

The `bytefield` package*

Scott Pakin
`scott+bf@pakin.org`

May 17, 2008

Abstract

The `bytefield` package helps the user create illustrations for network protocol specifications and anything else that utilizes fields of data. These illustrations show how the bits and bytes are laid out in a packet or in memory.

1 Introduction

Network protocols are usually specified in terms of a sequence of bits and bytes arranged in a field. This is portrayed graphically as a grid of boxes. Each row in the grid represents one word (frequently, 32 bits), and each column represents a bit within a word. The `bytefield` package makes it easy to typeset these sorts of figures.

`bytefield` lets one draw protocol diagrams that contain:

- Words of any arbitrary number of bits
- Column headers showing bit positions
- Multiword fields—even non-word-aligned and even if the total number of bits is not a multiple of the word length
- Word labels on either the left or right of the figure
- “Skipped words” within fields

Because `bytefield` draws its figures using only the \LaTeX `picture` environment, these figures are not specific to any particular backend, do not require PostScript support, and do not need support from external programs. Furthermore, unlike an imported graphic, `bytefield` pictures can include arbitrary \LaTeX constructs, such as mathematical equations, `\refs` and `\cites` to the surrounding document, and macro calls.

*This document corresponds to `bytefield` v1.2a, dated 2005/07/31.

2 Usage

2.1 Basic commands

This section explains how to use the `bytefield` package. It lists all the exported environments, commands, and variables in decreasing order of importance.

```
\begin{bytefield} {<bit-width>}  
<fields>  
\end{bytefield}
```

The top-level environment is called, not surprisingly, “`bytefield`”. It takes one (mandatory) argument, which is the number of bits in each word. One can think of a `bytefield` as being analogous to a `tabular`: words are separated by `\\`, and fields within a word are separated by `&`.

```
\wordbox [<sides>] {<height>} {<text>}  
\bitbox [<sides>] {<width>} {<text>}
```

The two main commands one uses within a `bytefield` environment are `\wordbox` and `\bitbox`. The former typesets a field that is one or more words tall and an entire word wide. The latter typesets a field that is one or more bits wide and a single word tall.

The optional argument, `<sides>`, is a list of letters specifying which sides of the field box to draw—`[l]eft`, `[r]ight`, `[t]op`, and/or `[b]ottom`. The default is “`lrtb`” (i.e., all sides are drawn). `<text>` is the text to include within the `\wordbox` or `\bitbox`. It is typeset horizontally centered within a vertically centered `\parbox`. Hence, words will wrap, and `\\` can be used to break lines manually.

The following example shows how to produce a simple 16-bit-wide byte field:

```
\begin{bytefield}{16}  
  \wordbox{1}{A 16-bit field} \\  
  \bitbox{8}{8 bits} & \bitbox{8}{8 more bits} \\  
  \wordbox{2}{A 32-bit field. Note that text wraps within the box.}  
\end{bytefield}
```

The resulting figure looks like this:

| | |
|--|-------------|
| A 16-bit field | |
| 8 bits | 8 more bits |
| A 32-bit field. Note that text wraps within the box. | |

It is the user’s responsibility to ensure that the total number of bits in each row adds up to the number of bits in a single word (the mandatory argument to the `bytefield` environment).

Within a `\wordbox` or `\bitbox`, the `bytefield` package defines `\height`, `\depth`, `\totalheight`, and `\width` to the corresponding dimensions of the box. Section 2.2 gives an example of how these lengths may be utilized.

`\bitheader [endianness] {bit-positions}`

To make the figure more readable, it helps to label bit positions across the top. The `\bitheader` command provides a flexible way to do that. The optional argument, `endianness` is one of “b” or “l” and specifies whether the bits in each word are numbered in big-endian style (right to left) or little-endian style (left to right). The default is little-endian (l).

`\bitheader`’s mandatory argument, `bit-positions`, is a comma-separated list of bit positions to label. For example, “0,2,4,6,8,10,12,14” means to label those bit positions. The numbers must be listed in increasing order. (Use `endianness` to display the header in reverse order.) Hyphen-separated ranges are also valid. For example, “0-15” means to label all bits from 0 to 15, inclusive. While not particularly useful, ranges and single numbers can be intermixed, as in “0-3,8,12-15”.

The following example shows how `\bitheader` may be used:

```
\begin{bytefield}{32}
  \bitheader{0-31} \\
  \bitbox{4}{Four} & \bitbox{8}{Eight} &
  \bitbox{16}{Sixteen} & \bitbox{4}{Four}
\end{bytefield}
```

The resulting figure looks like this:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-------|---|---|---|---|---|---|---------|---|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Four | Eight | | | | | | | Sixteen | | | | | | | | Four | | | | | | | | | | | | | | | |

`\wordgroup {text}`
`\endwordgroup`

`\wordgroupl {text}`
`\endwordgroupl`

When a set of words functions as a single, logical unit, it helps to group these words together visually. All words defined between `\wordgroup` and `\endwordgroup` will be labeled on the right with `text`. Similarly, all words defined between `\wordgroupl` and `\endwordgroupl` will be labeled on the left

with $\langle text \rangle$. $\backslash\text{wordgroup}x$ must lie at the beginning of a row (i.e., right after a $\backslash\backslash$), and $\backslash\text{endwordgroup}x$ must lie right *before* the end of the row (i.e., right before a $\backslash\backslash$).

$\backslash\text{wordgroup}r\dots\backslash\text{endwordgroup}r$ and $\backslash\text{wordgroup}l\dots\backslash\text{endwordgroup}l$ can overlap each other. However, they cannot overlap themselves. In other words, $\backslash\text{wordgroup}r\dots\backslash\text{wordgroup}l\dots\backslash\text{endwordgroup}r\dots\backslash\text{endwordgroup}l$ is a valid sequence, but $\backslash\text{wordgroup}r\dots\backslash\text{wordgroup}r\dots\backslash\text{endwordgroup}r\dots\backslash\text{endwordgroup}r$ is not.

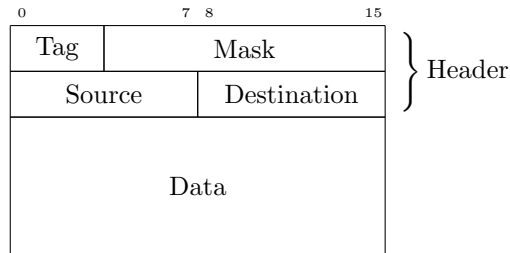
The following example shows how to use $\backslash\text{wordgroup}r$ and $\backslash\text{endwordgroup}r$:

```

\begin{bytefield}{16}
  \bitheader{0,7,8,15} \backslash
  \wordgroup{Header}
    \bitbox{4}{Tag} & \bitbox{12}{Mask} \backslash
    \bitbox{8}{Source} & \bitbox{8}{Destination}
  \endwordgroup \backslash
  \wordbox{3}{Data}
\end{bytefield}

```

Note the juxtaposition of $\backslash\backslash$ to $\backslash\text{wordgroup}r$ and $\backslash\text{endwordgroup}r$ in the above. The resulting figure looks like this:

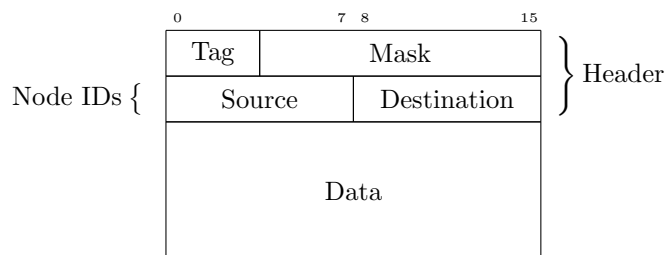


As a more complex example, the following nests left and right labels:

```

\begin{bytefield}{16}
  \bitheader{0,7,8,15} \backslash
  \wordgroup{Header}
    \bitbox{4}{Tag} & \bitbox{12}{Mask} \backslash
    \wordgroup{Node IDs}
      \bitbox{8}{Source} & \bitbox{8}{Destination}
    \endwordgroup
  \endwordgroup \backslash
  \wordbox{3}{Data}
\end{bytefield}

```



Again, note the juxtaposition of `\\` to the various word-grouping commands in the above.

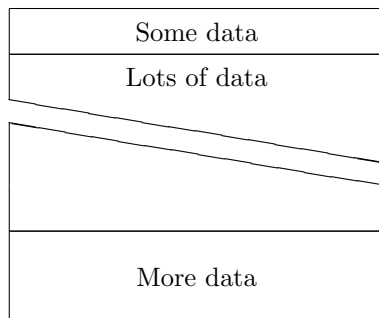
`\skippedwords`

Draw a graphic representing a number of words that are not shown. `\skippedwords` is intended to work with the *sides* argument to `\wordbox`. For example:

```

\begin{bytefield}{16}
  \wordbox{1}{Some data} \\
  \wordbox[lrt]{1}{Lots of data} \\
  \skippedwords \\
  \wordbox[lrb]{1}{ } \\
  \wordbox{2}{More data}
\end{bytefield}

```



`\bitwidth`
`\byteheight`

The above variables represent the width of each bit and height of each byte in the figure. Change them with `\setlength` to adjust the size of the figure. The default value of `\byteheight` is `2ex`, and the default value of `\bitwidth` is the width of `{\tiny 99i}`, i.e., the width of a two-digit number plus a small amount

of extra space. This enables `\bitheader` to show two-digit numbers without overlap.

| |
|--|
| <code>\curlyspace</code> <code>\labelspace</code> |
|--|

`\curlyspace` is the space to insert between the figure and the curly brace preceding a word group (default: `1ex`). `\labelspace` is the space to insert between the curly brace and the label (default: `0.5ex`). Change these with `\setlength` to adjust the spacing.

| |
|------------------------------|
| <code>\curlyshrinkage</code> |
|------------------------------|

In $\text{T}_{\text{E}}\text{X}/\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$, the height of a curly brace does not include the tips. Hence, in a word group label, the tips of the curly brace will extend beyond the height of the word group. `\curlyshrinkage` is an amount by which to reduce the height of curly braces in labels. It is set to `5pt`, and it is extremely unlikely that one would ever need to change it. Nevertheless, it is documented here in case the document is typeset with a math font containing radically different curly braces from the ones that come with $\text{T}_{\text{E}}\text{X}/\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$.

2.2 Common tricks

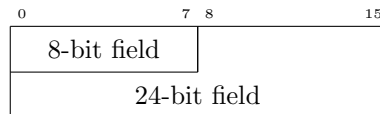
This section shows some clever ways to use `bytefield`'s commands to produce some useful effects.

Odd-sized fields To produce a field that is, say, $1\frac{1}{2}$ words long, use a `\bitbox` for the fractional part and specify appropriate values for the various *sides* parameters. For instance:

```

\begin{bytefield}{16}
  \bitheader{0,7,8,15} \\\
  \bitbox{8}{8-bit field} & \bitbox[lrt]{8}{} \\\
  \wordbox[lrb]{1}{24-bit field}
\end{bytefield}

```



Ellipses To skip words from the middle of enumerated data, put some `\vdots` in a `\wordbox` with empty *sides*:

```
\begin{bytefield}{16}
  \bitbox{8}{Type} & \bitbox{8}{\# of nodes} \\
  \wordbox{1}{Node~1} \\
  \wordbox{1}{Node~2} \\
  \wordbox[]{1}{$\vdots$} \\
  \wordbox{1}{Node~$N$} \\
\end{bytefield}
```

| | |
|---------------|------------|
| Type | # of nodes |
| Node 1 | |
| Node 2 | |
| ⋮ | |
| Node <i>N</i> | |

The extra `1ex` of vertical space helps center the `\vdots` a bit better.

Narrow fields There are a number of options for labeling a narrow field (e.g., one occupying a single bit):

| | | | |
|--|---|----|------|
| <i>Default:</i> | <table border="1"><tr><td>OK</td><td>Data</td></tr></table> | OK | Data |
| OK | Data | | |
| <code>\settowidth{\bitwidth}{OK~}</code> : | <table border="1"><tr><td>OK</td><td>Data</td></tr></table> | OK | Data |
| OK | Data | | |
| <code>\tiny OK:</code> | <table border="1"><tr><td>OK</td><td>Data</td></tr></table> | OK | Data |
| OK | Data | | |
| <code>\tiny 0 \ \ K:</code> | <table border="1"><tr><td>OK</td><td>Data</td></tr></table> | OK | Data |
| OK | Data | | |
| <code>\rotatebox{90}{\small OK}:</code> | <table border="1"><tr><td>OK</td><td>Data</td></tr></table> | OK | Data |
| OK | Data | | |

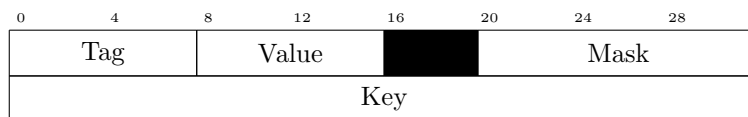
Unused bits Because `\width` and `\height` are defined within `\bitboxes` (also `\wordboxes`), we can represent unused bits by filling a `\bitbox` with a rule of size `\width × \height`:

```
\begin{bytefield}{32}
  \bitheader{0,4,8,12,16,20,24,28} \\
  \bitbox{8}{Tag} & \bitbox{8}{Value} &
```

```

\bitbox{4}{\rule{\width}{\height}} &
\bitbox{12}{Mask} \\
\wordbox{1}{Key}
\end{bytefield}

```

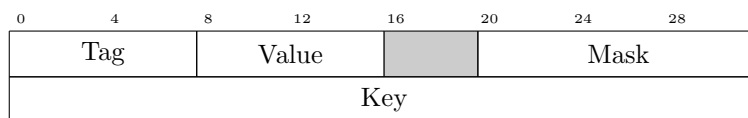


The effect is much better when the `color` package is used to draw the unused bits in color. (Gray looks nice.)

```

\definecolor{lightgray}{gray}{0.8}
\begin{bytefield}{32}
\bitheader{0,4,8,12,16,20,24,28} \\
\bitbox{8}{Tag} & \bitbox{8}{Value} &
\bitbox{4}{\color{lightgray}\rule{\width}{\height}} &
\bitbox{12}{Mask} \\
\wordbox{1}{Key}
\end{bytefield}

```



2.3 Not-so-common tricks

Colored fields A similar approach to that utilized to indicate unused bits can be applied to coloring an individual bit field. The trick is to use the \TeX `\rlap` primitive to draw a colored box that overlaps whatever follows it to the right:

```

\begingroup
\small
\newcommand{\colorbitbox}[3]{%
\rlap{\bitbox{#2}{\color{#1}\rule{\width}{\height}}}%
\bitbox{#2}{#3}}
\definecolor{lightcyan}{rgb}{0.84,1,1}
\definecolor{lightgreen}{rgb}{0.64,1,0.71}
\definecolor{lightred}{rgb}{1,0.7,0.71}

\settowidth{\byteheight}{\sim Sign \sim}
\setlength{\bitwidth}{\baselineskip}
\begin{bytefield}{32}
\bitheader[b]{31,23,0} \\

```



```

\colorbitbox{lightcyan}{1}{\rotatebox{90}{Sign}} &
\colorbitbox{lightgreen}{8}{Exponent} &
\colorbitbox{lightred}{23}{Mantissa} \\
\end{bytefield}
\endgroup

```



Memory-map diagrams While certainly not the intended purpose of the `bytefield` package, one can utilize `\wordboxes` with empty *sides* and word labels to produce memory-map diagrams:

```

\setlength{\byteheight}{4\baselineskip}
\newcommand{\descbox}[2]{\parbox[c][3.8\baselineskip]{0.95\width}{%
\raggedright #1\vfill #2}}
\begin{bytefield}[32]
\wordgroup{Partition 4}
\bitbox[8]{\texttt{0xFFFFFFFF} \\[2\baselineskip]
\texttt{0xC0000000}} &
\bitbox{24}{\descbox{1,GB area for VxDs, memory manager,
file system code; shared by all processes.}{Read/writable.}}
\endwordgroup \\
\wordgroup{Partition 3}
\bitbox[8]{\texttt{0xBFFFFFFF} \\[2\baselineskip]
\texttt{0x80000000}} &
\bitbox{24}{\descbox{1,GB area for memory-mapped files,
shared system \textsc{dll}s, file system code; shared by all
processes.}{Read/writable.}}
\endwordgroup \\
\wordgroup{Partition 2}
\bitbox[8]{\texttt{0x7FFFFFFF} \\[2\baselineskip]
\texttt{0x00400000}} &
\bitbox{24}{\descbox{\sim$2,GB area private to process,
process code, and data.}{Read/writable.}}
\endwordgroup \\
\wordgroup{Partition 1}
\bitbox[8]{\texttt{0x03FFFFFF} \\[2\baselineskip]
\texttt{0x00001000}} &
\bitbox{24}{\descbox{4,MB area for MS-DOS and Windows~3.1
compatibility.}{Read/writable.}} \\
\bitbox[8]{\texttt{0x0000FFF} \\[2\baselineskip]
\texttt{0x00000000}} &
\bitbox{24}{\descbox{4096~byte area for MS-DOS and Windows~3.1
compatibility.}{Protected---catches \textsc{null}
pointers.}}
\endwordgroup \\

```

| | | |
|------------|---|---------------|
| 0xFFFFFFFF | 1 GB area for VxDs, memory manager, file system code; shared by all processes. | } Partition 4 |
| 0xC0000000 | Read/writable. | |
| 0xBFFFFFFF | 1 GB area for memory-mapped files, shared system DLLs, file system code; shared by all processes. | } Partition 3 |
| 0x80000000 | Read/writable. | |
| 0x7FFFFFFF | ~2 GB area private to process, process code, and data. | } Partition 2 |
| 0x00400000 | Read/writable. | |
| 0x003FFFFF | 4 MB area for MS-DOS and Windows 3.1 compatibility. | } Partition 1 |
| 0x00001000 | Read/writable. | |
| 0x00000FFF | 4096 byte area for MS-DOS and Windows 3.1 compatibility. | |
| 0x00000000 | Protected—catches NULL pointers. | |

2.4 Putting it all together

The following code showcases most of `bytefield`'s features in a single figure.

```

\setlength{\byteheight}{2.5\baselineskip}
\begin{bytefield}{32}
  \bitheader{0,7,8,15,16,23,24,31} \
  \wordgroup{\parbox{6em}{\raggedright These words were taken
    verbatim from the TCP header definition (RFC~793).}
    \bitbox{4}{Data offset} & \bitbox{6}{Reserved} &
    \bitbox{1}{\tiny U\R\G} & \bitbox{1}{\tiny A\C\K} &
    \bitbox{1}{\tiny P\S\H} & \bitbox{1}{\tiny R\S\T} &
    \bitbox{1}{\tiny S\Y\N} & \bitbox{1}{\tiny F\I\N} &
    \bitbox{16}{Window} \
    \bitbox{16}{Checksum} & \bitbox{16}{Urgent pointer}
  \endwordgroup \
  \wordbox[lrt]{1}{Data octets} \
  \skippedwords \
  \wordbox[lrb]{1}{ } \
  \wordgroup{\parbox{6em}{\raggedright Note that we can display,
    for example, a misaligned 64-bit value with clever use of the
    optional argument to \texttt{\textbackslash wordbox} and
    \texttt{\textbackslash bitbox}.}
    \bitbox{8}{Source} & \bitbox{8}{Destination} &

```

```

        \bitbox[lrt]{16}{} \\  

        \wordbox[lr]{1}{Timestamp} \\  

        \wordgroup{\parbox{6em}{\raggedright Why two Length fields?  

        No particular reason.}}  

        \bitbox[lrb]{16}{} & \bitbox{16}{Length}  

    \endwordgroup1 \\  

        \bitbox{6}{Key} & \bitbox{6}{Value} & \bitbox{4}{Unused} &  

        \bitbox{16}{Length}  

    \endwordgroup \\  

    \wordbox{1}{Total number of 16-bit data words that follow this  

    header word, excluding the subsequent checksum-type value} \\  

    \bitbox{16}{Data~1} & \bitbox{16}{Data~2} \\  

    \bitbox{16}{Data~3} & \bitbox{16}{Data~4} \\  

    \bitbox[]{16}{\vdots$ \[1ex]} &  

    \bitbox[]{16}{\vdots$ \[1ex]} \\  

    \bitbox{16}{Data~$N-1$} & \bitbox{16}{Data~$N$} \\  

    \bitbox{20}{\[\ \mbox{A5A5}_{\mbox{\scriptsize H}} \oplus  

    \left(\sum_{i=1}^N \mbox{Data}_i \right) \bmod 2^{20} \]} &  

    \bitbox{12}{Command} \\  

    \wordbox{2}{64-bit random number}  

\end{bytefield}

```

Figure 1 shows the resulting protocol diagram.

3 Implementation

This section contains the complete source code for `bytefield`. Most users will not get much out of it, but it should be of use to those who need more precise documentation and those who want to extend (or debug ☺) the `bytefield` package.

In this section, macros marked in the margin with a “★” are intended to be called by the user (and were described in the previous section). All other macros are used only internally by `bytefield`.

```
1 \*package)
```

3.1 Required packages

Although `\widthof` and `\heightof` were introduced in June 1998, `TeX` 2.0—still in widespread use at the time of this writing (2005)—ships with an earlier `calc.sty` in the source directory. Because a misconfigured system may find the source version of `calc.sty` we explicitly specify a later date when loading the `calc` package.

```
2 \RequirePackage{calc}[1998/07/07]
```

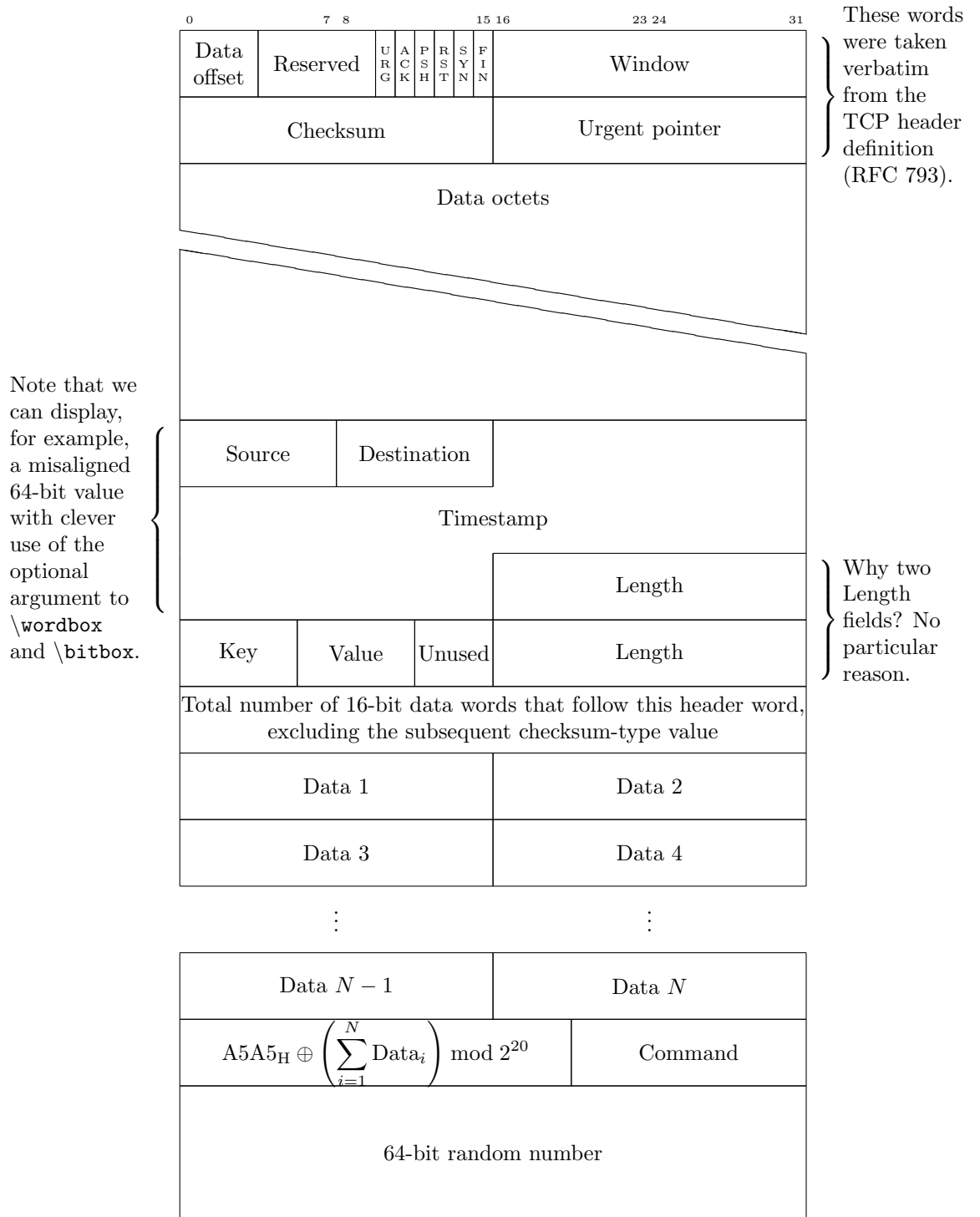


Figure 1: Complex protocol diagram drawn with the `bytefield` package

3.2 Utility macros

The following macros in this section are used by the box-drawing macros and the “skipped words”-drawing macros.

`\bf@newdimen` `\newdimen` defines new $\langle dimen \rangle$ s globally. `\bf@newdimen` defines them locally. It simply merges L^AT_EX 2_ε’s `\newdimen` and `\alloc@` macros while omitting `\alloc@`’s “`\global`” declaration.

```
3 \def\bf@newdimen#1{\advance\count11 by 1
4 \check1\insecunt\dimen% Check room
5 \allocationnumber=\count11
6 \dimendef#1=\allocationnumber
7 \wlog{\string#1=\string\dimen\the\allocationnumber\space (locally)}}
```

`\bytefield@height` When `\ifcounting@words` is TRUE, add the height of the next picture environment to `\bytefield@height`. We set `\counting@wordstrue` at the beginning of each word, and `\counting@wordsfalse` after each `\bitbox`, `\wordbox`, or `\skippedwords` picture.

```
8 \newlength{\bytefield@height}
9 \newif\ifcounting@words
```

`\inc@bytefield@height` We have to define a special macro to increment `\bytefield@height` because the `calc` package’s `\addtolength` macro doesn’t seem to see the global value. So we `\setlength` a temporary (to get `calc`’s nice infix features) and `\advance` `\bytefield@height` by that amount.

```
10 \newlength{\bytefield@height@increment}
11 \DeclareRobustCommand{\inc@bytefield@height}[1]{%
12 \setlength{\bytefield@height@increment}{#1}%
13 \global\advance\bytefield@height by \bytefield@height@increment}
```

3.3 Top-level environment

`bits@wide` The number of bits in each word (i.e., the argument to the `\bytefield` environment).

```
14 \newcounter{bits@wide}
```

`\entire@bytefield@picture` A box containing the entire bytefield. By storing everything in a box and then typesetting it later (at the `\end{bytefield}`), we can center the bytefield, put a box around it, and do other operations on the entire figure.

```
15 \newsavebox{\entire@bytefield@picture}
```

★ `bytefield` Environment containing the layout of bits in a sequence of bytes. This is the main environment defined by the `bytefield` package. The argument is the number of bits wide the bytefield should be. We turn `&` into a space character so the user can think of a `bytefield` as being analogous to a `tabular` environment, even though we’re really setting the bulk of the picture in a single column. (Row labels go in separate columns, however.)

```

16 \newenvironment{bytefield}[1]{%
17   \setcounter{bits@wide}{#1}%
18   \let\old@nl=\%
19   \let\amp=&%
20   \catcode'\&=10
21   \openup -1pt
22   \setlength{bytefield@height}{0pt}%
23   \setlength{unitlength}{1pt}%
24   \counting@wordstrue
25   \begin{lrbox}{\entire@bytefield@picture}%

```

\ \ We redefine \ \ within the bytefield environment to make it aware of curly braces that surround the protocol diagram.

```

26   \renewcommand{\ \}{%
27     \amp\show@wordlabelr\cr%
28     \ignorespaces\counting@wordstrue\make@lspace\amp}%

29   \vbox\bgroup\ialign\bgroup##\amp##\amp##\cr\amp%
30 }{%
31   \amp\show@wordlabelr\cr\egroup\egroup%
32   \end{lrbox}%
33   \usebox{\entire@bytefield@picture}}

```

3.4 Box-drawing macros

3.4.1 Drawing (proper)

★ `\bitwidth` The width of a single bit. Note that this is wide enough to display a two-digit number without it running into adjacent numbers. For larger words, be sure to `\setlength` this larger.

```

34 \newlength{\bitwidth}
35 \AtBeginDocument{\settowidth{\bitwidth}{\tiny 99i}}

```

★ `\byteheight` The height of a single byte.

```

36 \newlength{\byteheight}
37 \AtBeginDocument{\setlength{\byteheight}{4ex}}

```

`\units@wide` Scratch variables for storing the width and height (in points) of the box we're about to draw.

```

38 \newlength{\units@wide}
39 \newlength{\units@tall}

```

★ `\bitbox` Put some text (#3) in a box that's a given number of bits (#2) wide and one byte tall. An optional argument (#1) specifies which lines to draw—[l]eft, [r]ight, [t]op, and/or [b]ottom (default: lrtb).

```

40 \DeclareRobustCommand{\bitbox}[3][lrtb]{%
41   \setlength{\units@wide}{\bitwidth * #2}%
42   \parse@bitbox@arg{#1}%
43   \draw@bit@picture{\strip@pt\units@wide}{\strip@pt\byteheight}{#3}}

```

★ `\wordbox` Put some text (#3) in a box that's a given number of bytes (#2) tall and one word (`bits@wide` bits) wide. An optional argument (#1) specifies which lines to draw—[l]eft, [r]ight, [t]op, and/or [b]ottom (default: lrtb).

```
44 \DeclareRobustCommand{\wordbox}[3][lrtb]{%
45   \setlength{\units@wide}{\bitwidth * \value{bits@wide}}%
46   \setlength{\units@tall}{\byteheight * #2}%
47   \parse@bitbox@arg{#1}%
48   \draw@bit@picture{\strip@pt\units@wide}{\strip@pt\units@tall}{#3}}
```

`\draw@bit@picture` Put some text (#3) in a box that's a given number of units (#1) wide and a given number of units (#2) tall. We format the text with a `\parbox` to enable word-wrapping and explicit line breaks. In addition, we define `\height`, `\depth`, `\totalheight`, and `\width` (à la `\makebox` and friends), so the user can utilize those for special effects (e.g., a `\rule` that fills the entire box). As an added bonus, we define `\widthunits` and `\heightunits`, which are the width and height of the box in multiples of `\unitlength` (i.e., #1 and #2, respectively).

```
49 \DeclareRobustCommand{\draw@bit@picture}[3]{%
50   \begin{picture}(#1,#2)%
```

```

\height First, we plot the user's text, with all sorts of useful lengths predefined.
\depth 51   \put(0,0){\makebox(#1,#2){\parbox[c]{#1\unitlength}{%
\totalheight 52     \bf@newdimen\height
\width 53     \bf@newdimen\depth
\widthunits 54     \bf@newdimen\totalheight
\heightunits 55     \bf@newdimen\width
56     \height=#2\unitlength
57     \depth=0pt%
58     \totalheight=#2\unitlength
59     \width=#1\unitlength
60     \def\widthunits{#1}%
61     \def\heightunits{#2}%
62     \centering #3}}}%
```

Next, we draw each line individually. I suppose we could make a special case for “all lines” and use a `\framebox` above, but the following works just fine.

```
63   \ifbitbox@top
64     \put(0,#2){\line(1,0){#1}}
65   \fi
66   \ifbitbox@bottom
67     \put(0,0){\line(1,0){#1}}
68   \fi
69   \ifbitbox@left
70     \put(0,0){\line(0,1){#2}}
71   \fi
72   \ifbitbox@right
73     \put(#1,0){\line(0,1){#2}}
74   \fi
75 \end{picture}%
```

Finally, we indicate that we're no longer at the beginning of a word. The following code structure (albeit with different arguments to `\inc@bytefield@height`) is repeated in various places throughout this package. We document it only here, however.

```

76 \ifcounting@words
77   \inc@bytefield@height{\unitlength * \real{#2}}%
78   \counting@wordfalse
79 \fi
80 \ignorespaces}

```

3.4.2 Parsing arguments

The macros in this section are used to parse the optional argument to `\bitbox` or `\wordbox`, which is some subset of `{l, r, t, b}`.

`\ifbitbox@top` These macros are set to TRUE if we're to draw the corresponding edge on the subsequent `\bitbox` or `\wordbox`.

`\ifbitbox@bottom`

`\ifbitbox@left` 81 `\newif\ifbitbox@top`

`\ifbitbox@right` 82 `\newif\ifbitbox@bottom`

83 `\newif\ifbitbox@left`

84 `\newif\ifbitbox@right`

`\parse@bitbox@arg` This main parsing macro merely resets the above conditionals and calls a helper function, `\parse@bitbox@sides`.

```

85 \def\parse@bitbox@arg#1{%
86   \bitbox@topfalse
87   \bitbox@bottomfalse
88   \bitbox@leftfalse
89   \bitbox@rightfalse
90   \parse@bitbox@sides#1X}

```

`\parse@bitbox@sides` The helper function for `\parse@bitbox@arg` parses a single letter, sets the appropriate conditional to TRUE, and calls itself tail-recursively until it sees an "X".

```

91 \def\parse@bitbox@sides#1{%
92   \ifx#1X%
93   \else
94     \ifx#1t%
95       \bitbox@toptrue
96     \else
97       \ifx#1b%
98         \bitbox@bottomtrue
99     \else
100      \ifx#1l%
101        \bitbox@lefttrue
102      \else
103        \ifx#1r%
104          \bitbox@righttrue
105        \fi

```



```

106     \fi
107     \fi
108     \fi
109     \expandafter\parse@bitbox@sides
110 \fi}

```

3.5 Skipped words

`\units@high` The height of each diagonal line in the `\skippedwords` graphic. Note that `\units@high = \units@tall - optional argument to \skippedwords`.

```
111 \newlength{\units@high}
```

★ `\skippedwords` Output a fancy graphic representing skipped words. The optional argument is the vertical space between the two diagonal lines (default: `2ex`).

```

112 \DeclareRobustCommand{\skippedwords}[1][2ex]{%
113   \setlength{\units@wide}{\bitwidth * \value{bits@wide}}%
114   \setlength{\units@high}{1pt * \ratio{\units@wide}{6.0pt}}%
115   \setlength{\units@tall}{#1 + \units@high}%
116   \edef\num@wide{\strip@pt\units@wide}%
117   \edef\num@tall{\strip@pt\units@tall}%
118   \edef\num@high{\strip@pt\units@high}%
119   \begin{picture}(\num@wide,\num@tall)
120     \put(0,\num@tall){\line(6,-1){\num@wide}}
121     \put(\num@wide,0){\line(-6,1){\num@wide}}
122     \put(0,0){\line(0,1){\num@high}}
123     \put(\num@wide,\num@tall){\line(0,-1){\num@high}}
124   \end{picture}%
125   \ifcounting@words
126     \inc@bytefield@height{\unitlength * \real{\num@tall}}%
127   \counting@wordsfalse
128 \fi}

```

3.6 Bit-position labels

★ `\bitheader` Output a header of numbered bit positions. The optional argument (`#1`) is “l” for little-endian (default) or “b” for big-endian. The required argument (`#2`) is a list of bit positions to label. It is composed of comma-separated ranges of numbers, for example, “0-31”, “0,7-8,15-16,23-24,31”, or even something odd like “0-7,15-23”. Ranges must be specified in increasing order; use the optional argument to `\bitheader` to reverse the labels’ direction.

```

129 \DeclareRobustCommand{\bitheader}[2][l]{%
130   \parse@bitbox@arg{lrtb}%
131   \setlength{\units@wide}{\bitwidth * \value{bits@wide}}%
132   \setlength{\units@tall}{\heightof{\tiny 9}}%
133   \setlength{\units@high}{\units@tall * -1}%
134   \def\bit@endianness{#1}%
135   \begin{picture}(\strip@pt\units@wide,\strip@pt\units@tall)%
136     (0,\strip@pt\units@high)

```

```

137   \parse@range@list#2,X,
138   \end{picture}%
139   \ifcounting@words
140     \inc@bytefield@height{\unitlength * \real{\strip@pt\units@tall}}%
141     \counting@wordsfalse
142   \fi
143   \ignorespaces}

```

`\parse@range@list` Helper function #1 for `\bitheader`—parse a comma-separated list of ranges, calling `\parse@range` on each range.

```

144 \def\parse@range@list#1,{%
145   \ifx X#1
146   \else
147     \parse@range#1-#1-#1\relax
148     \expandafter\parse@range@list
149   \fi}

```

`\header@xpos` Miscellaneous variables used internally by `\parse@range`— x position of header,
`header@val` current label to output, and maximum label to output (+1).

```

max@header@val 150 \newlength{\header@xpos}
151 \newcounter{header@val}
152 \newcounter{max@header@val}

```

`\parse@range` Helper function #2 for `\bitheader`—parse a hyphen-separated pair of numbers (or a single number) and display the number at the correct bit position.

```

153 \def\parse@range#1-#2-#3\relax{%
154   \setcounter{header@val}{#1}
155   \setcounter{max@header@val}{#2 + 1}
156   \loop
157     \ifnum\value{header@val}<\value{max@header@val}%
158     \if\bit@endianness b%
159       \setlength{\header@xpos}{%
160         \bitwidth * (\value{bits@wide}-\value{header@val}-1)}
161     \else
162       \setlength{\header@xpos}{\bitwidth * \value{header@val}}
163     \fi
164     \put(\strip@pt\header@xpos,0){%
165       \makebox(\strip@pt\bitwidth,\strip@pt\units@tall){%
166         \tiny \theheader@val}}
167     \addtocounter{header@val}{1}
168   \repeat}

```

3.7 Word labels

3.7.1 Curly-brace manipulation

★ `\curlyshrinkage` Reduce the height of a curly brace by `\curlyshrinkage` so its ends don't overlap whatever is above or below it. The default value (5 pt.) was determined empirically and shouldn't need to be changed. However, on the off-chance the user

employs a math font with very different curly braces from Computer Modern’s, `\curlyshrinkage` can be modified.

```
169 \newlength{\curlyshrinkage}
170 \setlength{\curlyshrinkage}{5pt}
```

★ `\curlyspace` Space to insert before a curly brace and before a word label (i.e., after a curly
★ `\labelspace` brace). Because the default values are specified in terms of x heights, we wait until the `\begin{document}` to set them, after the default font has been selected.

```
171 \newlength{\curlyspace}
172 \AtBeginDocument{\setlength{\curlyspace}{1ex}}
173 \newlength{\labelspace}
174 \AtBeginDocument{\setlength{\labelspace}{0.5ex}}
```

`\curly@box` Define a box in which to temporarily store formatted curly braces.

```
175 \newbox{\curly@box}
```

`\store@rcurly` Store a “}” that’s #2 tall in box #1. The only unintuitive thing here is that we
`\curly@height` have to redefine `\fontdimen22`—axis height—to 0 pt. before typesetting the curly
`\half@curly@height` brace. Otherwise, the brace would be vertically off-center by a few points. When
`\curly@shift` we’re finished, we reset it back to its old value.

```
176 \def\store@rcurly#1#2{%
177   \begingroup
178     \bf@newdimen\curly@height%
179     \setlength{\curly@height}{#2 - \curlyshrinkage}%
180     \bf@newdimen\half@curly@height%
181     \setlength{\half@curly@height}{0.5\curly@height}%
182     \bf@newdimen\curly@shift%
183     \setlength{\curly@shift}{\half@curly@height + 0.5\curlyshrinkage}%
184     \global\sbox{#1}{\raisebox{\curly@shift}{%
185       $\xdef\old@axis{\the\fontdimen22\textfont2}$%
186       $\fontdimen22\textfont2=0pt%
187       \left.\vrule height\half@curly@height
188         width 0pt
189         depth\half@curly@height\right\}$%
190       $\fontdimen22\textfont2=\old@axis$}}%
191   \endgroup
192 }
```

`\store@lcurly` Same as `\store@rcurly`, but using a “{” instead of a “}”.

```
\curly@height 193 \def\store@lcurly#1#2{%
\half@curly@height 194   \begingroup
\curly@shift 195     \bf@newdimen\curly@height%
196     \setlength{\curly@height}{#2 - \curlyshrinkage}%
197     \bf@newdimen\half@curly@height%
198     \setlength{\half@curly@height}{0.5\curly@height}%
199     \bf@newdimen\curly@shift%
200     \setlength{\curly@shift}{\half@curly@height + 0.5\curlyshrinkage}%
201     \global\sbox{#1}{\raisebox{\curly@shift}{%
```

```

202     $\xdef\old@axis{\the\fontdimen22\textfont2}$%
203     $\fontdimen22\textfont2=0pt%
204     \left\{\vrule height\half@curly@height
205             width 0pt
206             depth\half@curly@height\right.$%
207     $\fontdimen22\textfont2=\old@axis$}%
208 \endgroup
209 }

```

3.7.2 Right-side labels

`\show@wordlabelr` This macro is output in the third column of every row of the `\ialigned` bytefield table. It's normally a no-op, but `\endwordgroup` defines it to output the word label and then reset itself to a no-op.

```
210 \def\show@wordlabelr{}
```

`\wordlabelr@start` The starting and ending height (in points) of the set of rows to be labelled on the right.

```
211 \newlength{\wordlabelr@start}
212 \newlength{\wordlabelr@end}
```

★ `\wordgroup` Label the words defined between `\wordgroup` and `\endwordgroup` on the right
★ `\endwordgroup` side of the figure. The argument is the text of the label. The label is typeset to the right of a large curly brace, which groups the words together.

```
213 \newenvironment{wordgroup}[1]{%
```

`\wordlabelr@start` `\wordgroup` merely stores the starting height in `\wordlabelr@start` and the
`\wordlabelr@text` user-supplied text in `\wordlabelr@text`. `\endwordgroup` does most of the work.

```

\wordlabelr@end 214 \global\wordlabelr@start=\bytefield@height
215 \gdef\wordlabelr@text{#1}%
216 \ignorespaces%
217 }{%
218 \global\wordlabelr@end=\bytefield@height

```

`\show@wordlabelr` Redefine `\show@wordlabelr` to output `\curlyspace` space, followed by a large curly brace (in `\curlybox`), followed by `\labelspace` space, followed by the user's text (previously recorded in `\wordlabelr@text`). We typeset `\wordlabelr@text` within a `tabular` environment, so L^AT_EX will calculate its width automatically.

```

219 \gdef\show@wordlabelr{%
220     \sbox{\word@label@box}{%
221         \begin{tabular}[b]{@{}l@{}}\wordlabelr@text\end{tabular}}%
222     \settowidth{\label@box@width}{\usebox{\word@label@box}}%
223     \setlength{\label@box@height}{\wordlabelr@end-\wordlabelr@start}%
224     \store@rcurly{\curly@box}{\label@box@height}%
225     \bf@newdimen\total@box@width%
226     \setlength{\total@box@width}{%
227         \curlyspace +
228         \widthof{\usebox{\curly@box}} +

```

```

229     \label@box@width +
230     \label@box@width}%
231 \begin{picture}(\strip@pt\total@box@width,0)
232   \put(0,0){%
233     \hspace*{\curlyspace}%
234     \usebox{\curly@box}%
235     \hspace*{\label@space}%
236     \makebox(\strip@pt\label@box@width,\strip@pt\label@box@height){%
237       \usebox{\word@label@box}}
238   \end{picture}%

```

The last thing `\show@wordlabelr` does is redefine itself back to a no-op.

```

239   \gdef\show@wordlabelr{}}%
240 \ignorespaces}

```

3.7.3 Left-side labels

```

\wordlabel@start The starting and ending height (in points) of the set of rows to be labelled on the
\wordlabel@end   left.
241 \newlength{\wordlabel@start}
242 \newlength{\wordlabel@end}

\total@box@width The total width of the next label to typeset on the left of the figure, that is, the
                  aggregate width of the text box, curly brace, and spaces on either side of the curly
                  brace.
243 \newlength{\total@lbox@width}

\make@l@space    This macro is output in the first column of every row of the \ialigned bytfield
                  table. It's normally a no-op, but \wordgroup1 defines it to output enough space
                  for the next word label and then reset itself to a no-op.
244 \gdef\make@l@space{}

```

★ `\wordgroup1` Same as `\wordgroupr` and `\endwordgroupr`, but put the label on the left.

★ `\endwordgroup1` However, the following code is not symmetric to that of `\wordgroupr` and `\endwordgroupr`. The problem is that we encounter `\wordgroup1` after entering the second (i.e., figure) column, which doesn't give us a chance to reserve space in the first (i.e., left label) column. When we reach the `\endwordgroup1`, we know the height of the group of words we wish to label. However, if we try to label the words in the subsequent first column, we won't know the vertical offset from the "cursor" at which to start drawing the label, because we can't know the height of the subsequent row until we reach the second column.¹

Our solution is to allocate space for the box the next time we enter a first column. As long as space is eventually allocated, the column will expand to fit that space. `\endwordgroup1` outputs the label immediately. Even though `\endwordgroup1` is called at the end of the *second* column, it `\puts` the label at a sufficiently negative x location for it to overlap the first column. Because there

¹Question: Is there a way to push the label up to the *top* of the subsequent row, perhaps with `\vfill`?

will eventually be enough space to accomodate the label, we know that the label won't overlap the figure or extend beyond the figure boundaries.

```
245 \newenvironment{wordgroup1}[1]{%
```

```
\wordlabell@start First, we store the starting height and label text, which are needed by
\wordlabell@text \endwordgroup1.
```

```
246 \global\wordlabell@start=\bytefield@height
247 \gdef\wordlabell@text{#1}%
```

Next, we typeset a draft version of the label into `\word@label@box`, which we measure (into `\total@lbox@width`) and then discard. We can't typeset the final version of the label until we reach the `\endwordgroup1`, because that's when we learn the height of the word group. Without knowing the height of the word group, we don't know how big to make the curly brace. In the scratch version, we make the curly brace 5 cm. tall. This should be more than large enough to reach the maximum curly-brace width, which is all we really care about at this point.

```
248 \sbox{\word@label@box}{%
249   \begin{tabular}[b]{@{}l@{}}\wordlabell@text\end{tabular}}%
250 \settowidth{\label@box@width}{\usebox{\word@label@box}}%
251 \store@lcurly{\curly@box}{5cm}%
252 \setlength{\total@lbox@width}{%
253   \curlyspace +
254   \widthof{\usebox{\curly@box}} +
255   \label@box@width}%
256 \global\total@lbox@width=\total@lbox@width
```

Now we know how wide the box is going to be (unless, of course, the user is using some weird math font that scales the width of a curly brace proportionally to its height). So we redefine `\make@lspace` to output `\total@lbox@width`'s worth of space and then redefine itself back to a no-op.

```
258 \gdef\make@lspace{%
259   \hspace*{\total@lbox@width}%
260   \gdef\make@lspace{}}%
261 \ignorespaces%
262 }%
```

`\endwordgroup1` is comparatively straightforward. We calculate the final height of the word group, and then output the label text, followed by `\label@space` space, followed by a curly brace (now that we know how tall it's supposed to be), followed by `\curly@space` space. The trick, as described earlier, is that we typeset the entire label in the second column, but in a `0x0 picture` environment and with a negative horizontal offset (`\starting@point`), thereby making it overlap the first column.

```
263 \global\wordlabell@end=\bytefield@height
264 \bf@newdimen\starting@point
265 \setlength{\starting@point}{%
266   -\total@lbox@width - \bitwidth*\value{bits@wide}}%
267 \sbox{\word@label@box}{%
268   \begin{tabular}[b]{@{}l@{}}\wordlabell@text\end{tabular}}%
```

```

269 \settoheight{\label@box@height}{\usebox{\word@label@box}}%
270 \setlength{\label@box@width}{\wordlabel@end-\wordlabel@start}%
271 \store@lcurly{\curly@box}{\label@box@height}%
272 \begin{picture}(0,0)
273   \put(\strip@pt\starting@point,0){%
274     \makebox(\strip@pt\label@box@width,\strip@pt\label@box@height){%
275       \usebox{\word@label@box}}%
276     \hspace*{\label@space}%
277     \usebox{\curly@box}%
278     \hspace*{\curly@space}}
279 \end{picture}%
280 \ignorespaces}

```

3.7.4 Scratch space

```

\label@box@width Scratch storage for the width, height, and contents of the word label we're about
\label@box@height to output.
\word@label@box 281 \newlength{\label@box@width}
                282 \newlength{\label@box@height}
                283 \newsavebox{\word@label@box}
                284 </package>

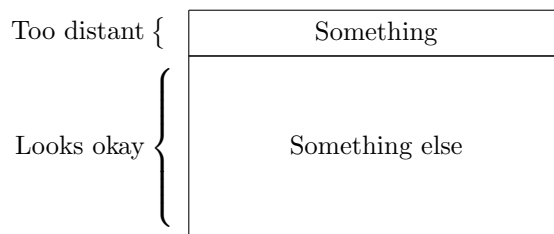
```

4 Future work

`bytefield` is my first \LaTeX package, and, as such, there are a number of macros that could probably have been implemented a lot better. The package should really get a major rewrite. If I were to do it all over again, I would probably not use an `\ialign` for the main `bytefield` environment. The problem—as I discovered too late—is that `\begin...end` blocks are unable to cross cells of an `\ialign` (or `tabular` environment, for that matter).

That said, I'd like the next major release of `bytefield` to let the user use `\begin{wordgroup}[r]...end{wordgroup}` instead of `\wordgroupr...endwordgroup` and `\begin{wordgroup}[l]...end{wordgroup}` instead of `\wordgroupl...endwordgroupl`. That would make the word-grouping commands a little more \LaTeX -ish.

Finally, a minor improvement I'd like to make in the package is to move left, small curly braces closer to the figure. In the following figure, notice how distant the small curly appears from the figure body:



The problem is that the curly braces are left-aligned relative to each other, while they should be right-aligned.

Change History

| | | | |
|---|----|--|----|
| v1.0 | | v1.2 | |
| General: Initial version | 1 | <code>\curly@box</code> : Bug fix: Defined | |
| v1.1 | | <code>\curly@box</code> globally (suggested by Stefan Ulrich) | 19 |
| General: Restructured the <code>.dtx</code> file | 1 | v1.2a | |
| <code>\bf@newdimen</code> : Bug fix: Added <code>\bf@newdimen</code> to eliminate “No room for a new <code>\dimen</code> ” errors (reported by Vitaly A. Repin) | 13 | General: Specified an explicit package date when loading the <code>calc</code> package to avoid loading an outdated version. Thanks to Kevin Quick for discovering that outdated versions of <code>calc</code> are still being included in T _E X distributions. | 11 |
| <code>\parse@range@list</code> : Bug fix: Swapped order of arguments to <code>\ifx</code> test (suggested by Hans-Joachim Widmaier) | 18 | | |

Index

Numbers written in *italics* refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in *roman* refer to the code lines where the entry is used.

| | | | |
|--------------------------------------|----------|--|----------|
| Symbols | | <code>\draw@bit@picture</code> | 49 |
| <code>\</code> | 26 | | |
| B | | E | |
| <code>\bf@newdimen</code> | 3 | <code>\endwordgroup1</code> | 245 |
| <code>\bitbox</code> | 40 | <code>\endwordgroupR</code> | 213 |
| <code>\bitheader</code> | 129 | <code>\entire@bytefield@picture</code> | 15 |
| <code>\bits@wide</code> | 14 | environments: | |
| <code>\bitwidth</code> | 34 | <code>bytefield</code> | 16 |
| <code>bytefield</code> (environment) | 16 | H | |
| <code>\bytefield@height</code> | 8 | <code>\half@curly@height</code> | 176, 193 |
| <code>\byteheight</code> | 36 | <code>\header@val</code> | 150 |
| C | | <code>\header@xpos</code> | 150 |
| <code>\curly@box</code> | 175 | <code>\height</code> | 51 |
| <code>\curly@height</code> | 176, 193 | <code>\heightunits</code> | 51 |
| <code>\curly@shift</code> | 176, 193 | I | |
| <code>\curlyshrinkage</code> | 169 | <code>\ifbitbox@bottom</code> | 81 |
| <code>\curlyspace</code> | 171 | <code>\ifbitbox@left</code> | 81 |
| D | | <code>\ifbitbox@right</code> | 81 |
| <code>\depth</code> | 51 | <code>\ifbitbox@top</code> | 81 |
| | | <code>\ifcounting@words</code> | 8 |

| | | | |
|-----------------------|----------|----------|--|
| \inc@bytefield@height | 10 | | |
| | | L | |
| \label@box@height | 281 | | |
| \label@box@width | 281 | | |
| \labelspace | 171 | | |
| | | M | |
| \make@lspace | 244 | | |
| \max@header@val | 150 | | |
| | | P | |
| \parse@bitbox@arg | 85 | | |
| \parse@bitbox@sides | 91 | | |
| \parse@range | 153 | | |
| \parse@range@list | 144 | | |
| | | S | |
| \show@wordlabelr | 210, 219 | | |
| \skippedwords | 112 | | |
| \store@lcurly | 193 | | |
| \store@rcurly | 176 | | |
| | | T | |
| \total@box@width | 243 | | |
| \totalheight | 51 | | |
| | | U | |
| \units@high | 111 | | |
| \units@tall | 38 | | |
| \units@wide | 38 | | |
| | | W | |
| \width | 51 | | |
| \widthunits | 51 | | |
| \word@label@box | 281 | | |
| \wordbox | 44 | | |
| \wordgroup1 | 245 | | |
| \wordgroupR | 213 | | |
| \wordlabell@end | 241 | | |
| \wordlabell@start | 241, 246 | | |
| \wordlabell@text | 246 | | |
| \wordlabelr@end | 211, 214 | | |
| \wordlabelr@start | 211, 214 | | |
| \wordlabelr@text | 214 | | |