentcolor{5}{black!55}
 \DTLsetpiesegmentcolor{6}{black!65}
 \DTLsetpiesegmentcolor{7}{black!75}
 \DTLsetpiesegmentcolor{8}{black!85}
\fi
```

Define keys for \DTLpiechart optional argument. Set the starting angle of the first segment.

```
\define@key{datapie}{start}{\def\DTLstartangle{#1}}
```

Set the radius of the pie chart (must be set prior to inneroffset and outeroffset keys.)

```
\define@key{datapie}{radius}{\DTLradius=#1\relax
\dtl@inneroffset=\DTLinnerratio\DTLradius
\dtl@outeroffset=\DTLouterratio\DTLradius
\dtl@cutawayoffset=\DTLcutawayratio\DTLradius}
```

Set the inner ratio.

```
\define@key{datapie}{innerratio}{%
\def\DTLinnerratio{#1}%
\dtl@inneroffset=\DTLinnerratio\DTLradius}
```

Set the outer ratio

```
\define@key{datapie}{outerratio}{%
\def\DTLouterratio{#1}%
\dtl@outeroffset=\DTLouterratio\DTLradius}
```

The cutaway offset ratio

```
\define@key{datapie}{cutawayratio}{%
\def\DTLcutawayratio{#1}%
\dtl@cutawayoffset=\DTLcutawayratio\DTLradius}
```

Set the inner offset as an absolute value (not dependent on the radius.)

```
\define@key{datapie}{inneroffset}{%
\dtl@inneroffset=#1}
```

Set the outer offset as an absolute value (not dependent on the radius.)

```
\define@key{datapie}{outeroffset}{%
\dtl@outeroffset=#1}
```

Set the cutaway offset as an absolute value (not dependent on the radius.)

```
\define@key{datapie}{cutawayoffset}{%
\dtl@cutawayoffset=#1}
```

List of cut away segments.

```
\define@key{datapie}{cutaway}{%
\renewcommand*{\dtl@piecutaways}{#1}}
```

Variable used to create the pie chart. (Must be a control sequence.)

```
\define@key{datapie}{variable}{%
\def\DTLpievariable{#1}}
```

Inner label

```
\define@key{datapie}{innerlabel}{%
\def\dtl@innerlabel{#1}}
```

Outer label

```
\define@key{datapie}{outerlabel}{%
\def\dtl@outerlabel{#1}}
```

\DTLpiechart

\DTLpiechart[⟨*conditions*⟩]{⟨*option list*⟩}{⟨*db name*⟩}{⟨*assign list*⟩}

Make a pie chart from data given in data base ⟨*db name*⟩, where ⟨*assign list*⟩ is a comma-separated list of ⟨*cmd*⟩=⟨*key*⟩ pairs. ⟨*option list*⟩ must include `variable=`⟨*cmd*⟩, where ⟨*cmd*⟩ is included in ⟨*assign list*⟩. The optional argument ⟨*conditions*⟩ is the same as that for `\DTLforeach`.

```
\newcommand*{\DTLpiechart}[4][\boolean{true}]{%
{\let\DTLpievariable=\relax
\setkeys{datapie}{#2}%
\ifx\DTLpievariable\relax
  \PackageError{datapie}{\string\DTLpiechart\space missing variable}{}%
\else
```

Compute the total.

```
\def\dtl@total{0}%
\@sDTLforeach[#1]{#3}{#4}{%
\let\dtl@oldtotal=\dtl@total
\expandafter\DTLconverttodecimal\expandafter
  {\DTLpievariable}{\dtl@variable}%
\FPadd{\dtl@total}{\dtl@variable}{\dtl@total}%
}%
```

Compute the angles

```
\expandafter\DTLconverttodecimal\expandafter
  {\DTLstartangle}{\@dtl@start}%
\@sDTLforeach[#1]{#3}{#4}{%
\expandafter\DTLconverttodecimal\expandafter
  {\DTLpievariable}{\dtl@variable}%
\dtl@computeangles{%
\csname c@DTLrow\romannumeral\dtlforeachlevel\endcsname}{%
\dtl@variable}%
\expandafter\@dtl@seg\expandafter=
\csname c@DTLrow\romannumeral\dtlforeachlevel\endcsname%
\FPmul{\dtl@tmp}{\dtl@variable}{100}%
\let\dtl@old=\dtl@tmp
\FPdiv{\dtl@tmp}{\dtl@old}{\dtl@total}%
\expandafter\FPround
\csname dtl@piepercent@\romannumeral\@dtl@seg\endcsname\dtl@tmp
\c@DTLpieroundvar
}%
```

Compute the offsets for each cut away segment

```
\@for\dtl@row:=\dtl@piecutaways\do{%
\expandafter\@dtl@set@off\dtl@row-\relax
}%
```

Set the starting angle

```
\let\dtl@start=\DTLstartangle
```

Do the pie chart

```
\begin{tikzpicture}
\DTLpieatbegintikz
\@sDTLforeach[#1]{#3}{#4}{%
```

Store the segment number in `\@dtl@seg`

```
\expandafter\@dtl@seg\expandafter=
\csname c@DTLrow\romannumeral\dtlforeachlevel\endcsname%
```

Set the start angle.

```
\edef\dtl@start{\csname dtl@sang@\romannumeral\@dtl@seg\endcsname}%
```

Set the extent

```
\edef\dtl@extent{\csname dtl@angle@\romannumeral\@dtl@seg\endcsname}%
```

Compute the end angle

```
\FPadd{\dtl@endangle}{\dtl@start}{\dtl@extent}%
```

Compute the shift.

```
\edef\dtl@angle{\csname dtl@cut@angle@\romannumeral\@dtl@seg\endcsname}%
\let\dtl@old=\dtl@angle
\dtl@truncatedecimal\dtl@angle
\ifnum\dtl@angle>180
  \FPsub{\dtl@angle}{\dtl@old}{360}%
  \dtl@truncatedecimal\dtl@angle
\fi
\edef\dtl@cutlen{%
\csname dtl@cut@len@\romannumeral\@dtl@seg\endcsname}
\edef\@dtl@shift{(\dtl@angle:\dtl@cutlen)}%
```

Compute the mid way angle.

```
\FPmul{\dtl@angle}{\dtl@extent}{0.5}%
\FPadd{\dtl@midangle}{\dtl@angle}{\dtl@start}%
```

Draw the segment.

```
\begin{scope}[shift={\@dtl@shift}]%
\dtl@truncatedecimal\dtl@start
\dtl@truncatedecimal\dtl@endangle
\fill[color=\DTLgetpiesegmentcolor\@dtl@seg] (0,0) --
(\dtl@start:\DTLradius)
arc (\dtl@start:\dtl@endangle:\DTLradius) -- cycle;
```

Draw the outline if required:

```
\ifdim\DTLpieoutlinewidth>0pt\relax
\draw[color=\DTLpieoutlinecolor,line width=\DTLpieoutlinewidth]
(0,0) -- (\dtl@start:\DTLradius)
arc (\dtl@start:\dtl@endangle:\DTLradius) -- cycle;
\fi
```

Convert decimal to an integer

```
\dtl@truncatedecimal\dtl@midangle
```

Determine whether to rotate inner labels

```
\ifDTLrotateinner
```

If the mid way angle is between 90 and 270, the text will look upside-down, so adjust accordingly.

```
\ifthenelse{\(\dtl@midangle > 90 \and \dtl@midangle < 270\)
\TE@or \dtl@midangle < -90}{%
  \FPsub{\dtl@labelangle}{\dtl@midangle}{180}%
  \dtl@truncatedecimal\dtl@labelangle
  \edef\dtl@innernodeopt{anchor=east,rotate=\dtl@labelangle}%
}{%
  \edef\dtl@innernodeopt{anchor=west,rotate=\dtl@midangle}%
}%
```

Don't rotate inner labels

```
\else
  \edef\dtl@innernodeopt{anchor=center}%
\fi
```

Determine whether to rotate outer labels

```
\ifDTLrotateouter
```

If the mid way angle is between 90 and 270, the text will look upside-down, so adjust accordingly.

```
\ifthenelse{\(\dtl@midangle > 90 \and \dtl@midangle < 270\)
\TE@or \dtl@midangle < -90}{%
  \FPsub{\dtl@labelangle}{\dtl@midangle}{180}%
  \dtl@truncatedecimal\dtl@labelangle
  \edef\dtl@outernodeopt{anchor=east,rotate=\dtl@labelangle}%
}{%
  \edef\dtl@outernodeopt{anchor=west,rotate=\dtl@midangle}%
}%
```

Don't rotate outer labels

```
\else
  \ifthenelse{\(\dtl@midangle<45\and\dtl@midangle>-45\)
    \TE@or \dtl@midangle=45
    \TE@or \dtl@midangle>315}{%
    % east quadrant
    \edef\dtl@outernodeopt{anchor=west}%
  }{%
    \ifthenelse{\(\dtl@midangle<135\and\dtl@midangle>45\)
     \TE@or \dtl@midangle=135}{%
      % north quadrant
      \edef\dtl@outernodeopt{anchor=south}%
    }{%
     \ifthenelse{\(\dtl@midangle<225\and\dtl@midangle>135\)
      \TE@or \dtl@midangle=225
      \TE@or \dtl@midangle=-135
      \TE@or \dtl@midangle<-135}{%
      % west quadrant
      \edef\dtl@outernodeopt{anchor=east}%
     }{%
      \edef\dtl@outernodeopt{anchor=north}%
     }%
    }
  }
\fi
```

Draw inner and outer labels

```
\edef\@dtl@dolabel{%
\noexpand\draw (\dtl@midangle:\the\dtl@inneroffset)
node[\dtl@innernodeopt]{%
\noexpand\DTLdisplayinnerlabel{\noexpand\dtl@innerlabel}};}%
\@dtl@dolabel
\edef\@dtl@dolabel{%
\noexpand\draw (\dtl@midangle:\the\dtl@outeroffset)
node[\dtl@outernodeopt]{%
\noexpand\DTLdisplayouterlabel{\noexpand\dtl@outerlabel}};}%
```

```
\@dtl@dolabel
\end{scope}
}%
\DTLpieatendtikz
\end{tikzpicture}
\fi
}}
```

\dtl@computeangles

\dtl@computeangles{⟨*n*⟩}{⟨*variable*⟩}

Compute the angles for segment ⟨*n*⟩. This sets \dtl@sang@⟨*n*⟩ (start angle), \dtl@angle@⟨*n*⟩ (extent angle), \dtl@cut@angle@⟨*n*⟩ (cut away angle) and \dtl@cut@len@⟨*n*⟩ (cut away length).

```
\newcommand*{\dtl@computeangles}[2]{%
\FPifgt{\@dtl@start}{180}%
% if startangle > 180
  \let\dtl@old=\@dtl@start
% startangle = startangle - 360
  \FPsub{\@dtl@start}{\dtl@old}{360}%
\fi
\FPiflt{\@dtl@start}{-180}%
% if startangle < -180
  \let\dtl@old=\@dtl@start
% startangle = startangle + 360
  \FPadd{\@dtl@start}{\dtl@old}{360}%
\fi
\expandafter\edef\csname dtl@sang@\romannumeral#1\endcsname{%
\@dtl@start}%
\FPmul{\dtl@angle}{360}{#2}%
\let\dtl@old=\dtl@angle
\FPdiv{\dtl@angle}{\dtl@old}{\dtl@total}%
\expandafter\let\csname dtl@angle@\romannumeral#1\endcsname=\dtl@angle
\let\dtl@old=\@dtl@start
\FPadd{\@dtl@start}{\dtl@old}{\dtl@angle}%
\expandafter\def\csname dtl@cut@angle@\romannumeral#1\endcsname{0}%
\expandafter\def\csname dtl@cut@len@\romannumeral#1\endcsname{0cm}%
}
```

Set the offset angles.

\@dtl@set@off

```
\def\@dtl@set@off#1-#2\relax{%
\ifthenelse{\equal{#2}{}}{%
\@@dtl@set@off{#1}}{%
\@@dtl@set@offr#1-#2\relax}%
}
```

Set offset for individual segment:

\@@dtl@set@off

```
\newcommand*{\@@dtl@set@off}[1]{%
\edef\dtl@old{\csname dtl@angle@\romannumeral#1\endcsname}%
\FPmul{\dtl@angle}{\dtl@old}{0.5}%
```

```
\let\dtl@old=\dtl@angle
\edef\dtl@sang{\csname dtl@sang@\romannumeral#1\endcsname}%
\FPadd{\dtl@angle}{\dtl@old}{\dtl@sang}%
\expandafter\edef\csname dtl@cut@angle@\romannumeral#1\endcsname{%
\dtl@angle}%
\expandafter\edef\csname dtl@cut@len@\romannumeral#1\endcsname{%
  \the\dtl@cutawayoffset}
}
```

Define count register to keep track of segments

`\@dtl@seg`

```
\newcount\@dtl@seg
```

`\@@dtl@setoffr` Set offset for a range of segments

```
\def\@@dtl@set@offr#1-#2-\relax{%
\ifnum#1>#2\relax
 \PackageError{datapie}{Segment ranges must go in ascending order}{%
Try #2-#1 instead of #1-#2}%
\else
 \def\dtl@angle{0}%
 \@dtl@seg=#1\relax
 \whiledo{\not\(\@dtl@seg > #2\)}{%
  \let\dtl@old=\dtl@angle
  \edef\dtl@segang{\csname dtl@angle@\romannumeral\@dtl@seg\endcsname}%
  \FPadd{\dtl@angle}{\dtl@old}{\dtl@segang}%
  \advance\@dtl@seg by 1\relax
 }%
 \let\dtl@old=\dtl@angle
 \FPmul{\dtl@angle}{\dtl@old}{0.5}%
 \edef\dtl@sang{\csname dtl@sang@\romannumeral#1\endcsname}%
 \let\dtl@old=\dtl@angle
 \FPadd{\dtl@angle}{\dtl@old}{\dtl@sang}%
 \@dtl@seg=#1\relax
 \whiledo{\not\(\@dtl@seg > #2\)}{%
 \expandafter
  \let\csname dtl@cut@angle@\romannumeral\@dtl@seg\endcsname
  =\dtl@angle
 \expandafter
  \edef\csname dtl@cut@len@\romannumeral\@dtl@seg\endcsname{%
  \the\dtl@cutawayoffset}
  \advance\@dtl@seg by 1\relax
 }%
\fi
}
```

12 dataplot.sty

Declare package:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{dataplot}[2009/02/27 v2.0 (NLCT)]
```

Required packages

```
\RequirePackage{xkeyval}
\RequirePackage{tikz}
\RequirePackage{datatool}
```

Load TikZ plot libraries

```
\usetikzlibrary{plotmarks}
\usetikzlibrary{plothandlers}
```

\DTLplotstream

\DTLplotstream[⟨*condition*⟩]{⟨*db name*⟩}{⟨*x key*⟩}{⟨*y key*⟩}

Add points to a stream from the database called ⟨*db name*⟩ where the x co-ordinates are given by the key ⟨*x key*⟩ and the y co-ordinates are given by the key ⟨*y key*⟩. The optional argument ⟨*condition*⟩ is the same as that for \DTLforeach

```
\newcommand*{\DTLplotstream}[4][\boolean{true}]{%
\@sDTLforeach[#1]{#2}{\dtl@x=#3,\dtl@y=#4}{%
\DTLconverttodecimal{\dtl@x}{\dtl@decx}%
\DTLconverttodecimal{\dtl@y}{\dtl@decy}%
\pgfplotstreampoint{\pgfpointxy{\dtl@decx}{\dtl@decy}}}}
```

\DTLplotmarks

\DTLplotmarks contains a list of plot marks used by \DTLplot.

```
\newcommand*{\DTLplotmarks}{%
  \pgfuseplotmark{o},%
  \pgfuseplotmark{x},%
  \pgfuseplotmark{+},%
  \pgfuseplotmark{square},%
  \pgfuseplotmark{triangle},%
  \pgfuseplotmark{diamond},%
  \pgfuseplotmark{pentagon},%
  \pgfuseplotmark{asterisk},%
  \pgfuseplotmark{star}%
}
```

\DTLplotmarkcolors

\DTLplotmarkcolors contains a list of the plot mark colours.

```
\newcommand*{\DTLplotmarkcolors}{%
  red,%
  green,%
  blue,%
  yellow,%
  magenta,%
  cyan,%
  orange,%
  black,%
  gray}
```

\DTLplotlines

\DTLplotlines contains a list of dash patterns used by \DLTplot.

```
\newcommand*{\DTLplotlines}{%
  \pgfsetdash{}{0pt},% solid line
  \pgfsetdash{{10pt}{5pt}}{0pt},%
  \pgfsetdash{{5pt}{5pt}}{0pt},%
  \pgfsetdash{{1pt}{5pt}}{0pt},%
  \pgfsetdash{{5pt}{5pt}{1pt}{5pt}}{0pt},%
  \pgfsetdash{{1pt}{3pt}}{0pt},%
}
```

`\DTLplotlinecolors` `\DTLplotlinecolors` contains a list of the plot line colours.

```
\newcommand*{\DTLplotlinecolors}{%
  red,%
  green,%
  blue,%
  yellow,%
  magenta,%
  cyan,%
  orange,%
  black,%
  gray}
```

`\DTLplotwidth` The default total plot width is stored in the length `\dtlplotwidth`

```
\newlength\DTLplotwidth
\setlength\DTLplotwidth{4in}
```

`\DTLplotheight` The default total plot height is stored in the length `\dtlplotheight`

```
\newlength\DTLplotheight
\setlength\DTLplotheight{4in}
```

`\DTLticklength` The length of the tick marks is given by `\DTLticklength`

```
\newlength\DTLticklength
\setlength\DTLticklength{5pt}
```

`\DTLminorticklength` The length of the minor tick marks is given by `\DTLminorticklength`.

```
\newlength\DTLminorticklength
\setlength\DTLminorticklength{2pt}
```

`\DTLticklabeloffset` The offset from the axis to the tick label is given by `\DTLticklabeloffset`.

```
\newlength\DTLticklabeloffset
\setlength\DTLticklabeloffset{8pt}
```

`\dtl@xticlabelheight` `\dtl@xticlabelheight` is used to store the height of the x tick labels.

```
\newlength\dtl@xticlabelheight
```

`\dtl@yticlabelwidth` `\dtl@yticlabelwidth` is used to store the width of the y tick labels.

```
\newlength\dtl@yticlabelwidth
```

`\DTLmintickgap` `\DTLmintickgap` stores the suggested minimum distance between tick marks where the gap is not specified.

```
\newlength\DTLmintickgap
\setlength\DTLmintickgap{20pt}
```

`\DTLminminortickgap` The suggested minimum distance between minor tick marks where the gap is not specified is given by `\DTLminminortickgap`.

```
\newlength\DTLminminortickgap
\setlength\DTLminminortickgap{5pt}
```

Round x tick labels to the number of digits given by the counter `DTLplotroundXvar`.

```
\newcounter{DTLplotroundXvar}
\setcounter{DTLplotroundXvar}{2}
```

Round y tick labels to the number of digits given by the counter `DTLplotroundYvar`.

```
\newcounter{DTLplotroundYvar}
\setcounter{DTLplotroundYvar}{2}
```

`\ifDTLxaxis` The conditional `\ifDTLxaxis` is used to determine whether or not to display the x axis.

```
\newif\ifDTLxaxis
\DTLxaxistrue
```

`\DTLXAxisStyle` The style of the x axis is given by `\DTLXAxisStyle`. This is just a solid line by default.

```
\newcommand*{\DTLXAxisStyle}{-}
```

`\ifDTLyaxis` The conditional `\ifDTLyaxis` is used to determine whether or not to display the y axis

```
\newif\ifDTLyaxis
\DTLyaxistrue
```

`\DTLYAxisStyle` The style of the y axis is given by `\DTLYAxisStyle`. This is just a solid line by default.

```
\newcommand*{\DTLYAxisStyle}{-}
```

`\DTLmajorgridstyle` The style of the major grid lines is given by `\DTLmajorgridstyle`.

```
\newcommand*{\DTLmajorgridstyle}{color=gray,-}
```

`\DTLminorgridstyle` The style of the minor grid lines is given by `\DTLminorgridstyle`.

```
\newcommand*{\DTLminorgridstyle}{color=gray,loosely dotted}
```

`\ifDTLxticsin` The conditional `\ifDTLxticsin` is used to determine whether the x tics should point in or out.

```
\newif\ifDTLxticsin
\DTLxticsintrue
```

`\ifDTLyticsin` The conditional `\ifDTLyticsin` is used to determine whether the y tics should point in or out.

```
\newif\ifDTLyticsin
\DTLyticsintrue
```

`\dtl@legendsetting` The legend setting is stored in the count register `\dtl@legendsetting`.

```
\newcount\dtl@legendsetting
```

`\DTLlegendxoffset` The gap between the border of plot and legend is given by the lengths `\DTLlegendxoffset` and `\DTLlegendyoffset`

```
\newlength\DTLlegendxoffset
\setlength\DTLlegendxoffset{10pt}
```

`\DTLlegendyoffset`

```
\newlength\DTLlegendyoffset
\setlength\DTLlegendyoffset{10pt}
```

\DTLformatlegend

\DTLformatlegend{⟨*legend*⟩}

This formats the legend.

```
\newcommand*{\DTLformatlegend}[1]{%
\setlength{\fboxrule}{1.1pt}%
\fcolorbox{black}{white}{#1}}
```

\ifDTLshowmarkers

The conditional \ifDTLshowmarkers is used to specify whether or not to use markers.

```
\newif\ifDTLshowmarkers
\DTLshowmarkerstrue
```

\ifDTLshowlines

The conditional \ifDTLshowlines is used to specify whether or not to use lines.

```
\newif\ifDTLshowlines
\DTLshowlinesfalse
```

\DTLplotatbegintikz

\DTLplotatbegintikz is a hook to insert stuff at the start of the tikzpicture environment (after the unit vectors have been set).

```
\newcommand*{\DTLplotatbegintikz}{}
```

\DTLplotatendtikz

\DTLplotatendtikz is a hook to insert stuff at the end of the tikzpicture environment.

```
\newcommand*{\DTLplotatendtikz}{}
```

Plot settings. The database key for the x value is given by the x setting:

```
\define@key{dataplot}{x}{%
\def\dtl@xkey{#1}}
```

The database key for the y value is given by the y setting:

```
\define@key{dataplot}{y}{%
\def\dtl@ykey{#1}}
```

The list of plot mark colours is given by the markcolors setting. (This should be a comma separated list of colour names.)

```
\define@key{dataplot}{markcolors}{%
\def\DTLplotmarkcolors{#1}}
```

The list of plot line colours is given by the linecolors setting. (This should be a comma separated list of colour names.)

```
\define@key{dataplot}{linecolors}{%
\def\DTLplotlinecolors{#1}}
```

The list of plot mark and line colours is given by the colors setting. (This should be a comma separated list of colour names.)

```
\define@key{dataplot}{colors}{%
\def\DTLplotmarkcolors{#1}%
\def\DTLplotlinecolors{#1}}
```

The list of plot marks is given by the marks setting. (This should be a comma separated list of code that generates pgf plot marks.)

```
\define@key{dataplot}{marks}{%
\def\DTLplotmarks{#1}}
```

The list of plot line styles is given by the `lines` setting. (This should be a comma separated list of code that sets the line style.) An empty set will create solid lines.

```
\define@key{dataplot}{lines}{%
\def\DTLplotlines{#1}}
```

The total width of the plot is given by the `width` setting.

```
\define@key{dataplot}{width}{%
\setlength\DTLplotwidth{#1}}
```

The total height of the plot is given by the `height` setting.

```
\define@key{dataplot}{height}{%
\setlength\DTLplotheight{#1}}
```

Determine whether to show lines, markers or both

```
\define@choicekey{dataplot}{style}[\val\nr]{both,lines,markers}{%
\ifcase\nr\relax
 \DTLshowlinestrue
 \DTLshowmarkerstrue
\or
 \DTLshowlinestrue
 \DTLshowmarkersfalse
\or
 \DTLshowmarkerstrue
 \DTLshowlinesfalse
\fi}
```

Determine whether or not to display the axes

```
\define@choicekey{dataplot}{axes}[\val\nr]{both,x,y,none}[both]{%
\ifcase\nr\relax
 % both
 \DTLxaxistrue
 \DTLxticstrue
 \DTLyaxistrue
 \DTLyticstrue
\or % x
 \DTLxaxistrue
 \DTLxticstrue
 \DTLyaxisfalse
 \DTLyticsfalse
\or % y
 \DTLxaxisfalse
 \DTLxticsfalse
 \DTLyaxistrue
 \DTLyticstrue
\or % none
 \DTLxaxisfalse
 \DTLxticsfalse
 \DTLyaxisfalse
 \DTLyticsfalse
\fi
}
```

`\ifDTLbox` Enclose plot in a box

```
\define@boolkey{dataplot}[DTL]{box}[true]{}
\DTLboxfalse
```

`\ifDTLxticstrue` Condition to determine whether to show the x tick marks

```
\define@boolkey{dataplot}[DTL]{xtics}[true]{}
\DTLxticstrue
```

`\ifDTLyticstrue` Condition to determine whether to show the y tick marks

```
\define@boolkey{dataplot}[DTL]{ytics}[true]{}
\DTLyticstrue
```

`\ifDTLxminortics` Condition to determine whether to show the x minor tick marks

```
\define@boolkey{dataplot}[DTL]{xminortics}[true]{%
\ifDTLxminortics \DTLxticstrue\fi}
\DTLxminorticsfalse
```

`\ifDTLyminortics` Condition to determine whether to show the y minor tick marks

```
\define@boolkey{dataplot}[DTL]{yminortics}[true]{%
\ifDTLyminortics \DTLyticstrue\fi}
\DTLyminorticsfalse
```

`\ifDTLgrid` Determine whether to draw the grid

```
\define@boolkey{dataplot}[DTL]{grid}[true]{}
```

Determine whether the x tick marks should point in or out:

```
\define@choicekey{dataplot}{xticdir}[\val\nr]{in,out}{%
\ifcase\nr\relax
 \DTLxticsintrue
\or
 \DTLxticsinfalse
\fi
}
```

Determine whether the y tick marks should point in or out:

```
\define@choicekey{dataplot}{yticdir}[\val\nr]{in,out}{%
\ifcase\nr\relax
 \DTLyticsintrue
\or
 \DTLyticsinfalse
\fi
}
```

Determine whether the x and y tick marks should point in or out;

```
\define@choicekey{dataplot}{ticdir}[\val\nr]{in,out}{%
\ifcase\nr\relax
 \DTLxticsintrue
 \DTLyticsintrue
\or
 \DTLxticsinfalse
 \DTLyticsinfalse
\fi
}
```

Set the bounds of the graph (value must be in the form $\langle min\ x\rangle$,$\langle min\ y\rangle$,$\langle max\ x\rangle$,$\langle max\ y\rangle$ (bounds overrides minx, miny, maxx and maxy settings.)

```
\define@key{dataplot}{bounds}{%
\def\dtl@bounds{#1}}
\let\dtl@bounds=\relax
```

Set only the lower x bound

```
\define@key{dataplot}{minx}{%
\def\dtl@minx{#1}}
\let\dtl@minx=\relax
```

Set only the upper x bound:

```
\define@key{dataplot}{maxx}{%
\def\dtl@maxx{#1}}
\let\dtl@maxx=\relax
```

Set only the lower y bound:

```
\define@key{dataplot}{miny}{%
\def\dtl@miny{#1}}
\let\dtl@miny=\relax
```

Set only the upper y bound:

```
\define@key{dataplot}{maxy}{%
\def\dtl@maxy{#1}}
\let\dtl@maxy=\relax
```

Define list of points for x ticks. (Must be a comma separated list of decimal numbers.)

```
\define@key{dataplot}{xticpoints}{%
\def\dtl@xticlist{#1}\DTLxticstrue\DTLxaxistrue}
\let\dtl@xticlist=\relax
```

Define list of points for y ticks. (Must be a comma separated list of decimal numbers.)

```
\define@key{dataplot}{yticpoints}{%
\def\dtl@yticlist{#1}\DTLyticstrue\DTLyaxistrue}
\let\dtl@yticlist=\relax
```

Define a the gap between x tick marks (xticpoints overrides xticgap)

```
\define@key{dataplot}{xticgap}{\def\dtl@xticgap{#1}%
\DTLxticstrue\DTLxaxistrue}
\let\dtl@xticgap=\relax
```

Define a the gap between y tick marks (yticpoints overrides yticgap)

```
\define@key{dataplot}{yticgap}{\def\dtl@yticgap{#1}%
\DTLyticstrue\DTLyaxistrue}
\let\dtl@yticgap=\relax
```

Define comma separated list of labels for x ticks.

```
\define@key{dataplot}{xticlabels}{%
\def\dtl@xticlabels{#1}\DTLxticstrue\DTLxaxistrue}
\let\dtl@xticlabels=\relax
```

Define comma separated list of labels for y ticks.

```
\define@key{dataplot}{yticlabels}{%
\def\dtl@yticlabels{#1}\DTLyticstrue\DTLyaxistrue}
\let\dtl@yticlabels=\relax
```

Define x axis label

```
\define@key{dataplot}{xlabel}{%
\def\dtl@xlabel{#1}}
\let\dtl@xlabel=\relax
```

Define y axis label

```
\define@key{dataplot}{ylabel}{%
\def\dtl@ylabel{#1}}
\let\dtl@ylabel=\relax
```

The legend setting may be one of: `none` (don't show it), `north`, `northeast`, `east`, `southeast`, `south`, `southwest`, `west`, or `northwest`. These set the count register `\dtl@legendsetting`.

```
\define@choicekey{dataplot}{legend}[\val\nr]{none,north,northeast,%
east,southeast,south,southwest,west,northwest}[northeast]{%
\dtl@legendsetting=\nr\relax
}
```

Legend labels (comma separated list). If omitted, the database name is used.

```
\define@key{dataplot}{legendlabels}{\def\dtl@legendlabels{#1}}
```

`\DTLplot` `\DTLplot[`⟨*condition*⟩`]{`⟨*db list*⟩`}{`⟨*settings*⟩`}`

Creates a plot (inside a tikzpicture environment) of all the data given in the databases listed in ⟨*db list*⟩.

```
\newcommand*{\DTLplot}[3][\boolean{true}]{{%
\let\dtl@xkey=\relax
\let\dtl@ykey=\relax
\let\dtl@legendlabels=\relax
\setkeys{dataplot}{#3}%
\let\dtl@plotmarklist=\DTLplotmarks
\let\dtl@plotlinelist=\DTLplotlines
\let\dtl@plotmarkcolorlist=\DTLplotmarkcolors
\let\dtl@plotlinecolorlist=\DTLplotlinecolors
\def\dtl@legend{}%
\ifx\dtl@legendlabels\relax
 \edef\dtl@legendlabels{#2}%
\fi
\ifx\dtl@xkey\relax
  \PackageError{dataplot}{Missing x setting for
    \string\DTLplot}{}%
\else
  \ifx\dtl@ykey\relax
    \PackageError{dataplot}{Missing y setting for
      \string\DTLplot}{}%
  \else
```

If user didn't specified bounds, compute the maximum and minimum x and y values over all the databases listed.

```
    \ifx\dtl@bounds\relax
      \DTLcomputebounds[#1]{#2}{\dtl@xkey}{\dtl@ykey}
        {\DTLminX}{\DTLminY}{\DTLmaxX}{\DTLmaxY}%
      \ifx\dtl@minx\relax
      \else
         \let\DTLminX=\dtl@minx
      \fi
      \ifx\dtl@maxx\relax
      \else
```

```
      \let\DTLmaxX=\dtl@maxx
    \fi
    \ifx\dtl@miny\relax
    \else
      \let\DTLminY=\dtl@miny
    \fi
    \ifx\dtl@maxy\relax
    \else
      \let\DTLmaxY=\dtl@maxy
    \fi
```

Otherwise extract information from `\dtl@bounds`

```
  \else
    \expandafter\dtl@getbounds\dtl@bounds\@nil
  \fi
```

Determine scaling factors.

```
  \@dtl@tmpcount=\DTLplotwidth
  \FPsub{\dtl@dx}{\DTLmaxX}{\DTLminX}%
  \FPdiv{\dtl@unit@x}{\number\@dtl@tmpcount}{\dtl@dx}%
  \@dtl@tmpcount=\DTLplotheight
  \FPsub{\dtl@dy}{\DTLmaxY}{\DTLminY}%
  \FPdiv{\dtl@unit@y}{\number\@dtl@tmpcount}{\dtl@dy}%
```

If x tics specified, construct a list of x tic points if not already specified.

```
  \ifDTLxtics
    \ifx\dtl@xticlist\relax
      \ifx\dtl@xticgap\relax
        \dtl@constructticklist\DTLminX\DTLmaxX
          \dtl@unit@x\dtl@xticlist
      \else
        \DTLifFPopenbetween{0}{\DTLminX}{\DTLmaxX}{%
          \dtl@constructticklistwithgapz
            \DTLminX\DTLmaxX\dtl@xticlist\dtl@xticgap}{%
          \dtl@constructticklistwithgap
            \DTLminX\DTLmaxX\dtl@xticlist\dtl@xticgap}%
      \fi
    \fi
```

Construct a list of x minor tick points if required

```
    \let\dtl@xminorticlist\@empty
    \ifDTLxminortics
      \let\dtl@prevtick=\relax
      \@for\dtl@nexttick:=\dtl@xticlist\do{%
        \ifx\dtl@prevtick\relax
        \else
          \dtl@constructminorticklist
              \dtl@prevtick\dtl@nexttick\dtl@unit@x\dtl@xminorticlist
        \fi
        \let\dtl@prevtick=\dtl@nexttick
      }%
    \fi
```

Determine the height of the x tick labels.

```
    \ifx\dtl@xticlabels\relax
      \settoheight{\dtl@xticlabelheight}{\dtl@xticlist}%
```

```
    \else
      \settoheight{\dtl@xticlabelheight}{\dtl@xticlabels}%
     \fi
  \else
    \setlength{\dtl@xticlabelheight}{0pt}%
  \fi
```

If y tics specified, construct a list of y tic points if not already specified.

```
  \setlength{\dtl@yticlabelwidth}{0pt}%
  \ifDTLytics
    \ifx\dtl@yticlist\relax
      \ifx\dtl@yticgap\relax
      \dtl@constructticklist\DTLminY\DTLmaxY
        \dtl@unit@y\dtl@yticlist
      \else
        \DTLifFPopenbetween{0}{\DTLminY}{\DTLmaxY}{%
          \dtl@constructticklistwithgapz
            \DTLminY\DTLmaxY\dtl@yticlist\dtl@yticgap}{%
          \dtl@constructticklistwithgap
            \DTLminY\DTLmaxY\dtl@yticlist\dtl@yticgap}%
      \fi
    \fi
```

Construct a list of y minor tick points if required

```
    \let\dtl@yminorticlist\@empty
    \ifDTLyminortics
      \let\dtl@prevtick=\relax
      \@for\dtl@nexttick:=\dtl@yticlist\do{%
        \ifx\dtl@prevtick\relax
        \else
          \dtl@constructminorticklist
             \dtl@prevtick\dtl@nexttick\dtl@unit@y\dtl@yminorticlist
        \fi
        \let\dtl@prevtick=\dtl@nexttick
      }%
    \fi
```

Determine the width of the y tick labels.

```
    \ifx\dtl@ylabel\relax
    \else
      \ifx\dtl@yticlabels\relax
        \@for\dtl@thislabel:=\dtl@yticlist\do{%
          \FPround{\dtl@thislabel}{\dtl@thislabel}
                  {\c@DTLplotroundYvar}%
          \settowidth{\dtl@tmplength}{\dtl@thislabel}%
          \ifdim\dtl@tmplength>\dtl@yticlabelwidth
            \setlength{\dtl@yticlabelwidth}{\dtl@tmplength}%
          \fi
        }%
      \else
        \@for\dtl@thislabel:=\dtl@yticlabels\do{%
          \settowidth{\dtl@tmplength}{\dtl@thislabel}%
          \ifdim\dtl@tmplength>\dtl@yticlabelwidth
            \setlength{\dtl@yticlabelwidth}{\dtl@tmplength}%
          \fi
```

```
      }%
    \fi
  \fi
```

Start the picture.

```
  \begin{tikzpicture}
```

Set the x and y unit vectors.

```
  \pgfsetxvec{\pgfpoint{\dtl@unit@x sp}{0pt}}%
  \pgfsetyvec{\pgfpoint{0pt}{\dtl@unit@y sp}}%
```

Add any extra information the user requires

```
\DTLplotatbegintikz
```

Determine whether to put a box around the plot

```
  \ifDTLbox
    \draw (\DTLminX,\DTLminY) -- (\DTLmaxX,\DTLminY) --
          (\DTLmaxX,\DTLmaxY) -- (\DTLminX,\DTLmaxY) --
          cycle;
  \else
```

Plot x axis if required.

```
    \ifDTLxaxis
      \expandafter\draw\expandafter[\DTLXAxisStyle]
        (\DTLminX,\DTLminY) -- (\DTLmaxX,\DTLminY);
    \fi
```

Plot y axis if required.

```
    \ifDTLyaxis
      \expandafter\draw\expandafter[\DTLYAxisStyle]
        (\DTLminX,\DTLminY) -- (\DTLminX,\DTLmaxY);
    \fi
  \fi
```

Plot grid if required

```
  \ifDTLgrid
    \ifDTLxminortics
      \@for\dtl@thistick:=\dtl@xminorticlist\do{%
        \expandafter\draw\expandafter[\DTLminorgridstyle]
         (\dtl@thistick,\DTLminY) -- (\dtl@thistick,\DTLmaxY);
      }%
    \fi
    \ifDTLyminortics
      \@for\dtl@thistick:=\dtl@yminorticlist\do{%
        \expandafter\draw\expandafter[\DTLminorgridstyle]
         (\DTLminX,\dtl@thistick) -- (\DTLmaxX,\dtl@thistick);
      }%
    \fi
    \@for\dtl@thistick:=\dtl@xticlist\do{%
      \expandafter\draw\expandafter[\DTLmajorgridstyle]
         (\dtl@thistick,\DTLminY) -- (\dtl@thistick,\DTLmaxY);
    }%
    \@for\dtl@thistick:=\dtl@yticlist\do{%
      \expandafter\draw\expandafter[\DTLmajorgridstyle]
         (\DTLminX,\dtl@thistick) -- (\DTLmaxX,\dtl@thistick);
    }%
  \fi
```

Plot x tics if required.

```
\ifDTLxtics
  \addtolength\dtl@xticlabelheight{\DTLticklabeloffset}%
  \@for\dtl@thistick:=\dtl@xticlist\do{%
    \pgfpathmoveto{\pgfpointxy{\dtl@thistick}{\DTLminY}}
    \ifDTLxticsin
      \pgfpathlineto{
          \pgfpointadd{\pgfpointxy{\dtl@thistick}{\DTLminY}}
                      {\pgfpoint{0pt}{\DTLticklength}}}
    \else
      \pgfpathlineto{
        \pgfpointadd{\pgfpointxy{\dtl@thistick}{\DTLminY}}
                    {\pgfpoint{0pt}{-\DTLticklength}}}
    \fi
    \ifDTLbox
      \pgfpathmoveto{\pgfpointxy{\dtl@thistick}{\DTLmaxY}}
      \ifDTLxticsin
        \pgfpathlineto{
            \pgfpointadd{\pgfpointxy{\dtl@thistick}{\DTLmaxY}}
                        {\pgfpoint{0pt}{-\DTLticklength}}}
      \else
        \pgfpathlineto{
          \pgfpointadd{\pgfpointxy{\dtl@thistick}{\DTLmaxY}}
                      {\pgfpoint{0pt}{\DTLticklength}}}
      \fi
    \fi
    \pgfusepath{stroke}%
```

Plot the tick labels

```
    \ifx\dtl@xticlabels\relax
       \FPround{\dtl@thislabel}{\dtl@thistick}
               {\c@DTLplotroundXvar}%
    \else
       \dtl@chopfirst\dtl@xticlabels\dtl@thislabel\dtl@rest
       \let\dtl@xticlabels=\dtl@rest
    \fi
    \pgftext[base,center,at={\pgfpointadd
                {\pgfpointxy{\dtl@thistick}{\DTLminY}}
                {\pgfpoint{0pt}{-\dtl@xticlabelheight}}}]
            {\dtl@thislabel}
  }%
\fi
```

Plot x label if required.

```
\ifx\dtl@xlabel\relax
\else
  \addtolength{\dtl@xticlabelheight}{\baselineskip}%
  \setlength{\dtl@tmplength}{0.5\DTLplotwidth}
  \pgftext[base,center,at={\pgfpointadd
      {\pgfpointxy{\DTLminX}{\DTLminY}}%
      {\pgfpoint{\dtl@tmplength}{-\dtl@xticlabelheight}}}]{%
    \dtl@xlabel}
\fi
```

Plot the x minor ticks if required

```
\ifDTLxminortics
   \@for\dtl@thistick:=\dtl@xminorticlist\do{%
     \pgfpathmoveto{\pgfpointxy{\dtl@thistick}{\DTLminY}}
     \ifDTLxticsin
       \pgfpathlineto{
           \pgfpointadd{\pgfpointxy{\dtl@thistick}{\DTLminY}}
                       {\pgfpoint{0pt}{\DTLminorticklength}}}
     \else
       \pgfpathlineto{
         \pgfpointadd{\pgfpointxy{\dtl@thistick}{\DTLminY}}
                     {\pgfpoint{0pt}{-\DTLminorticklength}}}
     \fi
     \ifDTLbox
       \pgfpathmoveto{\pgfpointxy{\dtl@thistick}{\DTLmaxY}}
       \ifDTLxticsin
         \pgfpathlineto{
             \pgfpointadd{\pgfpointxy{\dtl@thistick}{\DTLmaxY}}
                         {\pgfpoint{0pt}{-\DTLminorticklength}}}
       \else
         \pgfpathlineto{
           \pgfpointadd{\pgfpointxy{\dtl@thistick}{\DTLmaxY}}
                       {\pgfpoint{0pt}{\DTLminorticklength}}}
       \fi
     \fi
   }%
\fi
```

Plot y tics if required.

```
\ifDTLytics
  \@for\dtl@thistick:=\dtl@yticlist\do{%
    \pgfpathmoveto{\pgfpointxy{\DTLminX}{\dtl@thistick}}
    \ifDTLyticsin
      \pgfpathlineto{
          \pgfpointadd{\pgfpointxy{\DTLminX}{\dtl@thistick}}
                      {\pgfpoint{\DTLticklength}{0pt}}}
    \else
      \pgfpathlineto{
        \pgfpointadd{\pgfpointxy{\DTLminX}{\dtl@thistick}}
                    {\pgfpoint{-\DTLticklength}{0pt}}}
    \fi
    \ifDTLbox
      \pgfpathmoveto{\pgfpointxy{\DTLmaxX}{\dtl@thistick}}
      \ifDTLyticsin
        \pgfpathlineto{
            \pgfpointadd{\pgfpointxy{\DTLmaxX}{\dtl@thistick}}
                        {\pgfpoint{-\DTLticklength}{0pt}}}
      \else
        \pgfpathlineto{
            \pgfpointadd{\pgfpointxy{\DTLmaxX}{\dtl@thistick}}
                        {\pgfpoint{\DTLticklength}{0pt}}}
      \fi
    \fi
    \pgfusepath{stroke}
```

Plot the y tick labels if required

```
      \ifx\dtl@yticlabels\relax
         \FPround{\dtl@thislabel}{\dtl@thistick}
                 {\c@DTLplotroundYvar}%
      \else
         \dtl@chopfirst\dtl@yticlabels\dtl@thislabel\dtl@rest
         \let\dtl@yticlabels=\dtl@rest
      \fi
      \pgftext[right,at={\pgfpointadd
                  {\pgfpointxy{\DTLminX}{\dtl@thistick}}
                  {\pgfpoint{-\DTLticklabeloffset}{0pt}}}]
        {\dtl@thislabel}
    }%
  \fi
```

Plot y label if required.

```
\ifx\dtl@ylabel\relax
\else
  \addtolength{\dtl@yticlabelwidth}{\baselineskip}%
  \setlength{\dtl@tmplength}{0.5\DTLplotheight}
  \pgftext[bottom,center,at={\pgfpointadd
      {\pgfpointxy{\DTLminX}{\DTLminY}}%
      {\pgfpoint{-\dtl@yticlabelwidth}{\dtl@tmplength}}},
      rotate=90]{%
     \dtl@ylabel}
\fi
```

Plot the y minor ticks if required

```
\ifDTLyminortics
   \@for\dtl@thistick:=\dtl@yminorticlist\do{%
     \pgfpathmoveto{\pgfpointxy{\DTLminX}{\dtl@thistick}}
     \ifDTLyticsin
       \pgfpathlineto{
           \pgfpointadd{\pgfpointxy{\DTLminX}{\dtl@thistick}}
                       {\pgfpoint{\DTLminorticklength}{0pt}}}
     \else
       \pgfpathlineto{
         \pgfpointadd{\pgfpointxy{\DTLminX}{\dtl@thistick}}
                     {\pgfpoint{-\DTLminorticklength}{0pt}}}
     \fi
     \ifDTLbox
       \pgfpathmoveto{\pgfpointxy{\DTLmaxX}{\dtl@thistick}}
       \ifDTLyticsin
         \pgfpathlineto{
             \pgfpointadd{\pgfpointxy{\DTLmaxX}{\dtl@thistick}}
                         {\pgfpoint{-\DTLminorticklength}{0pt}}}
       \else
         \pgfpathlineto{
             \pgfpointadd{\pgfpointxy{\DTLmaxX}{\dtl@thistick}}
                         {\pgfpoint{\DTLminorticklength}{0pt}}}
       \fi
     \fi
     \pgfusepath{stroke}
   }%
\fi
```

Iterate through each database

```
\@for\dtl@thisdb:=#2\do{%
```

Get the current plot mark colour.

```
\ifx\dtl@plotmarkcolorlist\@empty
  \let\dtl@plotmarkcolorlist=\DTLplotmarkcolors
\fi
\dtl@chopfirst\dtl@plotmarkcolorlist\dtl@thisplotmarkcolor
   \dtl@remainder
\let\dtl@plotmarkcolorlist=\dtl@remainder
```

Get the current plot mark, and store in `\dtl@mark`

```
\ifDTLshowmarkers
  \ifx\dtl@plotmarklist\@empty
     \let\dtl@plotmarklist=\DTLplotmarks
  \fi
  \dtl@chopfirst\dtl@plotmarklist\dtl@thisplotmark
     \dtl@remainder
  \let\dtl@plotmarklist=\dtl@remainder
  \ifx\dtl@thisplotmark\relax
    \let\dtl@mark=\relax
  \else
    \expandafter\toks@\expandafter{\dtl@thisplotmark}%
    \ifx\dtl@thisplotmarkcolor\@empty
      \edef\dtl@mark{\the\toks@}%
    \else
      \edef\dtl@mark{%
          \noexpand\color{\dtl@thisplotmarkcolor}%
        \the\toks@}%
    \fi
  \fi
\else
  \let\dtl@mark=\relax
\fi
```

Get the current plot line colour.

```
\ifx\dtl@plotlinecolorlist\@empty
  \let\dtl@plotlinecolorlist=\DTLplotlinecolors
\fi
\dtl@chopfirst\dtl@plotlinecolorlist\dtl@thisplotlinecolor
   \dtl@remainder
\let\dtl@plotlinecolorlist=\dtl@remainder
```

Get the current line style, and store in `\dtl@linestyle`

```
\ifDTLshowlines
  \ifx\dtl@plotlinelist\@empty
     \let\dtl@plotlinelist=\DTLplotlines
  \fi
  \dtl@chopfirst\dtl@plotlinelist\dtl@thisplotline
     \dtl@remainder
  \let\dtl@plotlinelist=\dtl@remainder
  \expandafter\ifx\dtl@thisplotline\relax
    \let\dtl@linestyle=\relax
  \else
    \expandafter\toks@\expandafter{\dtl@thisplotline}%
```

```
      \ifx\dtl@thisplotlinecolor\@empty
        \edef\dtl@linestyle{\the\toks@}%
      \else
        \edef\dtl@linestyle{%
            \noexpand\color{\dtl@thisplotlinecolor}%
          \the\toks@}%
      \fi
    \fi
  \else
    \let\dtl@linestyle=\relax
  \fi
```

Append this plot setting to the legend.

```
  \ifnum\dtl@legendsetting>0\relax
    \dtl@chopfirst\dtl@legendlabels\dtl@thislabel\dtl@rest
    \let\dtl@legendlabels=\dtl@rest
    \expandafter\toks@\expandafter{\dtl@mark}%
    \expandafter\@dtl@toks\expandafter{\dtl@linestyle}%
    \edef\dtl@addtolegend{\noexpand\DTLaddtoplotlegend
      {\the\toks@}{\the\@dtl@toks}{\dtl@thislabel}}%
    \dtl@addtolegend
  \fi
```

Store stream in `\dtl@stream`

```
  \def\dtl@stream{\pgfplotstreamstart}%
```

Only plot points that lie inside bounds.

```
  \@sDTLforeach[#1]{\dtl@thisdb}{\dtl@x=\dtl@xkey,%
      \dtl@y=\dtl@ykey}{%
    \DTLconverttodecimal{\dtl@x}{\dtl@decx}%
    \DTLconverttodecimal{\dtl@y}{\dtl@decy}%
    \ifthenelse{%
      \DTLisclosedbetween{\dtl@x}{\DTLminX}{\DTLmaxX}%
      \and
      \DTLisclosedbetween{\dtl@y}{\DTLminY}{\DTLmaxY}%
      }{%
      \expandafter\toks@\expandafter{\dtl@stream}%
      \edef\dtl@stream{\the\toks@
        \noexpand\pgfplotstreampoint
         {\noexpand\pgfpointxy{\dtl@decx}{\dtl@decy}}}%
    }{}%
  }%
  \expandafter\toks@\expandafter{\dtl@stream}%
  \edef\dtl@stream{\the\toks@\noexpand\pgfplotstreamend}%
```

End plot stream and draw path.

```
  \ifx\dtl@linestyle\relax
  \else
    \begin{scope}
    \dtl@linestyle
    \pgfplothandlerlineto
    \dtl@stream
    \pgfusepath{stroke}
    \end{scope}
  \fi
  \ifx\dtl@mark\relax
```

```
    \else
      \begin{scope}
      \pgfplothandlermark{\dtl@mark}%
      \dtl@stream
      \pgfusepath{stroke}
      \end{scope}
    \fi
  }%
```

Plot legend if required.

```
  \ifcase\dtl@legendsetting
  % none
  \or % north
    \pgftext[top,center,at={\pgfpointadd
             {\pgfpointxy{\DTLminX}{\DTLmaxY}}
             {\pgfpoint{0.5\DTLplotwidth}{-\DTLlegendyoffset}}}]
            {\DTLformatlegend
              {\begin{tabular}{cl}\dtl@legend\end{tabular}}}
  \or % north east
    \pgftext[top,right,at={\pgfpointadd
             {\pgfpointxy{\DTLmaxX}{\DTLmaxY}}
             {\pgfpoint{-\DTLlegendxoffset}{-\DTLlegendyoffset}}}]
            {\DTLformatlegend
              {\begin{tabular}{cl}\dtl@legend\end{tabular}}}
  \or % east
    \pgftext[center,right,at={\pgfpointadd
             {\pgfpointxy{\DTLmaxX}{\DTLminY}}
             {\pgfpoint{-\DTLlegendxoffset}{0.5\DTLplotheight}}}]
            {\DTLformatlegend
              {\begin{tabular}{cl}\dtl@legend\end{tabular}}}
  \or % south east
    \pgftext[bottom,right,at={\pgfpointadd
             {\pgfpointxy{\DTLmaxX}{\DTLminY}}
             {\pgfpoint{-\DTLlegendxoffset}{\DTLlegendyoffset}}}]
            {\DTLformatlegend
               {\begin{tabular}{cl}\dtl@legend\end{tabular}}}
  \or % south
    \pgftext[center,bottom,at={\pgfpointadd
             {\pgfpointxy{\DTLminX}{\DTLminY}}
             {\pgfpoint{0.5\DTLplotwidth}{\DTLlegendyoffset}}}]
            {\DTLformatlegend
               {\begin{tabular}{cl}\dtl@legend\end{tabular}}}
  \or % south west
    \pgftext[bottom,left,at={\pgfpointadd
             {\pgfpointxy{\DTLminX}{\DTLminY}}
             {\pgfpoint{\DTLlegendxoffset}{\DTLlegendyoffset}}}]
            {\DTLformatlegend
               {\begin{tabular}{cl}\dtl@legend\end{tabular}}}
  \or % west
    \pgftext[center,left,at={\pgfpointadd
             {\pgfpointxy{\DTLminX}{\DTLminY}}
             {\pgfpoint{\DTLlegendxoffset}{0.5\DTLplotheight}}}]
            {\DTLformatlegend
               {\begin{tabular}{cl}\dtl@legend\end{tabular}}}
  \or % north west
```

```
        \pgftext[top,left,at={\pgfpointadd
                 {\pgfpointxy{\DTLminX}{\DTLmaxY}}
                 {\pgfpoint{\DTLlegendxoffset}{-\DTLlegendyoffset}}}]
                {\DTLformatlegend
                   {\begin{tabular}{cl}\dtl@legend\end{tabular}}}
      \fi
      \DTLplotatendtikz
      \end{tikzpicture}
    \fi
  \fi
\fi
}}
```

`\dtl@getbounds` Extract bounds:

```
\def\dtl@getbounds#1,#2,#3,#4\@nil{%
\def\DTLminX{#1}%
\def\DTLminY{#2}%
\def\DTLmaxX{#3}%
\def\DTLmaxY{#4}%
\FPifgt{\DTLminX}{\DTLmaxX}
 \PackageError{dataplot}{Min X > Max X in bounds #1,#2,#3,#4}{%
 The bounds must be specified as minX,minY,maxX,maxY}%
\fi
\FPifgt{\DTLminY}{\DTLmaxY}
 \PackageError{dataplot}{Min Y > Max Y in bounds #1,#2,#3,#4}{%
 The bounds must be specified as minX,minY,maxX,maxY}%
\fi
}
```

`\dtl@constructticklist` `\dtl@constructticklist{`⟨*min*⟩`}{`⟨*max*⟩`}{`⟨*scale factor*⟩`}{`⟨*list*⟩`}`

Constructs a list of tick points between ⟨*min*⟩ and ⟨*max*⟩ and store in ⟨*list*⟩ (a control sequence.)

```
\newcommand*{\dtl@constructticklist}[4]{%
\DTLifFPopenbetween{0}{#1}{#2}{%
  \FPsub{\@dtl@width}{0}{#1}%
  \FPmul{\@dtl@width}{\@dtl@width}{#3}%
  \FPdiv{\@dtl@neggap}{\@dtl@width}{10}%
  \setlength\dtl@tmplength{\@dtl@neggap sp}%
  \ifdim\dtl@tmplength<\DTLmintickgap
    \FPdiv{\@dtl@neggap}{\@dtl@width}{4}%
    \setlength\dtl@tmplength{\@dtl@neggap sp}%
    \ifdim\dtl@tmplength<\DTLmintickgap
      \FPdiv{\@dtl@neggap}{\@dtl@width}{2}%
      \setlength\dtl@tmplength{\@dtl@neggap sp}%
      \ifdim\dtl@tmplength<\DTLmintickgap
        \let\@dtl@neggap=\@dtl@width
      \fi
    \fi
  \fi
  \FPmul{\@dtl@width}{#2}{#3}%
  \FPdiv{\@dtl@posgap}{\@dtl@width}{10}%
```

```
    \setlength\dtl@tmplength{\@dtl@posgap sp}%
    \ifdim\dtl@tmplength<\DTLmintickgap
      \FPdiv{\@dtl@posgap}{\@dtl@width}{4}%
      \setlength\dtl@tmplength{\@dtl@posgap sp}%
      \ifdim\dtl@tmplength<\DTLmintickgap
        \FPdiv{\@dtl@posgap}{\@dtl@width}{2}%
        \setlength\dtl@tmplength{\@dtl@posgap sp}%
        \ifdim\dtl@tmplength<\DTLmintickgap
          \let\@dtl@posgap=\@dtl@width
        \fi
      \fi
    \fi
    \FPmax{\@dtl@gap}{\@dtl@neggap}{\@dtl@posgap}%
    \FPdiv{\@dtl@gap}{\@dtl@gap}{#3}%
    \dtl@constructticklistwithgapz{#1}{#2}{#4}{\@dtl@gap}%
  }{%
    \FPsub{\@dtl@width}{#2}{#1}%
    \FPmul{\@dtl@width}{\@dtl@width}{#3}%
    \FPdiv{\@dtl@gap}{\@dtl@width}{10}%
    \setlength\dtl@tmplength{\@dtl@gap sp}%
    \ifdim\dtl@tmplength<\DTLmintickgap
      \FPdiv{\@dtl@gap}{\@dtl@width}{4}%
      \setlength\dtl@tmplength{\@dtl@gap sp}%
      \ifdim\dtl@tmplength<\DTLmintickgap
        \FPdiv{\@dtl@gap}{\@dtl@width}{2}%
        \setlength\dtl@tmplength{\@dtl@gap sp}%
        \ifdim\dtl@tmplength<\DTLmintickgap
          \let\@dtl@gap=\@dtl@width
        \fi
      \fi
    \fi
    \FPdiv{\@dtl@gap}{\@dtl@gap}{#3}%
    \dtl@constructticklistwithgap{#1}{#2}{#4}{\@dtl@gap}%
  }%
  }
```

`\dtl@constructticklistwithgap`

`\dtl@constructticklistwithgap{`⟨*min*⟩`}{`⟨*max*⟩`}{`⟨*list*⟩`}{`⟨*gap*⟩`}`

Constructs a list of tick points between ⟨*min*⟩ and ⟨*max*⟩ and store in ⟨*list*⟩ (a control sequence) using the gap given by ⟨*gap*⟩ where the gap is given in user co-ordinates.

```
\newcommand*{\dtl@constructticklistwithgap}[4]{%
\edef\@dtl@thistick{#1}%
\edef#3{#1}%
\FPadd{\@dtl@thistick}{\@dtl@thistick}{#4}%
\whiledo{\DTLisFPopenbetween{\@dtl@thistick}{#1}{#2}}{%
  \expandafter\toks@\expandafter{\@dtl@thistick}%
  \edef#3{#3,\the\toks@}%
  \FPadd{\@dtl@thistick}{\@dtl@thistick}{#4}%
}%
\expandafter\toks@\expandafter{#2}%
\edef#3{#3,\the\toks@}%
```

```
}
```

\dtl@constructticklistwithgapz

\dtl@constructticklistwithgapz{⟨*min*⟩}{⟨*max*⟩}{⟨*list*⟩}{⟨*gap*⟩}

Constructs a list of tick points between ⟨*min*⟩ and ⟨*max*⟩ and store in ⟨*list*⟩ (a control sequence) using the gap given by ⟨*gap*⟩ where the tick list straddles zero.

```
\newcommand*{\dtl@constructticklistwithgapz}[4]{%
\edef\@dtl@thistick{0}%
\edef#3{0}%
\FPadd{\@dtl@thistick}{\@dtl@thistick}{#4}%
\whiledo{\DTLisFPopenbetween{\@dtl@thistick}{0}{#2}}{%
  \expandafter\toks@\expandafter{\@dtl@thistick}%
  \edef#3{#3,\the\toks@}%
  \FPadd{\@dtl@thistick}{\@dtl@thistick}{#4}%
}%
\expandafter\toks@\expandafter{#2}%
\edef#3{#3,\the\toks@}%
\FPifeq{#1}{0}%
\else
\edef\@dtl@thistick{0}%
  \FPsub{\@dtl@thistick}{\@dtl@thistick}{#4}%
  \whiledo{\DTLisFPopenbetween{\@dtl@thistick}{#1}{0}}{%
    \expandafter\toks@\expandafter{\@dtl@thistick}%
    \edef#3{\the\toks@,#3}%
    \FPsub{\@dtl@thistick}{\@dtl@thistick}{#4}%
  }%
  \expandafter\toks@\expandafter{#1}%
  \edef#3{\the\toks@,#3}%
\fi
}
```

\dtl@constructminorticklist

\dtl@constructminorticklist{⟨*min*⟩}{⟨*max*⟩}{⟨*scale factor*⟩}{⟨*list*⟩}

Constructs a list of minor tick points between ⟨*min*⟩ and ⟨*max*⟩ and append to ⟨*list*⟩ (a control sequence.)

```
\newcommand*{\dtl@constructminorticklist}[4]{%
  \FPsub{\@dtl@width}{#2}{#1}%
  \FPmul{\@dtl@width}{\@dtl@width}{#3}%
  \FPdiv{\@dtl@gap}{\@dtl@width}{10}%
  \setlength\dtl@tmplength{\@dtl@gap sp}%
  \ifdim\dtl@tmplength<\DTLminminortickgap
    \FPdiv{\@dtl@gap}{\@dtl@width}{4}%
    \setlength\dtl@tmplength{\@dtl@gap sp}%
    \ifdim\dtl@tmplength<\DTLminminortickgap
      \FPdiv{\@dtl@gap}{\@dtl@width}{2}%
      \setlength\dtl@tmplength{\@dtl@gap sp}%
      \ifdim\dtl@tmplength<\DTLminminortickgap
        \let\@dtl@gap=\@dtl@width
      \fi
    \fi
```

```
  \fi
  \FPdiv{\@dtl@gap}{\@dtl@gap}{#3}%
  \dtl@constructticklistwithgapex{#1}{#2}{\dtl@tmp}{\@dtl@gap}%
  \ifx#4\@empty
    \let#4=\dtl@tmp
  \else
    \expandafter\toks@\expandafter{#4}%
    \edef#4{#4,\dtl@tmp}%
  \fi
}
```

dtl@constructticklistwithgapex

\dtl@constructticklistwithgapex{⟨*min*⟩}{⟨*max*⟩}{⟨*list*⟩}{⟨*gap*⟩}

Constructs a list of tick points between ⟨*min*⟩ and ⟨*max*⟩ and store in ⟨*list*⟩ (a control sequence) using the gap given by ⟨*gap*⟩ where the gap is given in user co-ordinates. The end points are excluded from the list.

```
\newcommand*{\dtl@constructticklistwithgapex}[4]{%
\edef\@dtl@thistick{#1}%
\let#3=\@empty
\FPadd{\@dtl@thistick}{\@dtl@thistick}{#4}%
\whiledo{\DTLisFPopenbetween{\@dtl@thistick}{#1}{#2}}{%
  \expandafter\toks@\expandafter{\@dtl@thistick}%
  \ifx#3\@empty
    \edef#3{\the\toks@}%
  \else
    \edef#3{#3,\the\toks@}%
  \fi
  \FPadd{\@dtl@thistick}{\@dtl@thistick}{#4}%
}%
}
```

\DTLaddtoplotlegend

\DTLaddtoplotlegend{⟨*marker*⟩}{⟨*line style*⟩}{⟨*label*⟩}

Adds entry to legend.

```
\newcommand*{\DTLaddtoplotlegend}[3]{%
\def\dtl@legendline{}%
\ifx\relax#2\relax
\else
  \toks@{#2%
  \pgfpathmoveto{\pgfpoint{-10pt}{0pt}}%
  \pgfpathlineto{\pgfpoint{10pt}{0pt}}%
  \pgfusepath{stroke}}%
  \edef\dtl@legendline{\the\toks@}%
\fi
\ifx\relax#1\relax
\else
  \toks@{#1}%
  \expandafter\@dtl@toks\expandafter{\dtl@legendline}%
  \edef\dtl@legendline{\the\@dtl@toks\the\toks@}%
\fi
```

```
\expandafter\toks@\expandafter{\dtl@legendline}%
\ifx\dtl@legend\@empty
  \edef\dtl@legend{\noexpand\tikz\the\toks@; \noexpand& #3}%
\else
  \expandafter\@dtl@toks\expandafter{\dtl@legend}%
  \edef\dtl@legend{\the\@dtl@toks\noexpand\\%
    \noexpand\tikz\the\toks@; \noexpand& #3}%
\fi
}
```

13 databar.sty

Declare package:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{databar}[2009/02/27 v2.0 (NLCT)]
```

Require xkeyval package

```
\RequirePackage{xkeyval}
```

Require dataplot package

```
\RequirePackage{dataplot}
```

`\ifDTLcolorbarchart` The conditional `\ifDTLcolorbarchart` is used to determine whether to use colour or grey scale.

```
\newif\ifDTLcolorbarchart
\DTLcolorbarchart true
```

Package options to change the conditional:

```
\DeclareOption{color}{\DTLcolorbarchart true}
\DeclareOption{gray}{\DTLcolorbarchart false}
```

`\DTLbarXlabelalign` specifies the alignment for the x axis labels.

```
\newcommand*{\DTLbarXlabelalign}{left,rotate=-90}
```

`\DTLbarYticklabelalign` specifies the alignment for the x axis labels.

```
\newcommand*{\DTLbarYticklabelalign}{right}
```

`\ifDTLverticalbars` Define boolean keys to govern bar chart orientation.

```
\define@boolkey{databar}[DTL]{verticalbars}[true]{%
\ifDTLverticalbars
 \def\DTLbarXlabelalign{left,rotate=-90}%
 \def\DTLbarYticklabelalign{right}
\else
 \def\DTLbarXlabelalign{right}%
 \def\DTLbarYticklabelalign{center}
\fi}
```

Set defaults:

```
\DTLverticalbarstrue
```

Package options to change `\ifDTLverticalbars`

```
\DeclareOption{vertical}{\DTLverticalbarstrue
 \def\DTLbarXlabelalign{left,rotate=-90}%
 \def\DTLbarYticklabelalign{right}
```

```
}
\DeclareOption{horizontal}{\DTLverticalbarsfalse
 \def\DTLbarXlabelalign{right}%
 \def\DTLbarYticklabelalign{center}
}
```

Process options:

```
\ProcessOptions
```

Required packages:

```
\RequirePackage{datatool}
\RequirePackage{tikz}
```

Define some variables that govern the appearance of the bar chart.

`\DTLbarchartlength` The total height of the bar chart is given by `\DTLbarchartheight`

```
\newlength\DTLbarchartlength
\DTLbarchartlength=3in
```

`\DTLbarwidth` The width of each bar is given by `\DTLbarwidth`.

```
\newlength\DTLbarwidth
\DTLbarwidth=1cm
```

`\DTLbarlabeloffset` The offset from the x axis to the bar label if given by `\DTLbarlabeloffset`.

```
\newlength\DTLbarlabeloffset
\setlength\DTLbarlabeloffset{10pt}
```

`\DTLBarXAxisStyle` The style of the x axis is given by `\DTLBarXAxisStyle`

```
\newcommand*{\DTLBarXAxisStyle}{-}
```

`\DTLBarYAxisStyle` The style of the y axis is given by `\DTLBarYAxisStyle`.

```
\newcommand*{\DTLBarYAxisStyle}{-}
```

`DTLbarroundvar` is a counter governing the number of digits to round to when using `\FPround`.

```
\newcounter{DTLbarroundvar}
\setcounter{DTLbarroundvar}{1}
```

`\DTLbardisplayYticklabel` `\DTLbardisplayYticklabel` governs how the y tick labels appear.

```
\newcommand*{\DTLbardisplayYticklabel}[1]{#1}
```

`\DTLdisplaylowerbarlabel` `\DTLdisplaylowerbarlabel` governs how the lower bar labels appear.

```
\newcommand*{\DTLdisplaylowerbarlabel}[1]{#1}
```

`\DTLdisplaylowermultibarlabel` `\DTLdisplaylowermultibarlabel` governs how the lower multi bar labels appear.

```
\newcommand*{\DTLdisplaylowermultibarlabel}[1]{#1}
```

`\DTLdisplayupperbarlabel` `\DTLdisplayupperbarlabel` governs how the upper bar labels appear.

```
\newcommand*{\DTLdisplayupperbarlabel}[1]{#1}
```

`\DTLdisplayuppermultibarlabel` `\DTLdisplayuppermultibarlabel` governs how the upper multi bar labels appear.

```
\newcommand*{\DTLdisplayuppermultibarlabel}[1]{#1}
```

`\DTLbaratbegintikz` `\DTLbaratbegintikz` specifies any commands to apply at the start of the tikzpicture environment. By default it does nothing.

```
\newcommand*{\DTLbaratbegintikz}{}
```

`\DTLbaratendtikz` `\DTLbaratendtikz` specifies any commands to apply at the end of the tikzpicture environment. By default it does nothing.

```
\newcommand*{\DTLbaratendtikz}{}
```

`\ifDTLbarxaxis` The conditional `\ifDTLbarxaxis` is used to determine whether or not to display the x axis

```
\newif\ifDTLbarxaxis
```

`\ifDTLbaryaxis` The conditional `\ifDTLbaryaxis` is used to determine whether or not to display the y axis.

```
\newif\ifDTLbaryaxis
```

`\ifDTLbarytics` The conditional `\ifDTLbarytics` to determine whether or not to display the y tick marks.

```
\newif\ifDTLbarytics
```

`\@dtl@barcount` The count register `\@dtl@barcount` is used to store the current bar index.

```
\newcount\@dtl@barcount
```

`\DTLsetbarcolor`

`\DTLsetbarcolor{⟨n⟩}{⟨color⟩}`

Assigns colour name ⟨*color*⟩ to the ⟨*n*⟩th bar.

```
\newcommand*{\DTLsetbarcolor}[2]{%
\expandafter\def\csname dtlbar@segcol\romannumeral#1\endcsname{#2}%
}
```

`\DTLgetbarcolor`

`\DTLgetbarcolor{⟨n⟩}`

Gets the colour specification for the ⟨*n*⟩th bar.

```
\newcommand*{\DTLgetbarcolor}[1]{%
\csname dtlbar@segcol\romannumeral#1\endcsname}
```

`\DTLdobarcolor`

`\DTLdobarcolor{⟨n⟩}`

Sets the colour to that for the ⟨*n*⟩th bar.

```
\newcommand*{\DTLdobarcolor}[1]{%
\expandafter\color\expandafter
{\csname dtlbar@segcol\romannumeral#1\endcsname}}
```

`\DTLdocurrentbarcolor` `\DTLdocurrentbarcolor` sets the colour to that of the current bar.

```
\newcommand*{\DTLdocurrentbarcolor}{%
\ifnum\dtlforeachlevel=0\relax
  \PackageError{databar}{Can't use
```

```
    \string\DTLdocurrentbarcolor\space outside
    \string\DTLbarchart}{}%
  \else
    \expandafter\DTLdobarcolor\expandafter{%
    \csname c@DTLrow\romannumeral\dtlforeachlevel\endcsname}%
  \fi}
```

`\DTLbaroutlinecolor` `\DTLbaroutlinecolor` specifies what colour to draw the outline.

```
\newcommand*{\DTLbaroutlinecolor}{black}
```

`\DTLbaroutlinewidth` `\DTLbaroutlinewidth` specifies the line width of the outline: Outline is only drawn if the linewidth is greater than 0pt.

```
\newlength\DTLbaroutlinewidth
\DTLbaroutlinewidth=0pt
```

Set the default colours. If there are more than eight bars, more colours will need to be defined.

```
\ifDTLcolorbarchart
 \DTLsetbarcolor{1}{red}
 \DTLsetbarcolor{2}{green}
 \DTLsetbarcolor{3}{blue}
 \DTLsetbarcolor{4}{yellow}
 \DTLsetbarcolor{5}{magenta}
 \DTLsetbarcolor{6}{cyan}
 \DTLsetbarcolor{7}{orange}
 \DTLsetbarcolor{8}{white}
\else
 \DTLsetbarcolor{1}{black!15}
 \DTLsetbarcolor{2}{black!25}
 \DTLsetbarcolor{3}{black!35}
 \DTLsetbarcolor{4}{black!45}
 \DTLsetbarcolor{5}{black!55}
 \DTLsetbarcolor{6}{black!65}
 \DTLsetbarcolor{7}{black!75}
 \DTLsetbarcolor{8}{black!85}
\fi
```

`\DTLeverybarhook` Code to apply at every bar. The start point of the bar can be accessed via `\DTLstartpt`, the mid point of the bar can be accessed via `\DTLmidpt` and the end point of the bar can be accessed via `\DTLendpt`

```
\newcommand*{\DTLeverybarhook}{}
```

Define keys for `\DTLbarchart` optional argument. Set the maximum value of the y axis.

```
\define@key{databar}{max}{\def\DTLbarmax{#1}}
```

Set the total length of the bar chart

```
\define@key{databar}{length}{\DTLbarchartlength=#1\relax
}
```

Set the maximum depth (negative extent)

```
\define@key{databar}{maxdepth}{%
\ifnum#1>0\relax
 \PackageError{databar}{depth must be zero or negative}{}%
```

```
\else
 \def\DTLnegextent{#1}%
\fi}
```

Determine which axes should be shown

```
\define@choicekey{databar}{axes}[\var\nr]{both,x,y,none}{%
\ifcase\nr\relax
 % both
  \DTLbarxaxistrue
  \DTLbaryaxistrue
  \DTLbaryticstrue
\or
 % x only
  \DTLbarxaxistrue
  \DTLbaryaxisfalse
  \DTLbaryticsfalse
\or
 % y only
  \DTLbarxaxisfalse
  \DTLbaryaxistrue
  \DTLbaryticstrue
\or
 % neither
  \DTLbarxaxisfalse
  \DTLbaryaxisfalse
  \DTLbaryticsfalse
\fi
}
```

Variable used to create the bar chart. (Must be a control sequence.)

```
\define@key{databar}{variable}{%
\def\DTLbarvariable{#1}}
```

Variables used to create the multi bar chart. (Must be a comma separated list of control sequences.)

```
\define@key{databar}{variables}{%
\def\dtlbar@variables{#1}}
```

Bar width

```
\define@key{databar}{barwidth}{\setlength{\DTLbarwidth}{#1}}
```

Lower bar labels

```
\define@key{databar}{barlabel}{%
\def\dtl@barlabel{#1}}
\def\dtl@barlabel{}
```

Lower bar labels for multi-bar charts

```
\define@key{databar}{multibarlabels}{%
\def\dtl@multibarlabels{#1}}
\def\dtl@multibarlabels{}
```

Gap between groups in multi-bar charts (This should be in x units where 1 x unit is the width of a bar.)

```
\define@key{databar}{groupgap}{\def\dtlbar@groupgap{#1}}
\def\dtlbar@groupgap{1}
```

Upper bar labels

```
\define@key{databar}{upperbarlabel}{%
\def\dtl@upperbarlabel{#1}}
\def\dtl@upperbarlabel{}
```

Upper bar labels for multi-bar charts

```
\define@key{databar}{uppermultibarlabels}{%
\def\dtl@uppermultibarlabels{#1}}
\def\dtl@uppermultibarlabels{}
```

Define list of points for y tics. (Must be a comma separated list of decimal numbers.)

```
\define@key{databar}{yticpoints}{%
\def\dtlbar@yticlist{#1}\DTLbaryticstrue\DTLbaryaxistrue}
\let\dtlbar@yticlist=\relax
```

Set the y tick gap:

```
\define@key{databar}{yticgap}{%
\def\dtlbar@yticgap{#1}\DTLbaryticstrue\DTLbaryaxistrue}
\let\dtlbar@yticgap=\relax
```

Define list of labels for y tics.

```
\define@key{databar}{yticlabels}{%
\def\dtlbar@yticlabels{#1}\DTLbaryticstrue\DTLbaryaxistrue}
\let\dtlbar@yticlabels=\relax
```

Define y axis label.

```
\define@key{databar}{ylabel}{%
\def\dtlbar@ylabel{#1}}
\let\dtlbar@ylabel=\relax
```

\DTLbarchart

`\DTLbarchart[`⟨*conditions*⟩`]{`⟨*option list*⟩`}{`⟨*db name*⟩`}{`⟨*assign list*⟩`}`

Make a bar chart from data given in data base ⟨*db name*⟩, where ⟨*assign list*⟩ is a comma-separated list of ⟨*cmd*⟩=⟨*key*⟩ pairs. ⟨*option list*⟩ must include `variable=`⟨*cmd*⟩, where ⟨*cmd*⟩ is included in ⟨*assign list*⟩. The optional argument ⟨*conditions*⟩ is the same as that for `\DTLforeach`.

```
\newcommand*{\DTLbarchart}[4][\boolean{true}]{%
{\let\DTLbarvariable=\relax
\let\DTLbarmax=\relax
\let\DTLnegextent=\relax
\disable@keys{databar}{variables,multibarlabels,%
uppermultibarlabels,groupgap}%
\setkeys{databar}{#2}%
\ifx\DTLbarvariable\relax
  \PackageError{databar}{\string\DTLbarchart\space missing
   variable}{You haven't use the "variable" key}%
\else
```

Compute the maximum bar height, unless `\DTLbarmax` has been set.

```
  \ifx\DTLbarmax\relax
    \@sDTLforeach[#1]{#3}{#4}{%
      \expandafter\DTLconverttodecimal\expandafter
        {\DTLbarvariable}{\dtl@barvar}%
```

```
      \ifx\DTLbarmax\relax
        \let\DTLbarmax=\dtl@barvar
      \else
        \let\dtl@old=\DTLbarmax
        \FPmax{\DTLbarmax}{\dtl@old}{\dtl@barvar}%
      \fi
    }%
    \ifx\dtlbar@yticgap\relax
    \else
      \let\dtl@thistick=\dtlbar@yticgap%
      \whiledo{\DTLisFPopenbetween{\dtl@thistick}{0}{\DTLbarmax}}{%
        \FPadd{\dtl@thistick}{\dtl@thistick}{\dtlbar@yticgap}%
      }%
      \let\DTLbarmax=\dtl@thistick
    \fi
  \fi
```

Compute the bar depth, unless `\DTLnegextent` has been set.

```
  \ifx\DTLnegextent\relax
    \def\DTLnegextent{0}%
    \@sDTLforeach[#1]{#3}{#4}{%
      \expandafter\DTLconverttodecimal\expandafter
        {\DTLbarvariable}{\dtl@barvar}%
      \let\dtl@old=\DTLnegextent
      \DTLmin{\DTLnegextent}{\dtl@old}{\dtl@barvar}%
    }%
    \ifx\dtlbar@yticgap\relax
    \else
      \ifthenelse{\DTLisFPlt{\DTLnegextent}{0}}{%
      \edef\dtl@thistick{0}%
      \whiledo{\DTLisFPclosedbetween{\dtl@thistick}{\DTLnegextent}{0}}{%
        \FPsub{\dtl@thistick}{\dtl@thistick}{\dtlbar@yticgap}%
      }%
      \let\DTLnegextent=\dtl@thistick
      }{}%
    \fi
  \fi
```

Determine scaling factor

```
  \@dtl@tmpcount=\DTLbarchartlength
  \FPsub{\dtl@extent}{\DTLbarmax}{\DTLnegextent}%
  \FPdiv{\dtl@unit}{\number\@dtl@tmpcount}{\dtl@extent}%
```

Construct y tick list if required

```
  \setlength{\dtl@yticlabelwidth}{0pt}%
  \ifDTLbarytics
    \ifx\dtlbar@yticlist\relax
      \ifx\dtlbar@yticgap\relax
        \dtl@constructticklist\DTLnegextent\DTLbarmax
          \dtl@unit\dtlbar@yticlist
      \else
        \dtl@constructticklistwithgapz
          \DTLnegextent\DTLbarmax\dtlbar@yticlist\dtlbar@yticgap
      \fi
    \fi
```

```
    \ifx\dtlbar@ylabel\relax
    \else
      \ifx\dtlbar@yticlabels\relax
        \@for\dtl@thislabel:=\dtlbar@yticlist\do{%
          \FPround{\dtl@thislabel}{\dtl@thislabel}
                  {\c@DTLbarroundvar}%
          \ifDTLverticalbars
            \settowidth{\dtl@tmplength}{%
               \DTLbardisplayYticklabel{\dtl@thislabel}}%
          \else
            \settoheight{\dtl@tmplength}{%
               \DTLbardisplayYticklabel{\dtl@thislabel}}%
            \edef\@dtl@h{\the\dtl@tmplength}%
            \settodepth{\dtl@tmplength}{%
               \DTLbardisplayYticklabel{\dtl@thislabel}}%
            \addtolength{\dtl@tmplength}{\@dtl@h}%
            \addtolength{\dtl@tmplength}{\baselineskip}%
          \fi
          \ifdim\dtl@tmplength>\dtl@yticlabelwidth
            \setlength{\dtl@yticlabelwidth}{\dtl@tmplength}%
          \fi
        }%
      \else
        \@for\dtl@thislabel:=\dtlbar@yticlabels\do{%
          \ifDTLverticalbars
            \settowidth{\dtl@tmplength}{%
              \DTLbardisplayYticklabel{\dtl@thislabel}}%
          \else
            \settoheight{\dtl@tmplength}{%
               \DTLbardisplayYticklabel{\dtl@thislabel}}%
            \edef\@dtl@h{\the\dtl@tmplength}%
            \settodepth{\dtl@tmplength}{%
               \DTLbardisplayYticklabel{\dtl@thislabel}}%
            \addtolength{\dtl@tmplength}{\@dtl@h}%
            \addtolength{\dtl@tmplength}{\baselineskip}%
          \fi
          \ifdim\dtl@tmplength>\dtl@yticlabelwidth
            \setlength{\dtl@yticlabelwidth}{\dtl@tmplength}%
          \fi
        }%
      \fi
    \fi
  \fi
```

Store the width of the bar chart in `\DTLbarchartwidth`

```
\edef\DTLbarchartwidth{\expandafter\number\csname dtlrows@#3\endcsname}
```

Do the bar chart

```
\begin{tikzpicture}
```

Set unit vectors

```
\ifDTLverticalbars
  \pgfsetyvec{\pgfpoint{0pt}{\dtl@unit sp}}%
  \pgfsetxvec{\pgfpoint{\DTLbarwidth}{0pt}}%
\else
```

```
    \pgfsetxvec{\pgfpoint{\dtl@unit sp}{0pt}}%
    \pgfsetyvec{\pgfpoint{0pt}{\DTLbarwidth}}%
  \fi
```

Begin hook

```
  \DTLbaratbegintikz
```

Initialise

```
  \def\@dtl@start{0}%
```

Iterate through data

```
  \@sDTLforeach[#1]{#3}{#4}{%
```

Store the bar number in `\@dtl@bar`

```
  \expandafter\let\expandafter\@dtl@bar
    \csname c@DTLrow\romannumeral\dtlforeachlevel\endcsname%
```

Convert variable to decimal

```
  \expandafter\DTLconverttodecimal\expandafter
    {\DTLbarvariable}{\dtl@variable}%
```

Draw bars

```
  \begin{scope}
   \DTLdocurrentbarcolor
   \ifDTLverticalbars
     \fill (\@dtl@start,0) -- (\@dtl@start,\dtl@variable)
        -- (\@dtl@bar,\dtl@variable) -- (\@dtl@bar,0) -- cycle;
   \else
     \fill (0,\@dtl@start) -- (\dtl@variable,\@dtl@start)
        -- (\dtl@variable,\@dtl@bar) -- (0,\@dtl@bar) -- cycle;
   \fi
  \end{scope}
```

Draw outline

```
  \begin{scope}
  \ifdim\DTLbaroutlinewidth>0pt
   \expandafter\color\expandafter{\DTLbaroutlinecolor}
   \ifDTLverticalbars
     \draw (\@dtl@start,0) -- (\@dtl@start,\dtl@variable)
        -- (\@dtl@bar,\dtl@variable) -- (\@dtl@bar,0) -- cycle;
   \else
     \draw (0,\@dtl@start) -- (\dtl@variable,\@dtl@start)
        -- (\dtl@variable,\@dtl@bar) -- (0,\@dtl@bar) -- cycle;
   \fi
  \fi
  \end{scope}
```

Draw lower x labels

```
  \ifDTLverticalbars
    \edef\dtl@textopt{%
        at={\noexpand\pgfpointadd
              {\noexpand\pgfpointxy{\@dtl@start.5}{0}}
              {\noexpand\pgfpoint{0pt}{-\noexpand\DTLbarlabeloffset}}},
       \DTLbarXlabelalign
    }%
```

Set \DTLstartpt to the starting point.

```
    \edef\DTLstartpt{\noexpand\pgfpointxy{\@dtl@start.5}{0}}%
  \else
    \edef\dtl@textopt{%
        at={\noexpand\pgfpointadd
              {\noexpand\pgfpointxy{0}{\@dtl@start.5}}
              {\noexpand\pgfpoint{-\noexpand\DTLbarlabeloffset}{0pt}}},
       \DTLbarXlabelalign
    }%
```

Set \DTLstartpt to the starting point.

```
    \edef\DTLstartpt{\noexpand\pgfpointxy{0}{\@dtl@start.5}}%
  \fi
   \expandafter\pgftext\expandafter[\dtl@textopt]{%
     \DTLdisplaylowerbarlabel{\dtl@barlabel}}
```

Draw upper x labels

```
  \ifDTLverticalbars
```

Vertical bars

```
    \expandafter\DTLifnumlt\expandafter{\DTLbarvariable}{0}
    {
      \edef\dtl@textopt{%
        at={\noexpand\pgfpointadd
              {\noexpand\pgfpointxy{\@dtl@start.5}{\dtl@variable}}
              {\noexpand\pgfpoint{0pt}{-\noexpand\DTLbarlabeloffset}}}
      }%
    }{%
      \edef\dtl@textopt{%
        at={\noexpand\pgfpointadd
              {\noexpand\pgfpointxy{\@dtl@start.5}{\dtl@variable}}
              {\noexpand\pgfpoint{0pt}{\noexpand\DTLbarlabeloffset}}}
      }%
    }
```

Set \DTLendpt to the end point.

```
    \edef\DTLendpt{\noexpand\pgfpointxy{\@dtl@start.5}{\dtl@variable}}%
  \else
```

Horizontal bars

```
    \expandafter\DTLifnumlt\expandafter{\DTLbarvariable}{0}
    {
      \edef\dtl@textopt{right,
        at={\noexpand\pgfpointadd
              {\noexpand\pgfpointxy{\dtl@variable}{\@dtl@start.5}}
              {\noexpand\pgfpoint{-\noexpand\DTLbarlabeloffset}{0pt}}}
      }%
    }{%
      \edef\dtl@textopt{left,
        at={\noexpand\pgfpointadd
              {\noexpand\pgfpointxy{\dtl@variable}{\@dtl@start.5}}
              {\noexpand\pgfpoint{\noexpand\DTLbarlabeloffset}{0pt}}}
      }%
    }
```

Set \DTLendpt to the end point.

```
    \edef\DTLendpt{\noexpand\pgfpointxy{\dtl@variable}{\@dtl@start.5}}%
  \fi
   \expandafter\pgftext\expandafter[\dtl@textopt]{%
      \DTLdisplayupperbarlabel{\dtl@upperbarlabel}}
```

Set the mid point

```
  \def\DTLmidpt{\pgfpointlineattime{0.5}{\DTLstartpt}{\DTLendpt}}%
```

Do every bar hook

```
    \DTLeverybarhook
```

End of loop

```
    \edef\@dtl@start{\number\@dtl@bar}%
  }
```

Draw x axis

```
  \ifDTLbarxaxis
    \ifDTLverticalbars
      \expandafter\draw\expandafter[\DTLBarXAxisStyle]
        (0,0) -- (\DTLbarchartwidth,0);
    \else
      \expandafter\draw\expandafter[\DTLBarXAxisStyle]
        (0,0) -- (0,\DTLbarchartwidth);
    \fi
  \fi
```

Draw y axis

```
  \ifDTLbaryaxis
    \ifDTLverticalbars
      \expandafter\draw\expandafter[\DTLBarYAxisStyle]
        (0,\DTLnegextent) -- (0,\DTLbarmax);
    \else
      \expandafter\draw\expandafter[\DTLBarYAxisStyle]
        (\DTLnegextent,0) -- (\DTLbarmax,0);
    \fi
  \fi
```

Plot y tick marks if required

```
  \ifx\dtlbar@yticlist\relax
  \else
    \@for\dtl@thistick:=\dtlbar@yticlist\do{%
      \ifDTLverticalbars
        \pgfpathmoveto{\pgfpointxy{0}{\dtl@thistick}}
        \pgfpathlineto{
          \pgfpointadd{\pgfpointxy{0}{\dtl@thistick}}
                      {\pgfpoint{-\DTLticklength}{0pt}}}
      \else
        \pgfpathmoveto{\pgfpointxy{\dtl@thistick}{0}}
        \pgfpathlineto{
          \pgfpointadd{\pgfpointxy{\dtl@thistick}{0}}
                      {\pgfpoint{0pt}{-\DTLticklength}}}
      \fi
      \pgfusepath{stroke}
      \ifx\dtlbar@yticlabels\relax
         \FPround{\dtl@thislabel}{\dtl@thistick}
                 {\c@DTLbarroundvar}%
```

```
      \else
         \dtl@chopfirst\dtlbar@yticlabels\dtl@thislabel\dtl@rest
         \let\dtlbar@yticlabels=\dtl@rest
      \fi
      \ifDTLverticalbars
        \edef\dtl@textopt{\DTLbarYticklabelalign,%
          at={\noexpand\pgfpointadd
               {\noexpand\pgfpointxy{0}{\dtl@thistick}}
               {\noexpand\pgfpoint{-\noexpand\DTLticklabeloffset}{0pt}},
          }}%
      \else
        \edef\dtl@textopt{\DTLbarYticklabelalign,
          at={\noexpand\pgfpointadd
               {\noexpand\pgfpointxy{\dtl@thistick}{0}}
               {\noexpand\pgfpoint{0pt}{-\noexpand\DTLticklabeloffset}}
          }}%
      \fi
      \expandafter\pgftext\expandafter[\dtl@textopt]{%
        \DTLbardisplayYticklabel{\dtl@thislabel}}
    }%
  \fi
```

Plot the y label if required

```
  \ifx\dtlbar@ylabel\relax
  \else
    \addtolength{\dtl@yticlabelwidth}{\baselineskip}%
    \setlength{\dtl@tmplength}{0.5\DTLbarchartlength}
    \ifDTLverticalbars
      \pgftext[bottom,center,at={\pgfpointadd
          {\pgfpointxy{0}{\DTLnegextent}}%
          {\pgfpoint{-\dtl@yticlabelwidth}{\dtl@tmplength}}},
          rotate=90]{%
         \dtlbar@ylabel}
    \else
      \pgftext[bottom,center,at={\pgfpointadd
          {\pgfpointxy{\DTLnegextent}{0}}%
          {\pgfpoint{\dtl@tmplength}{-\dtl@yticlabelwidth}}}]{%
         \dtlbar@ylabel}
    \fi
  \fi
```

Finish bar chart

```
  \DTLbaratendtikz
  \end{tikzpicture}
  \fi
  }}
```

\DTLmultibarchart

\DTLmultibarchart[⟨*conditions*⟩]{⟨*option list*⟩}{⟨*db name*⟩}{⟨*assign list*⟩}

Make a multi-bar chart from data given in data base ⟨*db name*⟩, where ⟨*assign list*⟩ is a comma-separated list of ⟨*cmd*⟩=⟨*key*⟩ pairs. ⟨*option list*⟩ must include the `variables` key which must be a comma separated list of commands, where

each command is included in ⟨*assign list*⟩. The optional argument ⟨*conditions*⟩ is the same as that for \DTLforeach.

```
\newcommand*{\DTLmultibarchart}[4][\boolean{true}]{%
{\let\dtlbar@variables=\relax
\let\DTLbarmax=\relax
\let\DTLnegextent=\relax
\disable@keys{databar}{variable,upperbarlabel}%
\setkeys{databar}{#2}%
\ifx\dtlbar@variables\relax
  \PackageError{databar}{\string\DTLmultibarchart\space missing variables setting}{}%
\else
```

Compute the maximum bar height, unless \DTLbarmax has been set.

```
  \ifx\DTLbarmax\relax
    \@sDTLforeach[#1]{#3}{#4}{%
      \@for\DTLbarvariable:=\dtlbar@variables\do{%
        \expandafter\DTLconverttodecimal\expandafter
          {\DTLbarvariable}{\dtl@barvar}%
        \ifx\DTLbarmax\relax
          \let\DTLbarmax=\dtl@barvar
        \else
          \let\dtl@old=\DTLbarmax
          \FPmax{\DTLbarmax}{\dtl@old}{\dtl@barvar}%
        \fi
      }%
    }%
    \ifx\dtlbar@yticgap\relax
    \else
      \let\dtl@thistick=\dtlbar@yticgap%
      \whiledo{\DTLisFPopenbetween{\dtl@thistick}{0}{\DTLbarmax}}{%
        \FPadd{\dtl@thistick}{\dtl@thistick}{\dtlbar@yticgap}%
      }%
      \let\DTLbarmax=\dtl@thistick
    \fi
  \fi
```

Compute the bar depth, unless \DTLnegextent has been set.

```
  \ifx\DTLnegextent\relax
    \def\DTLnegextent{0}%
    \@sDTLforeach[#1]{#3}{#4}{%
      \@for\DTLbarvariable:=\dtlbar@variables\do{%
        \expandafter\DTLconverttodecimal\expandafter
          {\DTLbarvariable}{\dtl@barvar}%
        \let\dtl@old=\DTLnegextent
        \DTLmin{\DTLnegextent}{\dtl@old}{\dtl@barvar}%
      }%
    }%
    \ifx\dtlbar@yticgap\relax
    \else
      \ifthenelse{\DTLisFPlt{\DTLnegextent}{0}}{%
      \edef\dtl@thistick{0}%
      \whiledo{\DTLisFPclosedbetween{\dtl@thistick}{\DTLnegextent}{0}}{%
        \FPsub{\dtl@thistick}{\dtl@thistick}{\dtlbar@yticgap}%
      }%
      \let\DTLnegextent=\dtl@thistick
```

```
      }{}%
    \fi
  \fi
```

Determine scaling factor

```
  \@dtl@tmpcount=\DTLbarchartlength
  \FPsub{\dtl@extent}{\DTLbarmax}{\DTLnegextent}%
  \FPdiv{\dtl@unit}{\number\@dtl@tmpcount}{\dtl@extent}%
```

Construct y tick list if required

```
  \setlength{\dtl@yticlabelwidth}{0pt}%
  \ifDTLbarytics
    \ifx\dtlbar@yticlist\relax
      \ifx\dtlbar@yticgap\relax
        \dtl@constructticklist\DTLnegextent\DTLbarmax
          \dtl@unit\dtlbar@yticlist
      \else
        \dtl@constructticklistwithgapz
          \DTLnegextent\DTLbarmax\dtlbar@yticlist\dtlbar@yticgap
      \fi
    \fi
    \ifx\dtlbar@ylabel\relax
    \else
      \ifx\dtlbar@yticlabels\relax
        \@for\dtl@thislabel:=\dtlbar@yticlist\do{%
          \FPround{\dtl@thislabel}{\dtl@thislabel}
                  {\c@DTLbarroundvar}%
          \ifDTLverticalbars
            \settowidth{\dtl@tmplength}{%
               \DTLbardisplayYticklabel{\dtl@thislabel}}%
          \else
            \settoheight{\dtl@tmplength}{%
               \DTLbardisplayYticklabel{\dtl@thislabel}}%
            \edef\@dtl@h{\the\dtl@tmplength}%
            \settodepth{\dtl@tmplength}{%
               \DTLbardisplayYticklabel{\dtl@thislabel}}%
            \addtolength{\dtl@tmplength}{\@dtl@h}%
            \addtolength{\dtl@tmplength}{\baselineskip}%
          \fi
          \ifdim\dtl@tmplength>\dtl@yticlabelwidth
            \setlength{\dtl@yticlabelwidth}{\dtl@tmplength}%
          \fi
        }%
      \else
        \@for\dtl@thislabel:=\dtlbar@yticlabels\do{%
          \ifDTLverticalbars
            \settowidth{\dtl@tmplength}{%
              \DTLbardisplayYticklabel{\dtl@thislabel}}%
          \else
            \settoheight{\dtl@tmplength}{%
               \DTLbardisplayYticklabel{\dtl@thislabel}}%
            \edef\@dtl@h{\the\dtl@tmplength}%
            \settodepth{\dtl@tmplength}{%
               \DTLbardisplayYticklabel{\dtl@thislabel}}%
            \addtolength{\dtl@tmplength}{\@dtl@h}%
```

```
            \addtolength{\dtl@tmplength}{\baselineskip}%
          \fi
          \ifdim\dtl@tmplength>\dtl@yticlabelwidth
            \setlength{\dtl@yticlabelwidth}{\dtl@tmplength}%
          \fi
        }%
      \fi
    \fi
  \fi
```

Calculate the offset for the lower label and number of labels

```
\dtl@xticlabelheight=0pt\relax
\@dtl@tmpcount=0\relax
\@for\dtl@thislabel:=\dtl@multibarlabels\do{%
  \advance\@dtl@tmpcount by 1\relax
  \settoheight{\dtl@tmplength}{\tikz\expandafter\pgftext\expandafter
    [\DTLbarXlabelalign]{\DTLdisplaylowerbarlabel{\dtl@thislabel}};}%
  \edef\@dtl@h{\the\dtl@tmplength}%
  \settodepth{\dtl@tmplength}{\tikz\expandafter\pgftext\expandafter
    [\DTLbarXlabelalign]{\DTLdisplaylowerbarlabel{\dtl@thislabel}};}%
  \addtolength{\dtl@tmplength}{\@dtl@h}%
  \addtolength{\dtl@tmplength}{\baselineskip}%
  \ifdim\dtl@tmplength>\dtl@xticlabelheight
    \setlength{\dtl@xticlabelheight}{\dtl@tmplength}%
  \fi
}
```

Calculate number of bars per group

```
\@dtl@tmpcount=0\relax
\@for\dtl@this:=\dtlbar@variables\do{%
  \advance\@dtl@tmpcount by 1\relax
}%
\edef\DTLbargroupwidth{\number\@dtl@tmpcount}%
```

Compute the total width of the bar chart (in terms of the x unit vector.)

```
\edef\dtl@n{\expandafter\number\csname dtlrows@#3\endcsname}
\FPmul{\dtl@tmpi}{\dtl@n}{\DTLbargroupwidth}
\FPsub{\dtl@tmpii}{\dtl@n}{1}%
\FPmul{\dtl@tmpii}{\dtl@tmpii}{\dtlbar@groupgap}%
\FPadd{\DTLbarchartwidth}{\dtl@tmpi}{\dtl@tmpii}
```

Do the bar chart

```
\begin{tikzpicture}
```

Set unit vectors

```
\ifDTLverticalbars
  \pgfsetyvec{\pgfpoint{0pt}{\dtl@unit sp}}%
  \pgfsetxvec{\pgfpoint{\DTLbarwidth}{0pt}}%
\else
  \pgfsetxvec{\pgfpoint{\dtl@unit sp}{0pt}}%
  \pgfsetyvec{\pgfpoint{0pt}{\DTLbarwidth}}%
\fi
```

Begin hook

```
\DTLbaratbegintikz
```

Initialise

```
\def\@dtl@start{0}%
```

Iterate through data

```
\@sDTLforeach[#1]{#3}{#4}{%
```

Store the bar number in `\@dtl@bar`

```
\@dtl@barcount = 1\relax
```

Set the multibar label lists

```
\let\dtl@multibar@labels=\dtl@multibarlabels
\let\dtl@uppermultibar@labels=\dtl@uppermultibarlabels
```

Compute mid point over group

```
\FPmul{\dtl@multimidpt}{\DTLbargroupwidth}{0.5}%
\FPadd{\dtl@multimidpt}{\dtl@multimidpt}{\@dtl@start}%
```

Iterate through each variable

```
\@for\DTLbarvariable:=\dtlbar@variables\do{%
```

Set end point

```
\FPadd{\@dtl@endpt}{\@dtl@start}{1}%
```

Convert variable to decimal

```
\expandafter\DTLconverttodecimal\expandafter
  {\DTLbarvariable}{\dtl@variable}%
```

Get the current lower label:

```
\dtl@chopfirst\dtl@multibar@labels\dtl@thisbarlabel\dtl@rest
\let\dtl@multibar@labels=\dtl@rest
```

Get the current upper label:

```
\dtl@chopfirst\dtl@uppermultibar@labels\dtl@thisupperbarlabel\dtl@rest
\let\dtl@uppermultibar@labels=\dtl@rest
```

Draw bars

```
\begin{scope}
 \expandafter\color\expandafter{\DTLgetbarcolor{\@dtl@barcount}}%
 \ifDTLverticalbars
   \fill (\@dtl@start,0) -- (\@dtl@start,\dtl@variable)
      -- (\@dtl@endpt,\dtl@variable) -- (\@dtl@endpt,0) -- cycle;
 \else
   \fill (0,\@dtl@start) -- (\dtl@variable,\@dtl@start)
      -- (\dtl@variable,\@dtl@endpt) -- (0,\@dtl@endpt) -- cycle;
 \fi
\end{scope}
```

Draw outline

```
\begin{scope}
\ifdim\DTLbaroutlinewidth>0pt
 \expandafter\color\expandafter{\DTLbaroutlinecolor}
 \ifDTLverticalbars
   \draw (\@dtl@start,0) -- (\@dtl@start,\dtl@variable)
      -- (\@dtl@endpt,\dtl@variable) -- (\@dtl@endpt,0) -- cycle;
 \else
   \draw (0,\@dtl@start) -- (\dtl@variable,\@dtl@start)
      -- (\dtl@variable,\@dtl@endpt) -- (0,\@dtl@endpt) -- cycle;
 \fi
```

```
\fi
\end{scope}
```

Calculate mid point

```
\FPadd{\@dtl@midpt}{\@dtl@start}{0.5}%
```

Draw lower x labels

```
\ifDTLverticalbars
  \edef\dtl@textopt{%
      at={\noexpand\pgfpointadd
           {\noexpand\pgfpointxy{\@dtl@midpt}{0}}
           {\noexpand\pgfpoint{0pt}{-\noexpand\DTLbarlabeloffset}}},
     \DTLbarXlabelalign
  }%
  \edef\DTLstartpt{\noexpand\pgfpointxy{\@dtl@midpt}{0}}%
\else
  \edef\dtl@textopt{%
      at={\noexpand\pgfpointadd
           {\noexpand\pgfpointxy{0}{\@dtl@midpt}}
           {\noexpand\pgfpoint{-\noexpand\DTLbarlabeloffset}{0pt}}},
     \DTLbarXlabelalign
  }%
  \edef\DTLstartpt{\noexpand\pgfpointxy{0}{\@dtl@midpt}}%
\fi
 \expandafter\pgftext\expandafter[\dtl@textopt]{%
   \DTLdisplaylowermultibarlabel{\dtl@thisbarlabel}}
```

Draw upper x labels

```
\ifDTLverticalbars
  \expandafter\DTLifnumlt\expandafter{\DTLbarvariable}{0}
  {
    \edef\dtl@textopt{%
      at={\noexpand\pgfpointadd
           {\noexpand\pgfpointxy{\@dtl@midpt}{\dtl@variable}}
           {\noexpand\pgfpoint{0pt}{-\noexpand\DTLbarlabeloffset}}}
    }%
  }{%
    \edef\dtl@textopt{%
      at={\noexpand\pgfpointadd
           {\noexpand\pgfpointxy{\@dtl@midpt}{\dtl@variable}}
           {\noexpand\pgfpoint{0pt}{\noexpand\DTLbarlabeloffset}}}
    }%
  }
  \edef\DTLendpt{\noexpand\pgfpointxy{\@dtl@midpt}{\dtl@variable}}%
\else
  \expandafter\DTLifnumlt\expandafter{\DTLbarvariable}{0}
  {
    \edef\dtl@textopt{right,
      at={\noexpand\pgfpointadd
           {\noexpand\pgfpointxy{\dtl@variable}{\@dtl@midpt}}
           {\noexpand\pgfpoint{-\noexpand\DTLbarlabeloffset}{0pt}}}
    }%
  }{%
    \edef\dtl@textopt{left,
      at={\noexpand\pgfpointadd
```

```
            {\noexpand\pgfpointxy{\dtl@variable}{\@dtl@midpt}}
            {\noexpand\pgfpoint{\noexpand\DTLbarlabeloffset}{0pt}}}
      }%
    }
    \edef\DTLendpt{\noexpand\pgfpointxy{\dtl@variable}{\@dtl@midpt}}%
  \fi
   \expandafter\pgftext\expandafter[\dtl@textopt]{%
      \DTLdisplayuppermultibarlabel{\dtl@thisupperbarlabel}}
```

Set the mid point

```
  \def\DTLmidpt{\pgfpointlineattime{0.5}{\DTLstartpt}{\DTLendpt}}%
```

Do every bar hook

```
  \DTLeverybarhook
```

End of loop increment loop variables

```
    \FPadd{\@dtl@start}{\@dtl@start}{1}%
    \advance\@dtl@barcount by 1\relax
  }%
  % Draw lower group $x$ labels
  %    \begin{macrocode}
  \setlength{\dtl@tmplength}{\DTLbarlabeloffset}%
  \addtolength{\dtl@tmplength}{\dtl@xticlabelheight}%
  \ifDTLverticalbars
    \edef\dtl@textopt{%
        at={\noexpand\pgfpointadd
            {\noexpand\pgfpointxy{\dtl@multimidpt}{0}}
            {\noexpand\pgfpoint{0pt}{-\noexpand\dtl@tmplength}}},
      \DTLbarXlabelalign
    }%
  \else
    \edef\dtl@textopt{%
        at={\noexpand\pgfpointadd
            {\noexpand\pgfpointxy{0}{\dtl@multimidpt}}
            {\noexpand\pgfpoint{-\noexpand\dtl@tmplength}{0pt}}},
      \DTLbarXlabelalign
    }%
  \fi
   \expandafter\pgftext\expandafter[\dtl@textopt]{%
     \DTLdisplaylowerbarlabel{\dtl@barlabel}}
```

Increment starting position by `\dtlbar@groupgap`

```
    \FPadd{\@dtl@start}{\@dtl@start}{\dtlbar@groupgap}%
  }
```

Draw x axis

```
  \ifDTLbarxaxis
    \ifDTLverticalbars
      \expandafter\draw\expandafter[\DTLBarXAxisStyle]
        (0,0) -- (\DTLbarchartwidth,0);
    \else
      \expandafter\draw\expandafter[\DTLBarXAxisStyle]
        (0,0) -- (0,\DTLbarchartwidth);
    \fi
  \fi
```

Draw y axis

```
\ifDTLbaryaxis
  \ifDTLverticalbars
    \expandafter\draw\expandafter[\DTLBarYAxisStyle]
      (0,\DTLnegextent) -- (0,\DTLbarmax);
  \else
    \expandafter\draw\expandafter[\DTLBarYAxisStyle]
      (\DTLnegextent,0) -- (\DTLbarmax,0);
  \fi
\fi
```

Plot y tick marks if required

```
\ifx\dtlbar@yticlist\relax
\else
  \@for\dtl@thistick:=\dtlbar@yticlist\do{%
    \ifDTLverticalbars
      \pgfpathmoveto{\pgfpointxy{0}{\dtl@thistick}}
      \pgfpathlineto{
        \pgfpointadd{\pgfpointxy{0}{\dtl@thistick}}
                    {\pgfpoint{-\DTLticklength}{0pt}}}
    \else
      \pgfpathmoveto{\pgfpointxy{\dtl@thistick}{0}}
      \pgfpathlineto{
        \pgfpointadd{\pgfpointxy{\dtl@thistick}{0}}
                    {\pgfpoint{0pt}{-\DTLticklength}}}
    \fi
    \pgfusepath{stroke}
    \ifx\dtlbar@yticlabels\relax
       \FPround{\dtl@thislabel}{\dtl@thistick}
               {\c@DTLbarroundvar}%
    \else
       \dtl@chopfirst\dtlbar@yticlabels\dtl@thislabel\dtl@rest
       \let\dtlbar@yticlabels=\dtl@rest
    \fi
    \ifDTLverticalbars
      \edef\dtl@textopt{\DTLbarYticklabelalign,%
        at={\noexpand\pgfpointadd
             {\noexpand\pgfpointxy{0}{\dtl@thistick}}
             {\noexpand\pgfpoint{-\noexpand\DTLticklabeloffset}{0pt}},
        }}%
    \else
      \edef\dtl@textopt{\DTLbarYticklabelalign,
        at={\noexpand\pgfpointadd
             {\noexpand\pgfpointxy{\dtl@thistick}{0}}
             {\noexpand\pgfpoint{0pt}{-\noexpand\DTLticklabeloffset}}
        }}%
    \fi
    \expandafter\pgftext\expandafter[\dtl@textopt]{%
      \DTLbardisplayYticklabel{\dtl@thislabel}}
  }%
\fi
```

Plot the y label if required

```
\ifx\dtlbar@ylabel\relax
\else
```

```
    \addtolength{\dtl@yticlabelwidth}{\baselineskip}%
    \setlength{\dtl@tmplength}{0.5\DTLbarchartlength}
    \ifDTLverticalbars
      \pgftext[bottom,center,at={\pgfpointadd
          {\pgfpointxy{0}{\DTLnegextent}}%
          {\pgfpoint{-\dtl@yticlabelwidth}{\dtl@tmplength}}},
          rotate=90]{%
         \dtlbar@ylabel}
    \else
      \pgftext[bottom,center,at={\pgfpointadd
          {\pgfpointxy{\DTLnegextent}{0}}%
          {\pgfpoint{\dtl@tmplength}{-\dtl@yticlabelwidth}}}]{%
         \dtlbar@ylabel}
    \fi
  \fi
```

Finish bar chart

```
  \DTLbaratendtikz
  \end{tikzpicture}
  \fi
  }}
```

14 databib.sty

14.1 Package Declaration

```
  \NeedsTeXFormat{LaTeX2e}
  \ProvidesPackage{databib}[2009/02/27 v2.0 (NLCT)]
```

Load required packages:

```
  \RequirePackage{datatool}
```

14.2 Package Options

`\dtlbib@style` The default bib style is stored in `\dtlbib@style`.

```
  \newcommand*{\dtlbib@style}{plain}
```

The style package option sets `\dtlbib@style`.

```
  \define@choicekey{databib.sty}{style}{plain,abbrv,alpha}{%
  \def\dtlbib@style{#1}}
```

Process package options:

```
  \ProcessOptionsX
```

14.3 Loading BBL file

`\DTLloadbbl` `\DTLloadbib[`⟨*bbl file*⟩`]{`⟨*db name*⟩`}{`⟨*bib list*⟩`}`

```
  \newcommand*{\DTLloadbbl}[3][\jobname.bbl]{%
  \bibliographystyle{databib}%
  \if@filesw
   \immediate\write\@auxout{\string\bibdata{#3}}%
  \fi
```

```
\DTLnewdb{#2}%
\edef\DTLBIBdbname{#2}%
\@input@{#1}}
```

\DTLnewbibrow

\DTLnewbibrow adds a new row to the bibliography database. (\DTLBIBdbname must be set prior to use to the name of the datatool database which must exist. Any check to determine its existence should be performed when \DTLBIBdbname is set.)

```
\newcommand*{\DTLnewbibrow}{\@DTLnewrow{\DTLBIBdbname}}
```

\DTLnewbibitem

\DTLnewbibitem{⟨*key*⟩}{⟨*value*⟩}

Adds a new database entry with the given key and value.

```
\newcommand*{\DTLnewbibitem}[2]{%
  \@DTLnewdbentry{\DTLBIBdbname}{#1}{#2}}
```

14.4 Predefined text

\andname

```
\providecommand*{\andname}{and}
```

\ofname

```
\providecommand*{\ofname}{of}
```

\inname

```
\providecommand*{\inname}{in}
```

\etalname

```
\providecommand*{\etalname}{et al.}
```

\editorname

```
\providecommand*{\editorname}{editor}
```

\editorsname

```
\providecommand*{\editorsname}{editors}
```

\volumename

```
\providecommand*{\volumename}{volume}
```

\numbername

```
\providecommand*{\numbername}{number}
```

\pagesname

```
\providecommand*{\pagesname}{pages}
```

\pagename

```
\providecommand*{\pagename}{page}
```

\editionname

```
\providecommand*{\editionname}{edition}
```

\techreportname

```
\providecommand*{\techreportname}{Technical report}
```

\mscthesisname

```
\providecommand*{\mscthesisname}{Master's thesis}
```

\phdthesisname

```
\providecommand*{\phdthesisname}{PhD thesis}
```

14.5 Displaying the bibliography

\DTLbibliography{⟨*bib dbname*⟩}

Displays the bibliography for the database ⟨*bib dbname*⟩ which must have previously been loaded using \DTLloadbbl.

\DTLbibliography

```
\newcommand*{\DTLbibliography}[2][\boolean{true}]{%
  \begin{DTLthebibliography}[#1]{#2}%
  \DTLforeachbibentry[#1]{#2}{%
    \DTLbibitem \DTLformatbibentry \DTLendbibitem
  }%
  \end{DTLthebibliography}%
}
```

\DTLformatbibentry

\DTLformatbibentry

Formats the current bib entry.

```
\newcommand*{\DTLformatbibentry}{%
```

Check format for this type is defined.

```
  \@ifundefined{DTLformat\DBIBentrytype}%
  {%
    \PackageError{databib}{Don't know how to format bibliography
    entries of type '\DBIBentrytype'}{}%
  }%
  {%
```

Print information to terminal and log file if in verbose mode.

```
    \dtl@message{[\DBIBcitekey]}%
```

Initialise

```
    \DTLstartsentencefalse\DTLmidsentencefalse\DTLperiodfalse
```

Format this entry

```
    \csname DTLformat\DBIBentrytype\endcsname
  }%
}
```

\DTLendbibitem

Hook to add extra information at the end of a bibliography item. This does nothing by default.

```
\newcommand*{\DTLendbibitem}{}
```

\DTLwidest Define a length to store the widest bib entry label

```
\newlength\dtl@widest
```

\DTLcomputewidestbibentry

\DTLcomputewidestbibentry{⟨*conditions*⟩}{⟨*db name*⟩}{⟨*bib label*⟩}{⟨*cmd*⟩}

Computes the widest bibliography entry over all entries satisfying ⟨*condition*⟩ for the database called ⟨*db name*⟩, where the bibliography label is formated according to ⟨*bib label*⟩ and stores the result in ⟨*cmd*⟩ which must be a command name.

```
\newcommand*{\DTLcomputewidestbibentry}[4]{%
\dtl@widest=0pt\relax
\let#4=\@empty
\DTLforeachbibentry[#1]{#2}{%
\settowidth{\dtl@tmplength}{#3}%
\ifdim\dtl@tmplength>\dtl@widest\relax
 \dtl@widest=\dtl@tmplength
 \protected@edef#4{#3}%
\fi
}%
}
```

\DTLforeachbibentry

\DTLforeachbibentry[⟨*condition*⟩]{⟨*db name*⟩}{⟨*text*⟩}

\DTLforeachbibentry*[⟨*condition*⟩]{⟨*db name*⟩}{⟨*text*⟩}

Iterates through the database called ⟨*db name*⟩ and does ⟨*text*⟩ if ⟨*condition*⟩ is met. As with \DTLforeach, the starred version is read only.

```
\newcommand*{\DTLforeachbibentry}{%
\@ifstar\@DTLforeachbibentry\@DTLforeachbibentry}
```

\@DTLforeachbibentry Unstarred version

```
\newcommand*{\@DTLforeachbibentry}[3][\boolean{true}]{%
```

Store database name.

```
  \edef\DBIBname{#2}%
```

Reset row counter.

```
  \setcounter{DTLbibrow}{0}%
```

Iterate through the database.

```
  \@DTLforeach{#2}{\DBIBcitekey=CiteKey,\DBIBentrytype=EntryType}%
  {%
    \dtl@gathervalues{#2}{\dtlcurrentrow}%
    \ifthenelse{#1}{\refstepcounter{DTLbibrow}#3}{}%
  }%
}
```

\@sDTLforeachbibentry Starred version

```
\newcommand*{\@sDTLforeachbibentry}[3][\boolean{true}]{%
```

Store database name.

```
\edef\DBIBname{#2}%
```

Reset row counter.

```
\setcounter{DTLbibrow}{0}%
```

Iterate through the database (read only).

```
\@sDTLforeach{#2}{\DBIBcitekey=CiteKey,\DBIBentrytype=EntryType}%
{%
  \dtl@gathervalues{#2}{\dtlcurrentrow}%
  \ifthenelse{#1}{\refstepcounter{DTLbibrow}#3}{}%
}%
}
```

The counter DTLbibrow keeps track of the current row in the body of \DTLforeachbibentry. (You can't rely on DTLrowi, DTLrowii and DTLrowiii, as \DTLforeachbibentry pass the conditions to the optional argument of \DTLforeach, but instead uses \ifthenelse, which means that DTLrowi etc will be incremented, even when the given condition is not met.)

```
\newcounter{DTLbibrow}
```

Keep hyperref happy:

```
\def\theHDTLbibrow{\theHDTLrow.bib.\arabic{DTLbibrow}}%
```

\DTLbibfield

\DTLbibfield{⟨*field name*⟩}

Gets the value assigned to the field ⟨*field name*⟩ for the current row of \DTLforeachbibentry. (Doesn't check if the field exists, or if it is being used within \DTLforeachbibentry.)

```
\newcommand*{\DTLbibfield}[1]{\csname @dtl@key@#1\endcsname}
```

\DTLifbibfieldexists

\DTLifbibfieldexists{⟨*field name*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Determines whether the given field name exists for the current row of \DTLforeachbibentry.

```
\newcommand*{\DTLifbibfieldexists}[3]{%
\@ifundefined{@dtl@key@#1}{#3}{%
\expandafter\DTLifnull\csname @dtl@key@#1\endcsname
{#3}{#2}}}
```

\DTLifanybibfieldexists

\DTLifanybibfieldexists{⟨*list of field name*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Determines whether any of the listed fields exist for the current row of \DTLforeachbibentry.

```
\newcommand*{\DTLifanybibfieldexists}[3]{%
\@for\dtl@thisfield:=#1\do{%
\@ifundefined{@dtl@key@\dtl@thisfield}{}{%
\expandafter\DTLifnull\csname @dtl@key@\dtl@thisfield\endcsname
{}{%
```

```
\@endfortrue}}}%
\if@endfor
 #2%
\else
 #3%
\fi
\@endforfalse
}
```

\ifDTLperiod The conditional \ifDTLperiod is used to keep track of any abbreviations ending with a period, this is to ensure that abbreviations aren't followed by a full stop if they already have a full stop terminating the abbreviation.

```
\newif\ifDTLperiod
```

\DTLcheckendsperiod

\DTLcheckendperiod{⟨*string*⟩}

Checks if ⟨*string*⟩ ends with a full stop. This sets \ifDTLperiod.

```
\newcommand*{\DTLcheckendsperiod}[1]{%
\dtl@checkendsperiod#1\@nil\relax}
```

```
\def\dtl@checkendsperiod#1#2{%
\def\@dtl@argi{#1}\def\@dtl@argii{#2}%
\def\@dtl@period{.}%
\ifx\@dtl@argi\@nnil
  \global\DTLperiodfalse
  \let\@dtl@donext=\relax
\else
  \ifx\@dtl@argii\@nnil
    \ifx\@dtl@argi\@dtl@period
      \global\DTLperiodtrue
    \else
      \global\DTLperiodfalse
    \fi
    \let\@dtl@donext=\@gobble
  \else
    \let\@dtl@donext=\dtl@checkendsperiod
  \fi
\fi
\@dtl@donext{#2}%
}
```

\DTLcheckbibfieldendsperiod

\DTLcheckbibfieldendperiod{⟨*label*⟩}

Checks if the bib field ⟨*label*⟩ ends with a full stop. This sets \ifDTLperiod.

```
\newcommand*{\DTLcheckbibfieldendsperiod}[1]{%
\protected@edef\@dtl@tmp{\DTLbibfield{#1}}%
\expandafter\DTLcheckendsperiod\expandafter{\@dtl@tmp}}
```

\ifDTLmidsentence

\ifDTLmidsentence

Determine whether we are in the middle of a sentence.

```
\newif\ifDTLmidsentence
```

\ifDTLstartsentence

```
\ifDTLstartsentence
```

Determine whether we are at the start of a sentence.

```
\newif\ifDTLstartsentence
```

\DTLaddperiod

```
\DTLaddperiod
```

Adds a full stop and sets \DTLmidsentencefalse, \DTLstartsentencetrue and \DTLperiodfalse.

```
\newcommand*{\DTLaddperiod}{\DTLmidsentencefalse\DTLperiodfalse
\DTLstartsentencetrue
\ifDTLperiod\else.\fi}
```

\DTLaddcomma

```
\DTLaddcomma
```

Adds a comma and sets \DTLmidsentencetrue, \DTLperiodfalse and \DTLstartsentencefalse

```
\newcommand*{\DTLaddcomma}{, \DTLmidsentencetrue
\DTLperiodfalse\DTLstartsentencefalse}
```

\DTLstartsentencespace Adds a space if at the start of the sentence, otherwise does nothing. (The space between sentences is added this way, rather than in \DTLaddperiod otherwise spurious extra space can occur at the end of the bib item. The spacefactor needs to be set manually, because there's stuff in the way of the previous sentence's full stop and this space which confuses the inter sentence spacing (and, of course, the previous sentence could have ended with a capital letter.)

```
\newcommand*{\DTLstartsentencespace}{%
\ifDTLstartsentence\spacefactor=\sfcode`\.\relax\space
\fi\DTLstartsentencefalse}
```

\DTLtwoand In a list of only two author (or editor) names, the text between the two names is given by \DTLtwoand:

```
\newcommand*{\DTLtwoand}{\ \andname\ }
```

\DTLandlast In a list of author (or editor) names, the text between the penultimate and last name is given by \DTLandlast:

```
\newcommand*{\DTLandlast}{, \andname\ }
```

\DTLandnotlast In a list of author (or editor) names, the text between the names (except the penultimate and last name) is given by \DTLandnotlast:

```
\newcommand*{\DTLandnotlast}{, }
```

\dtl@authorcount Define a count register to keep track of the number of authors:

```
\newcount\@dtl@authorcount
```

The counter DTLmaxauthors indicates the maximum number of author names to display, if there are more than that number, \etalname is used.

```
\newcounter{DTLmaxauthors}
\setcounter{DTLmaxauthors}{10}
```

\DTLformatauthorlist Format a list of author names (the list is stored in \@dtl@key@Author):

```
\newcommand*{\DTLformatauthorlist}{%
\DTLifbibfieldexists{Author}{%
\DTLstartsentencespace
\@dtl@authorcount=0\relax
\@for\@dtl@author:=\@dtl@key@Author\do{%
\advance\@dtl@authorcount by 1\relax}%
\@dtl@tmpcount=0\relax
\ifnum\@dtl@authorcount>\c@DTLmaxauthors
{%
  \@for\@dtl@author:=\@dtl@key@Author\do{%
  \advance\@dtl@tmpcount by 1\relax
  \ifnum\@dtl@tmpcount=1\relax
   \expandafter\DTLformatauthor\@dtl@author
  \else
    \ifnum\@dtl@tmpcount>\c@DTLmaxauthors
      \DTLandnotlast \etalname
      \expandafter\DTLcheckendsperiod\expandafter{\etalname}%
      \@endfortrue
    \else
      \DTLandnotlast \expandafter\DTLformatauthor\@dtl@author
    \fi
  \fi
  }%
}%
\else
  \@for\@dtl@author:=\@dtl@key@Author\do{%
  \advance\@dtl@tmpcount by 1\relax
  \ifnum\@dtl@tmpcount=1\relax
   \expandafter\DTLformatauthor\@dtl@author
  \else
    \ifnum\@dtl@tmpcount=\@dtl@authorcount
      \ifnum\@dtl@authorcount=2\relax
        \DTLtwoand
      \else
        \DTLandlast
      \fi
      \expandafter\DTLformatauthor\@dtl@author
    \else
      \DTLandnotlast \expandafter\DTLformatauthor\@dtl@author
    \fi
  \fi
  }%
\fi
}{}%
}
```

The counter DTLmaxeditors indicates the maximum number of editor names to display, if there are more than that number, \etalname is used.

```
\newcounter{DTLmaxeditors}
\setcounter{DTLmaxeditors}{10}
```

\DTLformateditorlist Format a list of editor names (the list is stored in \@dtl@key@Editor):

```
\newcommand*{\DTLformateditorlist}{%
\DTLifbibfieldexists{Editor}{%
\DTLstartsentencespace
\@dtl@authorcount=0\relax
\@for\@dtl@author:=\@dtl@key@Editor\do{%
\advance\@dtl@authorcount by 1\relax}%
\@dtl@tmpcount=0\relax
\ifnum\@dtl@authorcount>\c@DTLmaxeditors
{%
  \@for\@dtl@author:=\@dtl@key@Editor\do{%
  \advance\@dtl@tmpcount by 1\relax
  \ifnum\@dtl@tmpcount=1\relax
   \expandafter\DTLformateditor\@dtl@author
  \else
    \ifnum\@dtl@tmpcount>\c@DTLmaxeditors
      \DTLandnotlast \etalname
      \expandafter\DTLcheckendsperiod\expandafter{\etalname}%
      \@endfortrue
    \else
      \DTLandnotlast \expandafter\DTLformateditor\@dtl@author
    \fi
  \fi
  }%
}%
\else
\@for\@dtl@author:=\@dtl@key@Editor\do{%
  \advance\@dtl@tmpcount by 1\relax
  \ifnum\@dtl@tmpcount=1\relax
   \expandafter\DTLformateditor\@dtl@author
  \else
    \ifnum\@dtl@tmpcount=\@dtl@authorcount
      \ifnum\@dtl@authorcount=2\relax
        \DTLtwoand
      \else
        \DTLandlast
      \fi
      \expandafter\DTLformateditor\@dtl@author
    \else
      \DTLandnotlast \expandafter\DTLformateditor\@dtl@author
    \fi
  \fi
  }%
\fi
,
\ifnum\@dtl@authorcount=1\relax
  \editorname
  \expandafter\DTLcheckendsperiod\expandafter{\editorname}%
\else
  \editorsname
  \expandafter\DTLcheckendsperiod\expandafter{\editorsname}%
```

```
\fi
}{}%
}
```

\DTLformatsurnameonly

\DTLformatsurnameonly{⟨*von part*⟩}{⟨*surname*⟩}{⟨*jr part*⟩}{⟨*forenames*⟩}

This is used when only the surname should be displayed. (The final argument, ⟨*forenames*⟩, is ignored.)

```
\newcommand*{\DTLformatsurnameonly}[4]{%
\DTLstartsentencespace
\def\@dtl@tmp{#1}%
\ifx\@dtl@tmp\@empty\else#1~\fi
#2%
\def\@dtl@tmp{#3}%
\ifx\@dtl@tmp\@empty
  \DTLcheckendsperiod{#2}%
\else
  , #3%
  \DTLcheckendsperiod{#3}%
\fi
}
```

\DTLformatforenames

\DTLformatforenames{⟨*forenames*⟩}

The format of an author/editor's forenames. If the forenames occur at the start of sentence, a new sentence space is added. The argument is checked to determine whether it ends with a full stop (sometimes the forenames may include initials.)

```
\newcommand*{\DTLformatforenames}[1]{%
\DTLstartsentencespace
#1%
\DTLcheckendsperiod{#1}}
```

\DTLformatabbrvforenames

\DTLformatabbrvforenames{⟨*forenames*⟩}

The format of an author/editor's abbreviated forenames. The initials may or may not end in a full stop depending on the commands governing the format of \DTLstoreinitials, so the initials need to be check using \DTLcheckendsperiod.

```
\newcommand*{\DTLformatabbrvforenames}[1]{%
\DTLstartsentencespace
\DTLstoreinitials{#1}{\@dtl@tmp}\@dtl@tmp
\expandafter\DTLcheckendsperiod\expandafter{\@dtl@tmp}}
```

\DTLformatvon

\DTLformatvon{⟨*von part*⟩}

The format of the "von" part. This does nothing if the argument is empty, otherwise it does the argument followed by a non-breakable space.

```
\newcommand*{\DTLformatvon}[1]{%
\DTLstartsentencespace
\def\@dtl@tmp{#1}%
\ifx\@dtl@tmp\@empty
\else
  #1~%
\fi
}
```

\DTLformatsurname

\DTLformatsurname{⟨*surname*⟩}

The format of an author/editor's surname.

```
\newcommand*{\DTLformatsurname}[1]{%
\DTLstartsentencespace
#1\DTLcheckendsperiod{#1}}
```

\DTLformatjr

\DTLformatjr{⟨*jr part*⟩}

The format of the "jr" part. This does nothing if the argument is empty.

```
\newcommand*{\DTLformatjr}[1]{%
\DTLstartsentencespace
\def\@dtl@tmp{#1}%
\ifx\@dtl@tmp\@empty
\else
  , #1\DTLcheckendsperiod{#1}%
\fi
}
```

\DTLformatcrossrefeditor

Format cross reference editors:

```
\newcommand*{\DTLformatcrossrefeditor}{%
\DTLifbibfieldexists{Editor}{%
\DTLstartsentencespace
\@dtl@authorcount=0\relax
\@for\@dtl@author:=\@dtl@key@Editor\do{%
\advance\@dtl@authorcount by 1\relax}%
{\@dtl@tmpcount=0\relax
\@for\@dtl@author:=\@dtl@key@Editor\do{%
\ifnum\@dtl@authorcount=1\relax
 \expandafter\DTLformatsurnameonly\@dtl@author
\else
 \advance\@dtl@tmpcount by 1\relax
 \ifnum\@dtl@tmpcount=1\relax
    \expandafter\DTLformatsurnameonly\@dtl@author
  \else
    \ifnum\@dtl@authorcount=2\relax
      \ \andname\ \expandafter\DTLformatsurnameonly\@dtl@author
    \else
      \ \etalname
      \expandafter\DTLcheckendsperiod\expandafter{\etalname}
    \fi
```

```
    \@endfortrue
  \fi
\fi
}}%
}{}%
}
```

\DTLformatvolnumpages Format volume, number and pages (of an article).

```
\newcommand*{\DTLformatvolnumpages}{%
\DTLifbibfieldexists{Volume}{%
\DTLstartsentencespace
\DTLbibfield{Volume}\DTLperiodfalse}{}%
\DTLifbibfieldexists{Number}{%
\DTLstartsentencespace
(\DTLbibfield{Number})\DTLperiodfalse}{}%
\DTLifbibfieldexists{Pages}{%
\DTLifanybibfieldexists{Volume,Number}{:}{%
\DTLstartsentencespace
\protected@edef\@dtl@pages{0\DTLbibfield{Pages}}%
\DTLifnumerical{\@dtl@pages}{\pagename}{\pagesname}~}%
\DTLbibfield{Pages}\DTLperiodfalse}{}%
}
```

\DTLformatbvolume Format book volume.

```
\newcommand*{\DTLformatbvolume}{%
\DTLifbibfieldexists{Volume}{%
\ifDTLmidsentence
  \volumename
\else
  \DTLstartsentencespace
  \expandafter\MakeUppercase\volumename
\fi
~\DTLbibfield{Volume}%
\DTLifbibfieldexists{Series}{\ \ofname\
{\em\DTLbibfield{Series}}\DTLcheckbibfieldendsperiod{Series}}{%
\DTLcheckbibfieldendsperiod{Volume}}%
}{}}
```

\DTLformatchapterpages Format chapter and pages:

```
\newcommand*{\DTLformatchapterpages}{%
\DTLifbibfieldexists{Chapter}{%
\DTLifbibfieldexists{Type}{%
\DTLstartsentencespace
\DTLbibfield{Type}}{%
\DTLstartsentencespace
\chaptername}~\DTLbibfield{Chapter}%
\DTLifbibfieldexists{Pages}{\DTLaddcomma}{%
\DTLcheckbibfieldendsperiod{Chapter}}}{}%
\DTLstartsentencespace
\DTLformatpages}
```

\DTLformatpages Format pages:

```
\newcommand*{\DTLformatpages}{%
\DTLifbibfieldexists{Pages}{%
```

```
\DTLstartsentencespace
\protected@edef\@dtl@pages{0\DTLbibfield{Pages}}%
\DTLifnumerical{\@dtl@pages}{\pagename}{\pagesname}~%
\DTLbibfield{Pages}\DTLcheckbibfieldendsperiod{Pages}}{}%
}
```

`\DTLformatnumberseries` Format number and series (of book)

```
\newcommand*{\DTLformatnumberseries}{%
\DTLifbibfieldexists{Volume}{}{%
\DTLifbibfieldexists{Number}{%
\ifDTLmidsentence
  \numbername
\else
  \DTLstartsentencespace
  \expandafter\MakeUppercase\numbername
\fi~\DTLbibfield{Number}%
\DTLifbibfieldexists{Series}{\ \inname\ \DTLbibfield{Series}%
\DTLcheckbibfieldendsperiod{Series}}{%
\DTLcheckbibfieldendsperiod{Number}}%
}{%
\DTLifbibfieldexists{Series}{%
\DTLstartsentencespace
\DTLbibfield{Series}%
\DTLcheckbibfieldendsperiod{Series}}{}}%
}%
}
```

`\DTLformatbookcrossref` Format a book cross reference.

```
\newcommand*{\DTLformatbookcrossref}{%
\DTLifbibfieldexists{Volume}{%
\ifDTLmidsentence
  \volumename
\else
  \DTLstartsentencespace
  \expandafter\MakeUppercase\volumename
\fi
~\DTLbibfield{Volume}\ \ofname\
}{%
\ifDTLmidsentence
  \inname
\else
  \DTLstartsentencespace
  \expandafter\MakeUppercase\inname
\fi\ }%
\DTLifbibfieldexists{Editor}{\DTLformatcrossrefeditor}{%
\DTLifbibfieldexists{Key}{%
\DTLbibfield{Key}}{%
\DTLifbibfieldexists{Series}{%
{\em\DTLbibfield{Series}}}{}%
}%
}%
\edef\@dtl@tmp{\DTLbibfield{CrossRef}}%
~\cite{\@dtl@tmp}%
}
```

`\DTLformatincollproccrossref` Format ‘incollections’ cross reference.

```
\newcommand*{\DTLformatincollproccrossref}{%
\DTLifbibfieldexists{Editor}{%
\ifDTLmidsentence
  \inname
\else
  \DTLstartsentencespace
  \expandafter\MakeUppercase\inname
\fi\
\DTLformatcrossrefeditor
}{%
\DTLifbibfieldexists{Key}{%
\ifDTLmidsentence
  \inname
\else
  \DTLstartsentencespace
  \expandafter\MakeUppercase\inname
\fi\ \DTLbibfield{Key}%
}{%
\DTLifbibfieldexists{BookTitle}{%
\ifDTLmidsentence
  \inname
\else
  \DTLstartsentencespace
  \expandafter\MakeUppercase\inname
\fi\ {\em\DTLbibfield{BookTitle}}}{}%
}}%
\edef\@dtl@tmp{\DTLbibfield{CrossRef}}%
~\cite{\@dtl@tmp}%
}
```

`\DTLformatinedbooktitle` Format editor and booktitle:

```
\newcommand*{\DTLformatinedbooktitle}{%
\DTLifbibfieldexists{BookTitle}{%
\ifDTLmidsentence
  \inname
\else
  \DTLstartsentencespace
  \expandafter\MakeUppercase\inname
\fi\
\DTLifbibfieldexists{Editor}{%
\DTLformateditorlist\DTLaddcomma {\em\DTLbibfield{BookTitle}}%
\DTLcheckbibfieldendsperiod{BookTitle}%
}{{\em\DTLbibfield{BookTitle}}%
\DTLcheckbibfieldendsperiod{BookTitle}%
}}{}}
```

`\DTLformatdate` Format date.

```
\newcommand*{\DTLformatdate}{%
\DTLifbibfieldexists{Year}{%
\DTLifbibfieldexists{Month}{%
\protected@edef\@dtl@tmp{\DTLbibfield{Month}}%
\ifDTLmidsentence
  \@dtl@tmp
```

```
\else
  \DTLstartsentencespace
  \expandafter\MakeUppercase\@dtl@tmp
\fi\
\DTLmidsentencefalse}{}%
\DTLstartsentencespace
\DTLbibfield{Year}}{%
\DTLifbibfieldexists{Month}{%
\protected@edef\@dtl@tmp{\DTLbibfield{Month}}%
\ifDTLmidsentence
  \@dtl@tmp
\else
  \DTLstartsentencespace
  \expandafter\MakeUppercase\@dtl@tmp
\fi
\DTLcheckbibfieldendsperiod{Month}%
}{}}}
```

`\DTLformatarticlecrossref` Format article cross reference.

```
\newcommand*{\DTLformatarticlecrossref}{%
\DTLifbibfieldexists{Key}{%
\ifDTLmidsentence
 \inname
\else
 \DTLstartsentencespace
 \expandafter\MakeUppercase\inname
\fi
\ {\em\DTLbibfield{Key}}}{%
\DTLifbibfieldexists{Journal}{%
\ifDTLmidsentence
 \inname
\else
 \DTLstartsentencespace
 \expandafter\MakeUppercase\inname
\fi
\ {\em\DTLbibfield{Journal}}}{}}%
\edef\@dtl@tmp{\DTLbibfield{CrossRef}}%
~\cite{\@dtl@tmp}%
}
```

14.5.1 ifthen conditionals

The conditionals defined in this section may be used in the optional argument of `\DTLforeachbib`. They may also be used in the first argument of `\ifthenelse`, but only if the command occurs within the body of `\DTLforeachbib`.

`\DTLbibfieldexists` `\DTLbibfieldexists{⟨field label⟩}`

Checks if named bib field exists for current entry

```
\newcommand*{\DTLbibfieldexists}[1]{%
\TE@throw\noexpand\dtl@testbibfieldexists{#1}%
\noexpand\if@dtl@condition}
```

\dtl@testbibfieldexists

```
\newcommand*{\dtl@testbibfieldexists}[1]{%
\DTLifbibfieldexists{#1}{\@dtl@conditiontrue}{\@dtl@conditionfalse}}
```

\DTLbibfieldiseq

\DTLbibfieldiseq{⟨*field label*⟩}{⟨*value*⟩}

Checks if the value of the bib field given by ⟨*field label*⟩ is equal to ⟨*value*⟩. (Uses \dtlcompare to determine if the values are equal. If the bib field doesn't exist, the condition is false.)

```
\newcommand*{\DTLbibfieldiseq}[2]{%
\TE@throw\noexpand\dtl@testbibfieldiseq{#1}{#2}%
\noexpand\if@dtl@condition}
```

\dtl@testbibfieldiseq

```
\newcommand*{\dtl@testbibfieldiseq}[2]{%
\DTLifbibfieldexists{#1}{%
\expandafter\let\expandafter\@dtl@tmp\expandafter
 =\csname @dtl@key@#1\endcsname
\expandafter\toks@\expandafter{\@dtl@tmp}%
\@dtl@toks{#2}%
\edef\@dtl@docompare{\noexpand\dtlcompare{\noexpand\@dtl@tmpcount}%
{\the\toks@}{\the\@dtl@toks}}%
\@dtl@docompare
\ifnum\@dtl@tmpcount=0\relax
 \@dtl@conditiontrue
\else
 \@dtl@conditionfalse
\fi
}{%
\@dtl@conditionfalse}%
}
```

\DTLbibfieldislt

\DTLbibfieldislt{⟨*field label*⟩}{⟨*value*⟩}

Checks if the value of the bib field given by ⟨*field label*⟩ is less than ⟨*value*⟩. (If the bib field doesn't exist, the condition is false.)

```
\newcommand*{\DTLbibfieldislt}[2]{%
\TE@throw\noexpand\dtl@testbibfieldislt{#1}{#2}%
\noexpand\if@dtl@condition}
```

\dtl@testbibfieldislt

```
\newcommand*{\dtl@testbibfieldislt}[2]{%
\DTLifbibfieldexists{#1}{%
\expandafter\let\expandafter\@dtl@tmp\expandafter
 =\csname @dtl@key@#1\endcsname
\expandafter\toks@\expandafter{\@dtl@tmp}%
\@dtl@toks{#2}%
\edef\@dtl@docompare{\noexpand\dtlcompare{\noexpand\@dtl@tmpcount}%
{\the\toks@}{\the\@dtl@toks}}%
\@dtl@docompare
```

```
\ifnum\@dtl@tmpcount=-1\relax
 \@dtl@conditiontrue
\else
 \@dtl@conditionfalse
\fi
}{%
\@dtl@conditionfalse}%
}
```

\DTLbibfieldisle

\DTLbibfieldisle{⟨*field label*⟩}{⟨*value*⟩}

Checks if the value of the bib field given by ⟨*field label*⟩ is less than or equal to ⟨*value*⟩. (If the bib field doesn't exist, the condition is false.)

```
\newcommand*{\DTLbibfieldisle}[2]{%
\TE@throw\noexpand\dtl@testbibfieldisle{#1}{#2}%
\noexpand\if@dtl@condition}
```

\dtl@testbibfieldisle

```
\newcommand*{\dtl@testbibfieldisle}[2]{%
\DTLifbibfieldexists{#1}{%
\expandafter\let\expandafter\@dtl@tmp\expandafter
 =\csname @dtl@key@#1\endcsname
\expandafter\toks@\expandafter{\@dtl@tmp}%
\@dtl@toks{#2}%
\edef\@dtl@docompare{\noexpand\dtlcompare{\noexpand\@dtl@tmpcount}%
{\the\toks@}{\the\@dtl@toks}}%
\@dtl@docompare
\ifnum\@dtl@tmpcount<1\relax
 \@dtl@conditiontrue
\else
 \@dtl@conditionfalse
\fi
}{%
\@dtl@conditionfalse}%
}
```

\DTLbibfieldisgt

\DTLbibfieldisgt{⟨*field label*⟩}{⟨*value*⟩}

Checks if the value of the bib field given by ⟨*field label*⟩ is greater than ⟨*value*⟩. (If the bib field doesn't exist, the condition is false.)

```
\newcommand*{\DTLbibfieldisgt}[2]{%
\TE@throw\noexpand\dtl@testbibfieldisgt{#1}{#2}%
\noexpand\if@dtl@condition}
```

\dtl@testbibfieldisgt

```
\newcommand*{\dtl@testbibfieldisgt}[2]{%
\DTLifbibfieldexists{#1}{%
\expandafter\let\expandafter\@dtl@tmp\expandafter
 =\csname @dtl@key@#1\endcsname
\expandafter\toks@\expandafter{\@dtl@tmp}%
```

```
\@dtl@toks{#2}%
\edef\@dtl@docompare{\noexpand\dtlcompare{\noexpand\@dtl@tmpcount}%
{\the\toks@}{\the\@dtl@toks}}%
\@dtl@docompare
\ifnum\@dtl@tmpcount=1\relax
 \@dtl@conditiontrue
\else
 \@dtl@conditionfalse
\fi
}{%
\@dtl@conditionfalse}%
}
```

`\DTLbibfieldisge`

`\DTLbibfieldisge{`⟨*field label*⟩`}{`⟨*value*⟩`}`

Checks if the value of the bib field given by ⟨*field label*⟩ is less than or equal to ⟨*value*⟩. (If the bib field doesn't exist, the condition is false.)

```
\newcommand*{\DTLbibfieldisge}[2]{%
\TE@throw\noexpand\dtl@testbibfieldisge{#1}{#2}%
\noexpand\if@dtl@condition}
```

`\dtl@testbibfieldisge`

```
\newcommand*{\dtl@testbibfieldisge}[2]{%
\DTLifbibfieldexists{#1}{%
\expandafter\let\expandafter\@dtl@tmp\expandafter
 =\csname @dtl@key@#1\endcsname
\expandafter\toks@\expandafter{\@dtl@tmp}%
\@dtl@toks{#2}%
\edef\@dtl@docompare{\noexpand\dtlcompare{\noexpand\@dtl@tmpcount}%
{\the\toks@}{\the\@dtl@toks}}%
\@dtl@docompare
\ifnum\@dtl@tmpcount>-1\relax
 \@dtl@conditiontrue
\else
 \@dtl@conditionfalse
\fi
}{%
\@dtl@conditionfalse}%
}
```

`\DTLbibfieldcontains`

`\DTLbibfieldcontains{`⟨*field label*⟩`}{`⟨*sub string*⟩`}`

Checks if the value of the bib field given by ⟨*field label*⟩ contains ⟨*sub string*⟩. (If the bib field doesn't exist, the condition is false.)

```
\newcommand*{\DTLbibfieldcontains}[2]{%
\TE@throw\noexpand\dtl@testbibfieldcontains{#1}{#2}%
\noexpand\if@dtl@condition}
```

`\dtl@testbibfieldcontains`

```
\newcommand*{\dtl@testbibfieldcontains}[2]{%
```

```
\DTLifbibfieldexists{#1}{%
\expandafter\let\expandafter\@dtl@tmp\expandafter
  =\csname @dtl@key@#1\endcsname
\expandafter\dtl@testifsubstring\expandafter{\@dtl@tmp}{#2}%
}{\@dtl@conditionfalse}}
```

14.6 Bibliography Style Macros

The macros defined in this section should be redefined by bibliography styles.

`DTLthebibliography` How to format the entire bibliography:

```
\newenvironment{DTLthebibliography}[2][\boolean{true}]{%
\@dtl@tmpcount=0\relax
\@sDTLforeach[#1]{#2}{}{\advance\@dtl@tmpcount by 1\relax}%
\begin{thebibliography}{\number\@dtl@tmpcount}
}{\end{thebibliography}}
```

`\DTLmonthname` The monthname style. The argument must be a number from 1 to 12. By default, uses `\dtl@monthname`.

```
\newcommand*{\DTLmonthname}[1]{%
\dtl@monthname{#1}}
```

`\dtl@monthname` Full month names:

```
\newcommand*{\dtl@monthname}[1]{%
\ifcase#1%
\or January%
\or February%
\or March%
\or April%
\or May%
\or June%
\or July%
\or August%
\or September%
\or October%
\or November%
\or December%
\fi}
```

`\dtl@abbrvmonthname` Abbreviated months:

```
\newcommand*{\dtl@abbrvmonthname}[1]{%
\ifcase#1%
\or Jan.%
\or Feb.%
\or Mar.%
\or Apr.%
\or May%
\or June%
\or July%
\or Aug.%
\or Sept.%
\or Oct.%
\or Nov.%
```

```
\or Dec.%
\fi}
```

\DTLbibitem Define how to start a new bibitem:

```
\newcommand*{\DTLbibitem}{\bibitem{\DBIBcitekey}}
```

\DTLmbibitem As \DTLbibitem but for \DTLmbibliography

```
\newcommand*{\DTLmbibitem}[1]{\bibitem{#1@\DBIBcitekey}}
```

\DTLformatauthor

\DTLformatauthor{⟨*von part*⟩}{⟨*surname*⟩}{⟨*junior part*⟩}{⟨*forenames*⟩}

The format of an author's name.

```
\newcommand*{\DTLformatauthor}[4]{%
\DTLformatforenames{#4}
\DTLformatvon{#1}%
\DTLformatsurname{#2}%
\DTLformatjr{#3}}
```

\DTLformateditor The format of an editor's name.

```
\newcommand*{\DTLformateditor}[4]{%
\DTLformatforenames{#4}
\DTLformatvon{#1}%
\DTLformatsurname{#2}%
\DTLformatjr{#3}}
```

\DTLformatedition The format of an edition:

```
\newcommand*{\DTLformatedition}[1]{#1 \editionname}
```

\DTLformatarticle The format of an article:

```
\newcommand{\DTLformatarticle}{}
```

\DTLformatbook The format of a book:

```
\newcommand{\DTLformatbook}{}
```

\DTLformatbooklet The format of a booklet:

```
\newcommand{\DTLformatbooklet}{}
```

\DTLformatinbook The format of an "inbook" type:

```
\newcommand{\DTLformatinbook}{}
```

\DTLformatincollection The format of an "incollection" type:

```
\newcommand{\DTLformatincollection}{}
```

\DTLformatinproceedings The format of an "inproceedings" type:

```
\newcommand{\DTLformatinproceedings}{}
```

\DTLformatmanual The format of a manual:

```
\newcommand{\DTLformatmanual}{}
```

\DTLformatmastersthesis The format of a master's thesis:

```
\newcommand{\DTLformatmastersthesis}{}
```

`\DTLformatmisc` The format of a miscellaneous entry:

```
\newcommand{\DTLformatmisc}{}
```

`\DTLformatphdthesis` The format of a Ph.D. thesis:

```
\newcommand{\DTLformatphdthesis}{}
```

`\DTLformatproceedings` The format of a proceedings:

```
\newcommand{\DTLformatproceedings}{}
```

`\DTLformattechreport` The format of a technical report:

```
\newcommand{\DTLformattechreport}{}
```

`\DTLformatunpublished` The format of an unpublished work:

```
\newcommand{\DTLformatunpublished}{}
```

Predefined names (these correspond to the standard BibTeX predefined strings of the same name without the leading `\DTL`):

`\DTLacmcs`

```
\newcommand*{\DTLacmcs}{ACM Computing Surveys}
```

`\DTLacta`

```
\newcommand*{\DTLacta}{Acta Informatica}
```

`\DTLcacm`

```
\newcommand*{\DTLcacm}{Communications of the ACM}
```

`\DTLibmjrd`

```
\newcommand*{\DTLibmjrd}{IBM Journal of Research and Development}
```

`\DTLibmsj`

```
\newcommand*{\DTLibmsj}{IBM Systems Journal}
```

`\DTLieeese`

```
\newcommand*{\DTLieeese}{IEEE Transactions on Software Engineering}
```

`\DTLieeetc`

```
\newcommand*{\DTLieeetc}{IEEE Transactions on Computers}
```

`\DTLieeetcad`

```
\newcommand*{\DTLieeetcad}{IEEE Transactions on Computer-Aided Design
of Integrated Circuits}
```

`\DTLipl`

```
\newcommand*{\DTLipl}{Information Processing Letters}
```

`\DTLjacm`

```
\newcommand*{\DTLjacm}{Journal of the ACM}
```

`\DTLjcss`

```
\newcommand*{\DTLjcss}{Journal of Computer and System Sciences}
```

\DTLscp

```
\newcommand*{\DTLscp}{Science of Computer Programming}
```

\DTLsicomp

```
\newcommand*{\DTLsicomp}{SIAM Journal on Computing}
```

\DTLtocs

```
\newcommand*{\DTLtocs}{ACM Transactions on Computer Systems}
```

\DTLtods

```
\newcommand*{\DTLtods}{ACM Transactions on Database Systems}
```

\DTLtog

```
\newcommand*{\DTLtog}{ACM Transactions on Graphics}
```

\DTLtoms

```
\newcommand*{\DTLtoms}{ACM Transactions on Mathematical Software}
```

\DTLtoois

```
\newcommand*{\DTLtoois}{ACM Transactions on Office Information
Systems}
```

\DTLtoplas

```
\newcommand*{\DTLtoplas}{ACM Transactions on Programming Languages
and Systems}
```

\DTLtcs

```
\newcommand*{\DTLtcs}{Theoretical Computer Science}
```

14.7 Bibliography Styles

Each bibliography style is set by the command `\dtlbst@`⟨*style*⟩, where ⟨*style*⟩ is the name of the bibliography style.

\dtlbst@plain The 'plain' style:

```
\newcommand{\dtlbst@plain}{%
```

Set how to format the entire bibliography:

```
\renewenvironment{DTLthebibliography}[2][\boolean{true}]{%
\@dtl@tmpcount=0\relax
\@sDTLforeach[##1]{##2}{}{\advance\@dtl@tmpcount by 1\relax}%
\begin{thebibliography}{\number\@dtl@tmpcount}%
}{\end{thebibliography}}%
```

Set how to start the bibliography entry:

```
\renewcommand*{\DTLbibitem}{\bibitem{\DBIBcitekey}}%
\renewcommand*{\DTLmbibitem}[1]{\bibitem{##1@\DBIBcitekey}}%
```

Sets the author name format.

```
\renewcommand*{\DTLformatauthor}[4]{%
\DTLformatforenames{##4}
\DTLformatvon{##1}%
\DTLformatsurname{##2}%
\DTLformatjr{##3}}
```

Sets the editor name format.

```
\renewcommand*{\DTLformateditor}[4]{%
\DTLformatforenames{##4}
\DTLformatvon{##1}%
\DTLformatsurname{##2}%
\DTLformatjr{##3}}
```

Sets the edition format.

```
\renewcommand*{\DTLformatedition}[1]{##1 \editionname}%
```

Sets the monthname format.

```
\let\DTLmonthname\dtl@monthname
```

Sets other predefined names:

```
\renewcommand*{\DTLacmcs}{ACM Computing Surveys}
\renewcommand*{\DTLacta}{Acta Informatica}
\renewcommand*{\DTLcacm}{Communications of the ACM}
\renewcommand*{\DTLibmjrd}{IBM Journal of Research and Development}
\renewcommand*{\DTLibmsj}{IBM Systems Journal}
\renewcommand*{\DTLieeese}{IEEE Transactions on Software Engineering}
\renewcommand*{\DTLieeetc}{IEEE Transactions on Computers}
\renewcommand*{\DTLieeetcad}{IEEE Transactions on Computer-Aided Design
of Integrated Circuits}
\renewcommand*{\DTLipl}{Information Processing Letters}
\renewcommand*{\DTLjacm}{Journal of the ACM}
\renewcommand*{\DTLjcss}{Journal of Computer and System Sciences}
\renewcommand*{\DTLscp}{Science of Computer Programming}
\renewcommand*{\DTLsicomp}{SIAM Journal on Computing}
\renewcommand*{\DTLtocs}{ACM Transactions on Computer Systems}
\renewcommand*{\DTLtods}{ACM Transactions on Database Systems}
\renewcommand*{\DTLtog}{ACM Transactions on Graphics}
\renewcommand*{\DTLtoms}{ACM Transactions on Mathematical Software}
\renewcommand*{\DTLtoois}{ACM Transactions on Office Information
Systems}
\renewcommand*{\DTLtoplas}{ACM Transactions on Programming Languages
and Systems}
\renewcommand*{\DTLtcs}{Theoretical Computer Science}
```

The format of an article.

```
\renewcommand*{\DTLformatarticle}{%
\DTLformatauthorlist
\DTLifbibfieldexists{Author}{\DTLaddperiod}{}%
\DTLifbibfieldexists{Title}{%
\DTLstartsentencespace\DTLbibfield{Title}%
\DTLcheckbibfieldendsperiod{Title}%
\DTLaddperiod}{}%
\DTLifbibfieldexists{CrossRef}{%
% cross ref field
\DTLformatarticlecrossref
\DTLifbibfieldexists{Pages}{\DTLaddcomma}{}%
\DTLformatpages
\DTLaddperiod
}{% no cross ref field
\DTLifbibfieldexists{Journal}{\DTLstartsentencespace
{\em\DTLbibfield{Journal}}%
```

```
\DTLcheckbibfieldendsperiod{Journal}%
\DTLifanybibfieldexists{Number,Volume,Pages,Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLformatvolnumpages
\DTLifanybibfieldexists{Volume,Number,Pages}{%
\DTLifanybibfieldexists{Year,Month}{\DTLaddcomma}{%
\DTLaddperiod}%
\DTLmidsentencefalse}{}%
\DTLformatdate
\DTLifanybibfieldexists{Year,Month}{\DTLaddperiod}{}%
}%
\DTLifbibfieldexists{Note}{\DTLstartsentencespace\DTLbibfield{Note}%
\DTLcheckbibfieldendsperiod{Note}%
\DTLaddperiod}{}%
}
```

The format of a book.

```
\renewcommand*{\DTLformatbook}{%
\DTLifbibfieldexists{Author}{%
\DTLformatauthorlist\DTLaddperiod
}{\DTLformateditorlist\DTLifbibfieldexists{Editor}{%
\DTLaddperiod}{}}%
\DTLifbibfieldexists{Title}{\DTLstartsentencespace
{\em\DTLbibfield{Title}}%
\DTLcheckbibfieldendsperiod{Title}}{}%
\DTLifbibfieldexists{CrossRef}{%
% cross ref field
\DTLifbibfieldexists{Title}{\DTLaddperiod}{}%
\DTLformatbookcrossref
\DTLifanybibfieldexists{Edition,Month,Year}{\DTLaddcomma
}{\DTLaddperiod}%
}{% no cross ref field
\DTLifbibfieldexists{Title}{%
\DTLifbibfieldexists{Volume}{\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLformatbvolume
\DTLformatnumberseries
\DTLifanybibfieldexists{Number,Series,Volume}{\DTLaddperiod}{}%
\DTLifbibfieldexists{Publisher}{\DTLstartsentencespace
\DTLbibfield{Publisher}%
\DTLcheckbibfieldendsperiod{Publisher}%
\DTLifbibfieldexists{Address}{\DTLaddcomma}{%
\DTLifanybibfieldexists{Month,Year}{\DTLaddcomma
}{\DTLaddperiod}%
}}{}%
\DTLifbibfieldexists{Address}{\DTLstartsentencespace
\DTLbibfield{Address}%
\DTLcheckbibfieldendsperiod{Address}%
\DTLifanybibfieldexists{Month,Year}{\DTLaddcomma}{\DTLaddperiod}}{}%
}%
\DTLifbibfieldexists{Edition}{%
\protected@edef\@dtl@tmp{\DTLformatedition{\DTLbibfield{Edition}}}%
\ifDTLmidsentence
 \@dtl@tmp
\else
 \DTLstartsentencespace\expandafter\MakeUppercase\@dtl@tmp
```

```
\fi
\expandafter\DTLcheckendsperiod\expandafter{\@dtl@tmp}%
\DTLifanybibfieldexists{Month,Year}{\DTLaddcomma}{\DTLaddperiod}%
}{}%
\DTLformatdate
\DTLifanybibfieldexists{Year,Month}{\DTLaddperiod}{}%
\DTLifbibfieldexists{Note}{\DTLstartsentencespace
\DTLbibfield{Note}%
\DTLcheckbibfieldendsperiod{Note}%
\DTLaddperiod}{}%
}%
```

The format of a booklet.

```
\renewcommand*{\DTLformatbooklet}{%
\DTLifbibfieldexists{Author}{%
\DTLformatauthorlist\DTLaddperiod}{}%
\DTLifbibfieldexists{Title}{\DTLstartsentencespace
\DTLbibfield{Title}%
\DTLcheckbibfieldendsperiod{Title}%
\DTLaddperiod}{}%
\DTLifbibfieldexists{HowPublished}{%
\DTLstartsentencespace\DTLbibfield{HowPublished}%
\DTLcheckbibfieldendsperiod{HowPublished}%
\DTLifanybibfieldexists{Address,Month,Year}{\DTLaddcomma
}{\DTLaddperiod}}{}%
\DTLifbibfieldexists{Address}{\DTLstartsentencespace
\DTLbibfield{Address}%
\DTLcheckbibfieldendsperiod{Address}%
\DTLifanybibfieldexists{Month,Year}{\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLformatdate
\DTLifanybibfieldexists{Year,Month}{\DTLaddperiod}{}%
\DTLifbibfieldexists{Note}{\DTLstartsentencespace\DTLbibfield{Note}%
\DTLcheckbibfieldendsperiod{Note}%
\DTLaddperiod}{}%
}%
```

The format of an ‘inbook’ entry.

```
\renewcommand*{\DTLformatinbook}{%
\DTLifbibfieldexists{Author}{%
\DTLformatauthorlist\DTLaddperiod}{%
\DTLifbibfieldexists{Editor}{\DTLformateditorlist\DTLaddperiod}{}}%
\DTLifbibfieldexists{Title}{%
\DTLstartsentencespace
{\em\DTLbibfield{Title}}%
\DTLcheckbibfieldendsperiod{Title}%
}{}%
\DTLifbibfieldexists{CrossRef}{%
% Cross ref entry
\DTLifbibfieldexists{Title}{%
\DTLifbibfieldexists{Chapter}{\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLformatchapterpages
\DTLifanybibfieldexists{Chapter,Pages}{\DTLaddperiod}{}%
\DTLformatbookcrossref
}{% no cross ref
\DTLifbibfieldexists{Title}{%
```

```
\DTLifanybibfieldexists{Chapter,Volume}{\DTLaddcomma
}{\DTLaddperiod}}{}%
\DTLformatbvolume
\DTLifanybibfieldexists{Volume,Series}{%
\DTLifanybibfieldexists{Chapter,Pages}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLformatchapterpages
\DTLifanybibfieldexists{Chapter,Pages}{\DTLaddperiod}{}%
\DTLifbibfieldexists{Publisher}{%
\DTLstartsentencespace
\DTLbibfield{Publisher}%
\DTLcheckbibfieldendsperiod{Publisher}%
\DTLifbibfieldexists{Address}{\DTLaddcomma}{}}{}%
\DTLifbibfieldexists{Address}{%
\DTLstartsentencespace
\DTLbibfield{Address}%
\DTLcheckbibfieldendsperiod{Address}}{}%
}%
\DTLifanybibfieldexists{Edition,Month,Year}{\DTLaddcomma
}{\DTLaddperiod}%
\DTLifbibfieldexists{Edition}{%
\protected@edef\@dtl@tmp{\DTLformatedition{\DTLbibfield{Edition}}}%
\ifDTLmidsentence
 \@dtl@tmp
\else
 \DTLstartsentencespace
 \expandafter\MakeUppercase\@dtl@tmp
\fi
\expandafter\DTLcheckendsperiod\expandafter{\@dtl@tmp}%
\DTLifanybibfieldexists{Month,Year}{\DTLaddcomma
}{\DTLaddperiod}%
}{}%
\DTLformatdate
\DTLifanybibfieldexists{Month,Year}{\DTLaddperiod}{}%
\DTLifbibfieldexists{Note}{%
\DTLstartsentencespace
\DTLbibfield{Note}%
\DTLcheckbibfieldendsperiod{Note}%
\DTLaddperiod}{}%
}%
```

The format of an ‘incollection’ entry.

```
\renewcommand*{\DTLformatincollection}{%
\DTLifbibfieldexists{Author}{\DTLformatauthorlist\DTLaddperiod}{}%
\DTLifbibfieldexists{Title}{%
\DTLstartsentencespace
\DTLbibfield{Title}%
\DTLcheckbibfieldendsperiod{Title}%
\DTLaddperiod}{}%
\DTLifbibfieldexists{CrossRef}{%
% cross ref entry
\DTLformatincollproccrossref
\DTLifanybibfieldexists{Chapter,Pages}{\DTLaddcomma}{}%
\DTLformatchapterpages\DTLaddperiod
}{% no cross ref entry
```

```
\DTLformatinedbooktitle
\DTLifbibfieldexists{BookTitle}{%
\DTLifanybibfieldexists{Volume,Series,Chapter,Pages,Number}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLformatbvolume
\DTLifbibfieldexists{Volume}{%
\DTLifanybibfieldexists{Number,Series,Chapter,Pages}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLformatnumberseries
\DTLifanybibfieldexists{Number,Series}{%
\DTLifanybibfieldexists{Chapter,Pages}{\DTLaddcomma
}{\DTLaddperiod}}{}%
\DTLformatchapterpages
\DTLifanybibfieldexists{Chapter,Pages}{\DTLaddperiod}{}%
\DTLifbibfieldexists{Publisher}{%
\DTLstartsentencespace
\DTLbibfield{Publisher}%
\DTLcheckbibfieldendsperiod{Publisher}%
\DTLifanybibfieldexists{Address,Edition,Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLifbibfieldexists{Address}{%
\DTLstartsentencespace
\DTLbibfield{Address}%
\DTLcheckbibfieldendsperiod{Address}%
\DTLifanybibfieldexists{Edition,Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLifbibfieldexists{Edition}{%
\protected@edef\@dtl@tmp{\DTLformatedition{\DTLbibfield{Edition}}}%
\ifDTLmidsentence
 \@dtl@tmp
\else
 \DTLstartsentencespace
 \expandafter\MakeUppercase\@dtl@tmp
\fi
\expandafter\DTLcheckendsperiod\expandafter{\@dtl@tmp}%
\DTLifanybibfieldexists{Month,Year}{\DTLaddcomma
}{\DTLaddperiod}%
}{}%
\DTLformatdate
\DTLifanybibfieldexists{Month,Year}{\DTLaddperiod}{}%
}%
\DTLifbibfieldexists{Note}{%
\DTLstartsentencespace
\DTLbibfield{Note}%
\DTLcheckbibfieldendsperiod{Note}%
\DTLaddperiod}{}%
}%
```

The format of an ‘inproceedings’ entry.

```
\renewcommand*{\DTLformatinproceedings}{%
\DTLifbibfieldexists{Author}{\DTLformatauthorlist
\DTLaddperiod}{}%
\DTLifbibfieldexists{Title}{%
\DTLstartsentencespace
\DTLbibfield{Title}%
```

```
\DTLcheckbibfieldendsperiod{Title}%
\DTLaddperiod}{}%
\DTLifbibfieldexists{CrossRef}{%
% cross ref entry
\DTLformatincollproccrossref
\DTLifbibfieldexists{Pages}{\DTLaddcomma}{%
\DTLaddperiod}%
\DTLformatpages
\DTLifbibfieldexists{Pages}{\DTLaddperiod}{}%
}{% no cross ref
\DTLformatinedbooktitle
\DTLifbibfieldexists{BookTitle}{%
\DTLifanybibfieldexists{Volume,Series,Pages,Number,Address,%
Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLformatbvolume
\DTLifbibfieldexists{Volume}{%
\DTLifanybibfieldexists{Number,Series,Pages,Address,Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLformatnumberseries
\DTLifanybibfieldexists{Number,Series}{%
\DTLifanybibfieldexists{Pages,Address,Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLformatpages
\DTLifbibfieldexists{Pages}{%
\DTLifanybibfieldexists{Address,Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLifbibfieldexists{Address}{%
\DTLstartsentencespace
\DTLbibfield{Address}%
\DTLcheckbibfieldendsperiod{Address}%
\DTLifanybibfieldexists{Month,Year}{\DTLaddcomma}{%
\DTLaddperiod}%
\DTLformatdate
\DTLifanybibfieldexists{Month,Year}{\DTLaddperiod}{}%
\DTLifbibfieldexists{Organization}{%
\DTLstartsentencespace
\DTLbibfield{Organization}%
\DTLcheckbibfieldendsperiod{Organization}%
\DTLifbibfieldexists{Publisher}{\DTLaddcomma}{%
\DTLaddperiod}}{}%
\DTLifbibfieldexists{Publisher}{%
\DTLstartsentencespace
\DTLbibfield{Publisher}%
\DTLcheckbibfieldendsperiod{Publisher}%
\DTLaddperiod}{}%
}{%
\DTLifanybibfieldexists{Publisher,Organization}{%
\DTLaddperiod}{}%
\DTLifbibfieldexists{Organization}{%
\DTLstartsentencespace
\DTLbibfield{Organization}%
\DTLcheckbibfieldendsperiod{Organization}%
\DTLifanybibfieldexists{Publisher,Month,Year}{%
```

```
\DTLaddcomma}{}}{}%
\DTLifbibfieldexists{Publisher}{%
\DTLstartsentencespace
\DTLbibfield{Publisher}%
\DTLcheckbibfieldendsperiod{Publisher}%
\DTLifanybibfieldexists{Month,Year}{\DTLaddcomma}{%
\DTLaddperiod}}{}%
\DTLformatdate
\DTLifanybibfieldexists{Month,Year}{\DTLaddperiod}{}%
}%
}%
\DTLifbibfieldexists{Note}{%
\DTLstartsentencespace
\DTLbibfield{Note}%
\DTLcheckbibfieldendsperiod{Note}%
\DTLaddperiod}{}%
}%
```

The format of a manual.

```
\renewcommand*{\DTLformatmanual}{%
\DTLifbibfieldexists{Author}{\DTLformatauthorlist
\DTLaddperiod}{%
\DTLifbibfieldexists{Organization}{%
\DTLstartsentencespace
\DTLbibfield{Organization}%
\DTLcheckbibfieldendsperiod{Organization}%
\DTLifbibfieldexists{Address}{\DTLaddcomma \DTLbibfield{Address}%
\DTLcheckbibfieldendsperiod{Address}%
}{}%
\DTLaddperiod}{}%
}%
\DTLifbibfieldexists{Title}{%
\DTLstartsentencespace
{\em\DTLbibfield{Title}}%
\DTLcheckbibfieldendsperiod{Title}%
\DTLifbibfieldexists{Author}{%
\DTLifanybibfieldexists{Organization,Address}{%
\DTLaddperiod}{\DTLaddcomma}}{%
\DTLifanybibfieldexists{Organization,Address,Edition,Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}}{}%
\DTLifbibfieldexists{Author}{%
\DTLifbibfieldexists{Organization}{%
\DTLstartsentencespace
\DTLbibfield{Organization}%
\DTLcheckbibfieldendsperiod{Organization}%
\DTLifanybibfieldexists{Address,Edition,Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLifbibfieldexists{Address}{%
\DTLstartsentencespace
\DTLbibfield{Address}%
\DTLcheckbibfieldendsperiod{Address}%
\DTLifanybibfieldexists{Edition,Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
}{%
\DTLifbibfieldexists{Organization}{}{%
```

```
\DTLifbibfieldexists{Address}{%
\DTLstartsentencespace
\DTLbibfield{Address}%
\DTLcheckbibfieldendsperiod{Address}%
\DTLifanybibfieldexists{Edition,Month,Year}{\DTLaddcomma}{%
\DTLaddperiod}}{}}%
}%
\DTLifbibfieldexists{Edition}{%
\protected@edef\@dtl@tmp{\DTLformatedition{\DTLbibfield{Edition}}}%
\ifDTLmidsentence
 \@dtl@tmp
\else
 \DTLstartsentencespace
 \expandafter\MakeUppercase\@dtl@tmp
\fi
\expandafter\DTLcheckendsperiod\expandafter{\@dtl@tmp}%
\DTLifanybibfieldexists{Month,Year}{\DTLaddcomma}{%
\DTLaddperiod}}{}%
\DTLformatdate
\DTLifanybibfieldexists{Month,Year}{\DTLaddperiod}{}%
\DTLifbibfieldexists{Note}{%
\DTLstartsentencespace
\DTLbibfield{Note}%
\DTLcheckbibfieldendsperiod{Note}%
\DTLaddperiod}{}%
}%
```

The format of a master's thesis.

```
\renewcommand*{\DTLformatmastersthesis}{%
\DTLifbibfieldexists{Author}{\DTLformatauthorlist\DTLaddperiod}{}%
\DTLifbibfieldexists{Title}{%
\DTLstartsentencespace
\DTLbibfield{Title}%
\DTLcheckbibfieldendsperiod{Title}%
\DTLaddperiod}{}%
\DTLifbibfieldexists{Type}{%
\DTLstartsentencespace
\DTLbibfield{Type}%
\DTLcheckbibfieldendsperiod{Type}%
\DTLifanybibfieldexists{School,Address,Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLifbibfieldexists{School}{%
\DTLstartsentencespace
\DTLbibfield{School}%
\DTLcheckbibfieldendsperiod{School}%
\DTLifanybibfieldexists{Address,Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLifbibfieldexists{Address}{%
\DTLstartsentencespace
\DTLbibfield{Address}%
\DTLcheckbibfieldendsperiod{Address}%
\DTLifanybibfieldexists{Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLformatdate
\DTLifanybibfieldexists{Month,Year}{\DTLaddperiod}{}%
```

```
\DTLifbibfieldexists{Note}{%
\DTLstartsentencespace
\DTLbibfield{Note}%
\DTLcheckbibfieldendsperiod{Note}%
\DTLaddperiod}{}%
}%
```

The format of a miscellaneous entry.

```
\renewcommand*{\DTLformatmisc}{%
\DTLifbibfieldexists{Author}{\DTLformatauthorlist\DTLaddperiod}{}%
\DTLifbibfieldexists{Title}{%
\DTLstartsentencespace
\DTLbibfield{Title}%
\DTLcheckbibfieldendsperiod{Title}%
\DTLifbibfieldexists{HowPublished}{\DTLaddperiod}{%
\DTLifanybibfieldexists{Month,Year}{\DTLaddcomma}{%
\DTLaddperiod}%
}%
\DTLmidsentencefalse}{}%
\DTLifbibfieldexists{HowPublished}{%
\DTLstartsentencespace
\DTLbibfield{HowPublished}%
\DTLcheckbibfieldendsperiod{HowPublished}%
\DTLifanybibfieldexists{Month,Year}{\DTLaddcomma}{%
\DTLaddperiod}}{}%
\DTLformatdate
\DTLifanybibfieldexists{Month,Year}{\DTLaddperiod}{}%
\DTLifbibfieldexists{Note}{%
\DTLstartsentencespace
\DTLbibfield{Note}%
\DTLcheckbibfieldendsperiod{Note}%
\DTLaddperiod}{}%
}%
```

The format of a PhD thesis.

```
\renewcommand*{\DTLformatphdthesis}{%
\DTLifbibfieldexists{Author}{\DTLformatauthorlist\DTLaddperiod}{}%
\DTLifbibfieldexists{Title}{%
\DTLstartsentencespace
{\em\DTLbibfield{Title}}%
\DTLcheckbibfieldendsperiod{Title}%
\DTLaddperiod}{}%
\DTLifbibfieldexists{Type}{%
\DTLstartsentencespace
\DTLbibfield{Type}%
\DTLcheckbibfieldendsperiod{Type}%
\DTLifanybibfieldexists{School,Address,Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLifbibfieldexists{School}{%
\DTLstartsentencespace
\DTLbibfield{School}%
\DTLcheckbibfieldendsperiod{School}%
\DTLifanybibfieldexists{Address,Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLifbibfieldexists{Address}{%
```

```
\DTLstartsentencespace
\DTLbibfield{Address}%
\DTLcheckbibfieldendsperiod{Address}%
\DTLifanybibfieldexists{Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLformatdate
\DTLifanybibfieldexists{Month,Year}{\DTLaddperiod}{}%
\DTLifbibfieldexists{Note}{%
\DTLstartsentencespace
\DTLbibfield{Note}%
\DTLcheckbibfieldendsperiod{Note}%
\DTLaddperiod}{}%
}%
```

The format of a proceedings.

```
\renewcommand*{\DTLformatproceedings}{%
\DTLifbibfieldexists{Editor}{%
\DTLformateditorlist\DTLaddperiod}{%
\DTLifbibfieldexists{Organization}{%
\DTLstartsentencespace
\DTLbibfield{Organization}%
\DTLcheckbibfieldendsperiod{Organization}%
\DTLaddperiod}{}}%
\DTLifbibfieldexists{Title}{%
\DTLstartsentencespace
{\em\DTLbibfield{Title}}%
\DTLcheckbibfieldendsperiod{Title}%
\DTLifanybibfieldexists{Volume,Number,Address,Editor,Publisher,%
Month,Year}{\DTLaddcomma}{\DTLaddperiod}%
}{}%
\DTLformatbvolume
\DTLifbibfieldexists{Volume}{%
\DTLifanybibfieldexists{Number,Address,Editor,Publisher,%
Month,Year}{\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLformatnumberseries
\DTLifbibfieldexists{Number}{%
\DTLifanybibfieldexists{Address,Editor,Publisher,%
Month,Year}{\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLifbibfieldexists{Address}{%
\DTLstartsentencespace
\DTLbibfield{Address}%
\DTLcheckbibfieldendsperiod{Address}%
\DTLifanybibfieldexists{Month,Year}{\DTLaddcomma}{\DTLaddperiod}%
\DTLformatdate
\DTLifanybibfieldexists{Month,Year}{\DTLaddperiod}{}%
\DTLifbibfieldexists{Editor}{\DTLifbibfieldexists{Organization}{%
\DTLstartsentencespace
\DTLbibfield{Organization}%
\DTLcheckbibfieldendsperiod{Organization}%
\DTLifbibfieldexists{Publisher}{%
\DTLaddcomma}{\DTLaddperiod}}{}}{}%
\DTLifbibfieldexists{Publisher}{%
\DTLstartsentencespace
\DTLbibfield{Publisher}%
\DTLcheckbibfieldendsperiod{Publisher}%
```

```
\DTLaddperiod
}{}%
}{% no address
\DTLifbibfieldexists{Editor}{%
\DTLifbibfieldexists{Organization}{%
\DTLstartsentencespace
\DTLbibfield{Organization}%
\DTLcheckbibfieldendsperiod{Organization}%
\DTLifanybibfieldexists{Publisher,Month,Year}{%
\DTLaddcomma}{\DTLaddperiod}}{}%
}{}%
\DTLifbibfieldexists{Publisher}{%
\DTLstartsentencespace
\DTLbibfield{Publisher}%
\DTLcheckbibfieldendsperiod{Publisher}%
\DTLifanybibfieldexists{Month,Year}{\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLformatdate
\DTLifanybibfieldexists{Month,Year}{\DTLaddperiod}{}%
}%
\DTLifbibfieldexists{Note}{%
\DTLstartsentencespace
\DTLbibfield{Note}%
\DTLcheckbibfieldendsperiod{Note}%
\DTLaddperiod}{}%
}%
```

The format of a technical report.

```
\renewcommand*{\DTLformattechreport}{%
\DTLifbibfieldexists{Author}{\DTLformatauthorlist\DTLaddperiod}{}%
\DTLifbibfieldexists{Title}{%
\DTLstartsentencespace
\DTLbibfield{Title}%
\DTLcheckbibfieldendsperiod{Title}%
\DTLaddperiod}{}%
\DTLifbibfieldexists{Type}{%
\DTLstartsentencespace
\DTLbibfield{Type}%
\DTLcheckbibfieldendsperiod{Type}%
\DTLifbibfieldexists{Number}{~}{}}{}%
\DTLifbibfieldexists{Number}{%
\DTLstartsentencespace
\DTLbibfield{Number}%
\DTLcheckbibfieldendsperiod{Number}%
}{}%
\DTLifanybibfieldexists{Type,Number}{%
\DTLifanybibfieldexists{Institution,Address,Month,Year}{\DTLaddcomma
}{\DTLaddperiod}}{}%
\DTLifbibfieldexists{Institution}{%
\DTLstartsentencespace
\DTLbibfield{Institution}%
\DTLcheckbibfieldendsperiod{Institution}%
\DTLifanybibfieldexists{Address,Month,Year}{\DTLaddcomma
}{\DTLaddperiod}}{}%
\DTLifbibfieldexists{Address}{%
\DTLstartsentencespace
```

```
\DTLbibfield{Address}%
\DTLcheckbibfieldendsperiod{Address}%
\DTLifanybibfieldexists{Month,Year}{\DTLaddcomma
}{\DTLaddperiod}}{}%
\DTLformatdate
\DTLifanybibfieldexists{Month,Year}{\DTLaddperiod}{}%
\DTLifbibfieldexists{Note}{%
\DTLstartsentencespace
\DTLbibfield{Note}%
\DTLcheckbibfieldendsperiod{Note}%
\DTLaddperiod}{}%
}%
```

The format of an unpublished work.

```
\renewcommand*{\DTLformatunpublished}{%
\DTLifbibfieldexists{Author}{\DTLformatauthorlist\DTLaddperiod}{}%
\DTLifbibfieldexists{Title}{%
\DTLstartsentencespace
\DTLbibfield{Title}%
\DTLcheckbibfieldendsperiod{Title}%
\DTLaddperiod}{}%
\DTLifbibfieldexists{Note}{%
\DTLstartsentencespace
\DTLbibfield{Note}%
\DTLcheckbibfieldendsperiod{Note}%
\DTLifanybibfieldexists{Month,Year}{\DTLaddcomma}{\DTLaddperiod}}{}%
\DTLformatdate
\DTLifanybibfieldexists{Month,Year}{\DTLaddperiod}{}%
}%
```

End of 'plain' style.

```
}
```

`\dtlbst@abbrv` Define 'abbrv' style. This is similar to 'plain' except that some of the values are abbreviated

```
\newcommand{\dtlbst@abbrv}{%
```

Base this style on 'plain':

```
\dtlbst@plain
```

Sets the author name format.

```
\renewcommand*{\DTLformatauthor}[4]{%
\DTLformatabbrvforenames{##4}
\DTLformatvon{##1}%
\DTLformatsurname{##2}%
\DTLformatjr{##3}}
```

Sets the editor name format.

```
\renewcommand*{\DTLformateditor}[4]{%
\DTLformatabbrvforenames{##4}
\DTLformatvon{##1}%
\DTLformatsurname{##2}%
\DTLformatjr{##3}}
```

Sets the monthname format.

```
\let\DTLmonthname\dtl@abbrvmonthname
```

Sets other predefined names:

```
\renewcommand*{\DTLacmcs}{ACM Comput.\ Surv.}
\renewcommand*{\DTLacta}{Acta Inf.}
\renewcommand*{\DTLcacm}{Commun.\ ACM}
\renewcommand*{\DTLibmjrd}{IBM J.\ Res.\ Dev.}
\renewcommand*{\DTLibmsj}{IBM Syst.~J.}
\renewcommand*{\DTLieeese}{IEEE Trans. Softw.\ Eng.}
\renewcommand*{\DTLieeetc}{IEEE Trans.\ Comput.}
\renewcommand*{\DTLieeetcad}{IEEE Trans.\ Comput.-Aided Design
Integrated Circuits}
\renewcommand*{\DTLipl}{Inf.\ Process.\ Lett.}
\renewcommand*{\DTLjacm}{J.~ACM}
\renewcommand*{\DTLjcss}{J.~Comput.\ Syst.\ Sci.}
\renewcommand*{\DTLscp}{Sci.\ Comput.\ Programming}
\renewcommand*{\DTLsicomp}{SIAM J.~Comput.}
\renewcommand*{\DTLtocs}{ACM Trans.\ Comput.\ Syst.}
\renewcommand*{\DTLtods}{ACM Trans.\ Database Syst.}
\renewcommand*{\DTLtog}{ACM Trans.\ Gr.}
\renewcommand*{\DTLtoms}{ACM Trans.\ Math. Softw.}
\renewcommand*{\DTLtoois}{ACM Trans. Office Inf.\ Syst.}
\renewcommand*{\DTLtoplas}{ACM Trans.\ Prog. Lang.\ Syst.}
\renewcommand*{\DTLtcs}{Theoretical Comput.\ Sci.}
```

End of 'abbrv' style.

```
}
```

`\dtlbst@alpha` Define 'alpha' style. This is similar to 'plain' except that the labels are strings rather than numerical.

```
\newcommand{\dtlbst@alpha}{%
```

Base this style on 'plain':

```
\dtlbst@plain
```

Set how to format the entire bibliography:

```
\renewenvironment{DTLthebibliography}[2][\boolean{true}]{%
\dtl@createalphabiblabels{##1}{##2}%
\begin{thebibliography}{\@dtl@widestlabel}%
}{\end{thebibliography}}%
```

Set how to start the bibliography entry:

```
\renewcommand*{\DTLbibitem}{%
\expandafter\bibitem\expandafter
 [\csname dtl@biblabel@\DBIBcitekey\endcsname]{\DBIBcitekey}}%
\renewcommand*{\DTLmbibitem}[1]{%
\expandafter\bibitem\expandafter
 [\csname dtl@biblabel@\DBIBcitekey\endcsname]{##1@\DBIBcitekey}}%
```

End of 'alpha' style.

```
}
```

`dtl@createalphabiblabels` `\dtl@createalphabiblabels{`⟨*condition*⟩`}{`⟨*db name*⟩`}`

Constructs the alpha style bib labels for the given database. (Labels are stored in the control sequence `\dtl@biblabel@`⟨*citekey*⟩.) This also sets `\@dtl@widestlabel` to the widest label.

```
\newcommand*{\dtl@createalphabiblabels}[2]{%
\dtl@message{Creating bib labels}%
\begingroup
\gdef\@dtl@widestlabel{}%
\dtl@widest=0pt\relax
\DTLforeachbibentry[#1]{#2}{%
\dtl@message{\DBIBcitekey}%
\DTLifbibfieldexists{Author}{%
  \dtl@listgetalphalabel{\@dtl@thislabel}{\@dtl@key@Author}%
}{%
\DTLifbibfieldexists{Editor}{%
    \dtl@listgetalphalabel{\@dtl@thislabel}{\@dtl@key@Editor}%
 }{%
   \DTLifbibfieldexists{Key}{%
     \expandafter\dtl@get@firstthree\expandafter
       {\@dtl@key@Key}{\@dtl@thislabel}%
   }{%
     \DTLifbibfieldexists{Organization}{%
       \expandafter\dtl@get@firstthree\expandafter
         {\@dtl@key@Organization}{\@dtl@thislabel}%
   }{%
       \expandafter\dtl@get@firstthree\expandafter
         {\DBIBentrytype}{\@dtl@thislabel}%
   }%
   }}}%
\DTLifbibfieldexists{Year}{}{\DTLifbibfieldexists{CrossRef}{%
\DTLgetvalueforkey{\@dtl@key@Year}{Year}{#2}{CiteKey}{%
\@dtl@key@CrossRef}}{}}%
\DTLifbibfieldexists{Year}{%
\expandafter\dtl@get@yearsuffix\expandafter{\@dtl@key@Year}%
\expandafter\toks@\expandafter{\@dtl@thislabel}%
\expandafter\@dtl@toks\expandafter{\@dtl@year}%
\edef\@dtl@thislabel{\the\toks@\the\@dtl@toks}%
}{}%
\let\@dtl@s@thislabel=\@dtl@thislabel
\@onelevel@sanitize\@dtl@s@thislabel
\@ifundefined{c@biblabel@\@dtl@s@thislabel}{%
\newcounter{biblabel@\@dtl@s@thislabel}%
\setcounter{biblabel@\@dtl@s@thislabel}{1}%
\expandafter\edef\csname @dtl@bibfirst@\@dtl@s@thislabel\endcsname{%
\DBIBcitekey}%
\expandafter\global
\expandafter\let\csname dtl@biblabel@\DBIBcitekey\endcsname=
  \@dtl@thislabel
}{%
\expandafter\ifnum\csname c@biblabel@\@dtl@s@thislabel\endcsname=1\relax
 \expandafter\let\expandafter\@dtl@tmp
   \csname @dtl@bibfirst@\@dtl@s@thislabel\endcsname
 \expandafter\protected@xdef\csname dtl@biblabel@\@dtl@tmp\endcsname{%
   \@dtl@thislabel a}%
\fi
```

```
\stepcounter{biblabel@\@dtl@s@thislabel}%
\expandafter\protected@xdef\csname dtl@biblabel@\DBIBcitekey\endcsname{%
   \@dtl@thislabel\alph{biblabel@\@dtl@s@thislabel}}%
}%
\settowidth{\dtl@tmplength}{%
 \csname dtl@biblabel@\DBIBcitekey\endcsname}%
\ifdim\dtl@tmplength>\dtl@widest
 \dtl@widest=\dtl@tmplength
 \expandafter\global\expandafter\let\expandafter\@dtl@widestlabel
  \expandafter=\csname dtl@biblabel@\DBIBcitekey\endcsname
\fi
}%
\endgroup
}
```

`\dtl@listgetalphalabel` Determine the alpha style label from a list of authors/editors (the first argument must be a control sequence (in which the label is stored), the second argument must be the list of names.)

```
\newcommand*{\dtl@listgetalphalabel}[2]{%
\@dtl@authorcount=0\relax
\@for\@dtl@author:=#2\do{%
\advance\@dtl@authorcount by 1\relax}%
\ifnum\@dtl@authorcount=1\relax
 \expandafter\dtl@getsinglealphalabel#2{#1}\relax
\else
  {%
  \xdef#1{}%
  \@dtl@tmpcount=0\relax
   \def\DTLafterinitials{}\def\DTLbetweeninitials{}%
   \def\DTLafterinitialbeforehyphen{}\def\DTLinitialhyphen{}%
   \@for\@dtl@author:=#2\do{%
     \expandafter\dtl@getauthorinitial\@dtl@author
     \expandafter\toks@\expandafter{\@dtl@tmp}%
     \expandafter\@dtl@toks\expandafter{#1}%
     \xdef#1{\the\@dtl@toks\the\toks@}%
     \advance\@dtl@tmpcount by 1\relax
     \ifnum\@dtl@tmpcount>2\relax\@endfortrue\fi
  }}%
\fi
}
```

Get author's initial (stores in `\@dtl@tmp`):

```
\newcommand*{\dtl@getauthorinitial}[4]{%
\def\@dtl@vonpart{#1}%
\ifx\@dtl@vonpart\@empty
 \DTLstoreinitials{#2}{\@dtl@tmp}%
\else
 \DTLstoreinitials{#1 #2}{\@dtl@tmp}%
\fi}
```

Get label for single author (last argument is control sequence in which to store the label):

```
\newcommand*{\dtl@getsinglealphalabel}[5]{%
\def\@dtl@vonpart{#1}%
```

```
\ifx\@dtl@vonpart\@empty
 \DTLifSubString{#2}{-}{%
   {\def\DTLafterinitials{}\def\DTLbetweeninitials{}%
    \def\DTLafterinitialbeforehyphen{}%
    \def\DTLinitialhyphen{}%
    \DTLstoreinitials{#2}{\@dtl@tmp}\global\let#5=\@dtl@tmp}%
   }{%
   \dtl@getfirstthree{#5}#2{}{}{}{}\@nil
 }
\else
 {\def\DTLafterinitials{}\def\DTLbetweeninitials{}%
  \def\DTLafterinitialbeforehyphen{}%
  \def\DTLinitialhyphen{}%
  \DTLstoreinitials{#1 #2}{\@dtl@tmp}\global\let#5=\@dtl@tmp}%
\fi
}
```

Get first three letters from the given string:

```
\def\dtl@getfirstthree#1#2#3#4#5\@nil{%
  \def#1{#2#3#4}%
}
\newcommand*{\dtl@get@firstthree}[2]{%
\dtl@getfirstthree#2#1{}{}{}{}{}\@nil}
```

Get year suffix:

```
\newcommand*{\dtl@get@yearsuffix}[1]{%
\dtl@getyearsuffix#1\@nil\relax\relax}

\def\dtl@getyearsuffix#1#2#3{%
\def\@dtl@argi{#1}\def\@dtl@argii{#2}%
\def\@dtl@argiii{#3}%
\ifx\@dtl@argi\@nnil
 \def\@dtl@year{}%
 \let\@dtl@donext=\relax
\else
  \ifx\@dtl@argii\@nnil
    \dtl@ifsingle{#1}{%
      \def\@dtl@year{#1}%
      \let\@dtl@donext=\relax
    }{%
      \def\@dtl@donext{\dtl@getyearsuffix#1#2#3}%
    }%
  \else
    \ifx\@dtl@argiii\@nnil
      \dtl@ifsingle{#1}{%
        \dtl@ifsingle{#2}{%
          \def\@dtl@year{#1#2}%
          \let\@dtl@donext=\relax
        }{%
          \def\@dtl@donext{\dtl@getyearsuffix#2#3}%
        }%
      }{%
        \def\@dtl@donext{\dtl@getyearsuffix#2#3}%
      }%
    \else
```

```
      \def\@dtl@donext{\dtl@getyearsuffix{#2}{#3}}%
    \fi
  \fi
\fi
\@dtl@donext
}
```

\DTLbibliographystyle

\DTLbibliographystyle{⟨*style*⟩}

Sets the bibliography style.

```
\newcommand*{\DTLbibliographystyle}[1]{%
\@ifundefined{dtlbst@#1}{\PackageError{databib}{Unknown
bibliography style ‘#1’}{}}{\csname dtlbst@#1\endcsname}}
```

Set the default bibliography style:

```
\DTLbibliographystyle{\dtlbib@style}
```

14.8 Multiple Bibliographies

In order to have multiple bibliographies, there needs to be an aux file for each bibliography. The main bibliography is in `\jobname`.aux, but need to provide a means of creating additional aux files.

\DTLmultibibs

\DTLmultibibs{⟨*list*⟩}

This creates an auxiliary file for each name in ⟨*list*⟩. For example, `\DTLmultibibs{foo,bar}` will create the files `foo.aux` and `bar.aux`.

```
\newcommand*{\DTLmultibibs}[1]{%
\@for\@dtl@af:=#1\do{%
\@ifundefined{dtl@aux@\@dtl@af}{%
\expandafter\newwrite\csname dtl@aux@\@dtl@af\endcsname
\expandafter\immediate
\expandafter\openout\csname dtl@aux@\@dtl@af\endcsname=\@dtl@af.aux
\expandafter\def\csname b@\@dtl@af @*\endcsname{}%
}{%
\PackageError{databib}{Can’t create auxiliary file ‘\@dtl@af.aux’,
\expandafter\string\csname dtl@aux@\@dtl@af\endcsname\space
already exists}{}}}}
```

Can only be used in the preamble:

```
\@onlypreamble{\DTLmultibibs}
```

\DTLcite

\DTLcite[⟨*text*⟩]{⟨*mbib*⟩}{⟨*labels*⟩}

This is similar to `\cite[`⟨*text*⟩`]{`⟨*labels*⟩`}`, except 1) the cite information is written to the auxiliary file associated with the multi-bib ⟨*mbib*⟩ (which must be named in `\DTLmultibibs`) and 2) the cross referencing label is constructed from

⟨*mbib*⟩ and ⟨*label*⟩ to allow for the same citation to appear in multiple bibliographies.

```
\newcommand*{\DTLcite}{\@ifnextchar[{\@tempswatrue \dtl@citex
}{\@tempswafalse \dtl@citex[]}}
```

\dtl@citex

```
\def\dtl@citex[#1]#2#3{%
\leavevmode\let\@citea\@empty
\@cite{\@for\@citeb:=#3\do{\@citea
  \def\@citea{,\penalty \@m \ }%
  \edef\@citeb{\expandafter\@firstofone\@citeb\@empty}%
  \if@filesw
    \@ifundefined{dtl@aux@#2}{%
      \PackageError{databib}{multibib ‘#2’ not defined}{%
      You need to define ‘#2’ in \string\DTLmutlibibs}%
    }{%
      \expandafter\immediate
      \expandafter\write\csname dtl@aux@#2\endcsname{%
        \string\citation{\@citeb}}%
    }%
  \fi
  \@ifundefined{b@#2@\@citeb}{%
    \hbox{\reset@font\bfseries ?}%
    \G@refundefinedtrue
    \@latex@warning{Citation ‘\@citeb ’ on page \thepage \space
      undefined}%
  }{%
    \@cite@ofmt{\csname b@#2@\@citeb \endcsname }%
  }%
}}{#1}%
}
```

\DTLnocite

\DTLnocite{⟨*mbib*⟩}{⟨*key list*⟩}

As \nocite but uses the aux file associated with ⟨*mbib*⟩ which must have been defined using \DTLmultibibs.

```
\newcommand*{\DTLnocite}[2]{%
\@ifundefined{dtl@aux@#1}{%
  \PackageError{databib}{multibib ‘#1’ not defined}{%
  You need to define ‘#1’ in \string\DTLmutlibibs}%
}{%
  \@bsphack
  \ifx\@onlypreamble\document
    \@for\@citeb:=#2\do{%
      \edef\@citeb{\expandafter\@firstofone\@citeb}%
      \if@filesw
        \expandafter\immediate
        \expandafter\write\csname dtl@aux@#1\endcsname{%
          \string\citation{\@citeb}}%
      \fi
      \@ifundefined{b@#1@\@citeb}{%
        \G@refundefinedtrue
```

```
        \@latex@warning{Citation `\@citeb ' undefined}}{}%
      }%
    \else
      \@latex@error{Cannot be used in preamble}\@eha
    \fi
    \@esphack
  }%
  }
```

\DTLloadmbbl

\DTLloadmbib{⟨*mbib*⟩}{⟨*db name*⟩}{⟨*bib list*⟩}

```
\newcommand*{\DTLloadmbbl}[3]{%
\@ifundefined{dtl@aux@#1}{%
  \PackageError{databib}{multibib `#1' not defined}{%
  You need to define `#1' in \string\DTLmutlibibs}%
}{%
  \if@filesw
    \expandafter\immediate\expandafter
      \write\csname dtl@aux@#1\endcsname{\string\bibstyle{databib}}%
    \expandafter\immediate\expandafter
      \write\csname dtl@aux@#1\endcsname{\string\bibdata{#3}}%
  \fi
  \DTLnewdb{#2}%
  \edef\DTLBIBdbname{#2}%
  \@input@{#1.bbl}%
}%
}
```

\DTLmbibliography[⟨*condition*⟩]{⟨*mbib name*⟩}{⟨*bib dbname*⟩}

Displays the bibliography for the database ⟨*bib dbname*⟩ which must have previously been loaded using \DTLloadmbbl, where ⟨*mbib name*⟩ must be listed in \DTLmultibibs.

\DTLmbibliography

```
\newcommand*{\DTLmbibliography}[3][\boolean{true}]{%
\begin{DTLthebibliography}[#1]{#3}%
\DTLforeachbibentry[#1]{#3}{%
\DTLmbibitem{#2} \DTLformatbibentry \DTLendbibitem
}%
\end{DTLthebibliography}%
}
```

References

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, 1994.

15 Acknowledgements

Many thanks to Morten Høgholm for suggesting a much more efficient way of storing the information in databases which has significantly improved the time it takes to LaTeX documents containing large databases.

Change History