The \texttt{bytefield} package\footnote{This document corresponds to \texttt{bytefield} v2.4, dated 2017/09/15.}

Scott Pakin
\texttt{scott+bf@pakin.org}

September 15, 2017

Abstract

The \texttt{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.

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{Warning:} & \texttt{bytefield} version 2.x breaks compatibility with older versions of the package. See Section 2.7 for help porting documents to the new interface. \\
\hline
\end{tabular}
\end{center}

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, 8, 16, or 32 bits), and each column represents a bit within a word. The \texttt{bytefield} package makes it easy to typeset these sorts of figures. \texttt{bytefield} facilitates drawing protocol diagrams that contain

\begin{itemize}
\item words of any arbitrary number of bits,
\item column headers showing bit positions,
\item multiword fields—even non-word-aligned and even if the total number of bits is not a multiple of the word length,
\item word labels on either the left or right of the figure, and
\item “skipped words” within fields.
\end{itemize}
Because \textit{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, \textit{bytefield} pictures can include arbitrary \LaTeX constructs, such as mathematical equations, \texttt{\ref}s and \texttt{\cite}s to the surrounding document, and macro calls.

2 Usage

2.1 A first example

The Internet Engineering Task Force’s Request for Comments (RFC) number 3016 includes the following ASCII-graphics illustration of the RTP packetization of an MPEG-4 Visual bitstream:

```
0 1 2 3 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| V=2 | P | X | CC | M | PT | sequence number | RTP |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| timestamp | Header |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| synchronization source (SSRC) identifier |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| contributing source (CSRC) identifiers |
| .... |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| MPEG-4 Visual stream (byte aligned) | Pay-load |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| :...OPTIONAL RTP padding |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The following \LaTeX code shows how straightforward it is to typeset that illustration using the \textit{bytefield} package:

```
\begin{bytefield}[bitwidth=1.1em]{32}
\bitheader{0-31} \\
\begin{rightwordgroup}{\texttt{RTP \ Header}}
\bitbox{2}{V=2} & \bitbox{1}{P} & \bitbox{1}{X} & \bitbox{4}{CC} & \bitbox{1}{M} & \bitbox{7}{PT} & \bitbox{16}{sequence number} \\
\bitbox{32}{timestamp} \\
\end{rightwordgroup} \\
\begin{wordbox}[tlr]{1}{\texttt{contributing source (CSRC) identifiers}} \\
\begin{wordbox}[blr]{1}{$\cdots$} \\
\begin{wordbox}[tlr]{1}{\texttt{synchronization source (SSRC) identifier}} \\
\begin{wordbox}[blr]{1}{\texttt{RTP}} \\
\begin{wordbox}[tlr]{1}{\texttt{MPEG-4 Visual stream (byte aligned)}} \\
\begin{wordbox}[blr]{1}{\texttt{Pay-load}} \\
\begin{wordbox}[tlr]{1}{\texttt{:...OPTIONAL RTP padding}} \\
\end{bytefield}
```

2
Figure 1 presents the typeset output of the preceding code. Sections 2.2 and 2.3 explain each of the environments, macros, and arguments that were utilized plus many additional features of the bytefield package.

\begin{bytefield} \[ \langle \text{parameters} \rangle \] \{ \langle \text{bit-width} \rangle \} \langle \text{fields} \rangle \end{bytefield}

The bytefield package’s top-level environment is called, not surprisingly, “bytefield”. It takes one mandatory argument, which is the number of bits in each word, and one optional argument, which is a set of parameters, described in Section 2.3 for formatting the bit-field’s layout. One can think of a bytefield as being analogous to a tabular: words are separated by “\", and fields within a word are separated by “&”. As in a tabular, “\" accepts a \langle length \rangle as an optional argument, and this specifies the amount of additional vertical whitespace to include after the current word is typeset.
The two main commands one uses within a bytefield environment are \bitbox and \wordbox. The former typesets a field that is one or more bits wide and a single word tall. The latter typesets a field that is an entire word wide and one or more words tall.

The optional argument, \langle sides \rangle, is a list of letters specifying which sides of the field box to draw—\langle L \rangleeft, \langle R \rangleight, \langle T \rangleop, and/or \langle B \rangleottom\rangle. The default is \langle lrtb \rangle (i.e., all sides are drawn). \langle text \rangle is the text to include within the \bitbox or \wordbox. It is typeset horizontally centered within a vertically centered \parbox. Hence, words will wrap, and \langle \\container \rangle can be used to break lines manually.

The following example shows how to produce a simple 16-bit-wide 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 bit field looks like this:

\begin{tabular}{|c|c|}
\hline
A 16-bit field & \\
8 bits & 8 more bits \\
\hline
A 32-bit field. Note that text wraps within the box. \\
\hline
\end{tabular}

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); bytefield does not currently check for under- or overruns.

Within a \bitbox or \wordbox, the bytefield package defines \height, \depth, \totalheight, and \width to the corresponding dimensions of the box. Section \ref{sec:2} gives an example of how these lengths may be utilized.

\begin{itemize}
\item \bitboxes \langle \{sides\} \rangle \{ \langle width \rangle \} \{ \langle tokens \rangle \}
\item \bitboxes\* \langle \{sides\} \rangle \{ \langle width \rangle \} \{ \langle tokens \rangle \}
\end{itemize}

The \bitboxes command provides a shortcut for typesetting a sequence of fields of the same width. It takes essentially the same arguments as \bitbox but interprets these differently. Instead of representing a single piece of text to typeset

\footnote{Uppercase L, R, T, and B undo a prior L, r, t, or b and may be useful for writing wrapper commands around \bitbox and \wordbox.}
within a field of width \( \langle \text{width} \rangle \), \texttt{\textbackslash bitboxes}'s \( \langle \text{tokens} \rangle \) argument represents a list of tokens (e.g., individual characters), each of which is typeset within a separate box of width \( \langle \text{width} \rangle \). Consider, for example, the following sequence of \texttt{\textbackslash bitbox} commands:

\begin{verbatim}
\begin{bytefield}{8}
  \bitbox{1}{D} & \bitbox{1}{R} & \bitbox{1}{M} & \bitbox{1}{F} & \bitbox{1}{S} & \bitbox{1}{L} & \bitbox{1}{T} & \bitbox{1}{D}
\end{bytefield}
\end{verbatim}

With \texttt{\textbackslash bitboxes} this can be abbreviated to

\begin{verbatim}
\begin{bytefield}{8}
  \bitboxes{1}{DRMFSLTD}
\end{bytefield}
\end{verbatim}

Spaces are ignored within \texttt{\textbackslash bitboxes}'s \( \langle \text{text} \rangle \) argument, and curly braces can be used to group multiple characters into a single token:

\begin{verbatim}
\begin{bytefield}{24}
  \bitboxes{3}{{DO\,} {RE\,} {MI\,} {FA\,} {SOL\,} {LA\,} {TI\,} {DO}}
\end{bytefield}
\end{verbatim}

The starred form of \texttt{\textbackslash bitboxes} is identical except that it suppresses all internal vertical lines. It can therefore be quite convenient for typesetting binary constants:

\begin{verbatim}
\begin{bytefield}{16}
  \bitboxes*{1}{01000010} & \bitbox{4}{\text{src}} & \bitbox{4}{\text{dest}} & \bitbox{4}{\text{const}}
\end{bytefield}
\end{verbatim}

To make the bit field more readable, it helps to label bit positions across the top. The \texttt{\textbackslash bitheader} command provides a flexible way to do that. The
optional argument is a set of parameters from the set described in Section 2.3. In practice, the only parameters that are meaningful in the context of \bitheader are bitformatting, endianness, and lsb. See Section 2.3 for descriptions and examples of those parameters.

\bitheader’s mandatory argument, \langle bit-positions \rangle, 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 the endianness parameter 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. Ranges and single numbers can even 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 bit field 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 |

When a set of words functions as a single, logical unit, it helps to group these words together visually. All words defined between \begin{rightwordgroup} and \end{rightwordgroup} will be labeled on the right with \langle text \rangle. Similarly, all words defined between \begin{leftwordgroup} and \end{leftwordgroup} will be labeled on the left with \langle text \rangle. \begin{sidewordgroup} must lie at the beginning of a row (i.e., right after a “\"), and \end{sidewordgroup} must lie right before the end of the row (i.e., right before a “\").

Unlike other \TeX environments, rightwordgroup and leftwordgroup do not have to nest properly with each other. However, they cannot overlap themselves. In other words, \begin{rightwordgroup}...\begin{leftwordgroup}...\end{rightwordgroup}...\end{leftwordgroup}...\end{rightwordgroup} is a valid sequence, but \begin{rightwordgroup}...\begin{rightwordgroup}...\end{rightwordgroup}...\end{rightwordgroup} is not.
The following example presents the basic usage of `\begin{rightwordgroup}` and `\end{rightwordgroup}:

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

Note the juxtaposition of "\" to the `\begin{rightwordgroup}` and the `\end{rightwordgroup}` in the above. The resulting bit field looks like this:

```
0 7 8 15
Tag Mask
Source Destination
\text{Header}
\begin{split}
\text{Data}
\end{split}
```

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

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

```
0 7 8 15
Tag Mask
Source Destination
\text{Node IDs}
\begin{split}
\text{Data}
\end{split}
\text{Header}
```
Because \texttt{rightwordgroup} and \texttt{leftwordgroup} are not required to nest properly, the resulting bit field would look the same if the \texttt{\end{leftwordgroup}} and \texttt{\end{rightwordgroup}} were swapped. Again, note the justaposition of “\” to the various word-grouping commands in the above.

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

\bytefieldsetup {⟨key-value list⟩}

Alter the formatting of all subsequent bit fields. Section \ref{sec:formatting} describes the possible values for each \langle key⟩=⟨value⟩ item in the comma-separated list that \texttt{\bytefieldsetup} accepts as its argument. Note that changes made with \texttt{\bytefieldsetup} are local to their current scope. Hence, if used within an environment (e.g., \texttt{figure}), \texttt{\bytefieldsetup} does not impact bit fields drawn outside that environment.

\section{Formatting options}

A document author can customize many of the \texttt{bytefield} package’s figure-formatting parameters, either globally or on a per-figure basis. The parameters described below can be specified in four locations:
• as package options (i.e., in the \usepackage\{options\}\{bytefield\} line),
which affects all bytefield environments in the entire document,
• anywhere in the document using the \bytefieldsetup command, which
affects all subsequent bytefield environments in the current scope,
• as the optional argument to a \begin\{bytefield\}, which affects only that
single bit-field figure, or
• as the optional argument to a \bitheader, which affects only that particular
header. (Only a few parameters are meaningful in this context.)

Unfortunately, \LaTeX{} tends to abort with a “\TeX{} capacity exceeded”
or “Missing \endcsname inserted” error when a control sequence
(i.e., \langle name\rangle or \langle symbol\rangle) is encountered within the optional
argument to \usepackage. Hence, parameters that typically expect
a control sequence in their argument—in particular, bitformatting,
boxformatting, leftcurly, and rightcurly—should best be avoided within the
\usepackage\{options\}\{bytefield\} line.

\begin{bytefield}[\texttt{endianness=\texttt{l}ittle},\texttt{bitwidth=0.11111\linewidth}]\{8\}
\bitheader{0-7} \$
\bitbox{1}\{\text{Res}\} & \bitbox{1}\{\text{BE}\} & \bitbox{1}\{\text{CF}\}
& \bitbox{3}\{$\text{Name}\_\text{Len}-1$} & \bitbox{2}\{\text{Len}\_\text{Len}\} \$
\end{bytefield}

\begin{verbatim}
\begin{bytefield}[\texttt{endianness=\texttt{l}ittle,bitwidth=0.11111\linewidth}]\{8\}
  \bitheader{0-7} \$
  \bitbox{1}\{\text{Res}\} & \bitbox{1}\{\text{BE}\} & \bitbox{1}\{\text{CF}\}
  & \bitbox{3}\{$\text{Name}\_\text{Len}-1$} & \bitbox{2}\{\text{Len}\_\text{Len}\} \$
\end{bytefield}
\end{verbatim}

\begin{tabular}{l}
\textbf{bitwidth} = \langle length\rangle \\
\textbf{bitheight} = \langle length\rangle \\
\end{tabular}

The above parameters represent the width and height of each bit in a bit field.
The default value of \texttt{bitwidth} is the width of “\texttt{\tiny 99i\texttt{}}”, i.e., the width of a
two-digit number plus a small amount of extra space. This enables \texttt{bitheader}
to show two-digit numbers without overlap. The default value of \texttt{bitheight} is 2ex,
which should allow a normal piece of text to appear within a \texttt{bitbox} or \texttt{wordbox}
without abutting the box’s top or bottom edge.

As a special case, if \texttt{bitwidth} is set to the word “\texttt{auto\texttt{}}”, it will be set to the
width of “\texttt{99i\texttt{}}” in the current bit-number formatting (cf. \texttt{bitformatting} below).
This feature provides a convenient way to adjust the bit width after a formatting
change.

\begin{tabular}{l}
\textbf{endianness} = \texttt{\texttt{\texttt{l}}}ittle \texttt{or} \texttt{\texttt{\texttt{b}}}ig \\
\end{tabular}

Specify either little-endian (left-to-right) or big-endian (right-to-left) ordering
of the bit numbers. The default is little-endian numbering. Contrast the following
two examples. The first formats a bit field in little-endian ordering using an
explicit \texttt{endianness=little}, and the second formats the same bit field in big-
endian ordering using \texttt{endianness=big}.

\begin{verbatim}
\begin{bytefield}[\texttt{endianness=\texttt{l}ittle,bitwidth=0.11111\linewidth}]\{8\}
  \bitheader{0-7} \$
  \bitbox{1}\{\text{Res}\} & \bitbox{1}\{\text{BE}\} & \bitbox{1}\{\text{CF}\}
  & \bitbox{3}\{$\text{Name}\_\text{Len}-1$} & \bitbox{2}\{\text{Len}\_\text{Len}\} \$
\end{bytefield}
\end{verbatim}
The numbers that appear in a bit header are typeset in the \texttt{bitformatting} style, which defaults to \texttt{\tiny}. To alter the style of bit numbers in the bit header, set \texttt{bitformatting} to a macro that takes a single argument (like \texttt{\textbf}) or no arguments (like \texttt{\small}). Groups of commands (e.g., \texttt{\{\large\textit\}}) are also acceptable.

When \texttt{bitformatting} is set, \texttt{bitwidth} usually needs to be recalculated as well to ensure that a correct amount of spacing surrounds each number in the bit header. As described above, setting \texttt{bitwidth=auto} is a convenient shortcut for recalculating the bit-width in the common case of bit fields containing no more than 99 bits per line and no particularly wide labels in bit boxes that contain only a few bits.

The following example shows how to use \texttt{bitformatting} and \texttt{bitwidth} to format a bit header with small, boldface text:

\begin{bytefield}[bitformatting={\small\textbf}, bitwidth=auto, endianness=big]{20}
\bitheader{0-19} \ \\
\bitbox{0}{\tiny F/E} & \bitbox{1}{\tiny T0} & \bitbox{1}{\tiny T1} & \bitbox{16}{\textit Data value} \ \\
\end{bytefield}

The resulting bit field looks like this:

\begin{array}{cccccccccccccc}
19 & 18 & 17 & 16 & 15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\end{array}

\begin{array}{cccc}
\texttt{F/E} & \texttt{T0} & \texttt{T1} & \texttt{Fwd} \\
\end{array}

\texttt{Data value}
The text that appears in a `\bitbox` or `\wordbox` is formatted in the `boxformatting` style, which defaults to `\centering`. To alter the style of bit numbers in the bit header, set `boxformatting` to a macro that takes a single argument (like `\textbf` but not `\textbf`—see below) or no arguments (like `\small`). Groups of commands (e.g., `{\large\itshape}`) are also acceptable.

If `boxformatting` is set to a macro that takes an argument, the macro must be defined as a “long” macro, which means it can accept more than one paragraph as an argument. Commands defined with `\newcommand` are automatically made long, but commands defined with `\newcommand*` are not. LaTeX’s `\text... formatting commands (e.g., `\textbf`) are not long and therefore cannot be used directly in `boxformatting`; use the zero-argument versions (e.g., `\bfseries`) instead.

The following example shows how to use `boxformatting` to format the text within each box horizontally centered and italicized:

```latex
\begin{bytefield}[boxformatting={\centering\itshape}, bitwidth=1.5em, endianness=big]{20}
  \bitheader{0-19}  \\
  \bitbox{1}{\tiny F/E} & \bitbox{1}{\tiny T0} & \bitbox{1}{\tiny T1} & \bitbox{1}{\tiny Fwd} & \bitbox{16}{Data value}  \\
\end{bytefield}
```

The resulting bit field looks like this:

```
01 2345 678 9 1011 12131415 161718 19
 F/E T0 T1 Fwd Data value
```

Word groups are normally indicated by a curly brace spanning all of its rows. However, the curly brace can be replaced by any other extensible math delimiter (i.e., a symbol that can meaningfully follow `\left` or `\right` in math mode) via a suitable redefinition of `leftcurly` or `rightcurly`. As in math mode, “." means “no symbol”, as in the following example (courtesy of Steven R. King):

```latex
\begin{bytefield}[rightcurly=. \rightcurlyspace=0pt]{32}
  \bitheader[endianness=big]{0,7,8,15,16,23,24,31}  \\
  \begin{rightwordgroup}{0Ch}
    \bitbox{8}{Byte 15 \" \tiny (highest address)}
    & \bitbox{8}{Byte 14} & \bitbox{8}{Byte 13} & \bitbox{8}{Byte 12}
  \end{rightwordgroup}
\end{bytefield}
```
leftcurlyspace = \langle length \rangle  \\
rightcurlyspace = \langle length \rangle  \\
curlyspace = \langle length \rangle  \\
leftcurlyspace and rightcurlyspace specify the space to insert between the bit field and the curly brace in a left or right word group (default: 1ex). Setting curlyspace is a shortcut for setting both leftcurlyspace and rightcurlyspace to the same value.

leftlabelspace = \langle length \rangle  \\
rightlabelspace = \langle length \rangle  \\
labelspace = \langle length \rangle  \\
leftlabelspace and rightlabelspace specify the space to insert between the curly brace and the text label in a left or right word group (default: 0.5ex). Setting labelspace is a shortcut for setting both leftlabelbrace and rightlabelspace to the same value.

Figure 2 illustrates the juxtaposition of rightcurlyspace and rightlabelspace to a word group and its label. The leftcurlyspace and leftlabelspace parameters are symmetric.
\begin{bytefield}{32}\bitheader[\lsb=0]\{4,12,20,28\} \\\n\bitbox{16}\{\ar$hrd\} & \bitbox{16}\{\ar$pro\} \\
\bitbox{8}\{\ar$hl\} & \bitbox{8}\{\ar$pl\} & \bitbox{16}\{\ar$op\} \\\n\end{bytefield}

\begin{tabular}{|c|c|c|c|}
\hline
4 & 12 & 20 & 28 \\
\hline
\ar$hrd$ & \ar$pro$ & \\
\ar$hl$ & \ar$pl$ & \ar$op$ \\
\hline
\end{tabular}
\begin{bytefield}{32}
\bitheader[lsb=4]{4,12,20,28} \\
\bitbox{16}{ar\$hrd} & \bitbox{16}{ar\$pro} \\
\bitbox{8}{ar\$hln} & \bitbox{8}{ar\$pln} & \bitbox{16}{ar\$op} \\
\end{bytefield}

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ar$hrd</td>
<td>ar$pro</td>
</tr>
<tr>
<td>ar$hln</td>
<td>ar$pln</td>
</tr>
<tr>
<td></td>
<td>ar$op</td>
</tr>
</tbody>
</table>

2.4 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, 1½ words long, use a \bitbox for the fractional part and specify appropriate values for the various \textit{(sides)} parameters. For instance:

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit field</td>
<td></td>
</tr>
<tr>
<td>24-bit field</td>
<td></td>
</tr>
</tbody>
</table>

Ellipses  To skip words that appear the middle of enumerated data, put some \vdots in a \wordbox with empty \textit{(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}

\begin{bytefield}{16}
\bitbox{8}{Type} & \bitbox{8}{\# of nodes} \\
\wordbox{1}{Node-1} \\
\wordbox{1}{Node-2} \\
\wordbox[]{1}{$\vdots$} \\
\wordbox[1ex]{1}{$\vdots$} \\
\wordbox{1}{Node-$N$} \\
\end{bytefield}
<table>
<thead>
<tr>
<th>Type</th>
<th># of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 1</td>
<td></td>
</tr>
<tr>
<td>Node 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Node N</td>
<td></td>
</tr>
</tbody>
</table>

The extra \texttt{\textbackslash ex} of vertical space helps vertically center the \texttt{\textbackslash vdots} a bit better.

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

- **Default**: \verb!ytefieldsetup{%\bitwidth=\widthof{OK~}}:

  \begin{verbatim}
  \verb|	iny OK| \hfil \verb|Data|
  \end{verbatim}

  \begin{verbatim}
  \verb|otatebox{90}{\small OK}| \hfil \verb|Data|
  \end{verbatim}

  \begin{verbatim}
  \verb|\let\bw=\width| \verb|\resizebox{\bw}{!}{~OK~}|
  \end{verbatim}

**Multi-line bit fields** Presentations of wide registers are often easier to read when split across multiple lines. (This capability was originally requested by Chris L’Esperance and is currently implemented in \texttt{bytefield} based on code provided by Renaud Pacalet.) The trick behind the typesetting of multi-line bit fields is to pass the \texttt{lsb} option to \texttt{\bitheader} to change the starting bit number used in each bit header:

\begin{verbatim}
\verb!\begin{bytefield}[endianness=big,bitwidth=2em]{16}! \hfil
\verb!\bitheader[lsb=16]{16-31} \hfil
\verb!\bitbox{1}{\tiny Enable} \hfil \bitbox{7}{Reserved} \hfil
\verb!\bitbox{8}{Bus} \hfil \verb!\bitheader{0-15} \hfil
\verb!\bitbox{5}{Device} \hfil \bitbox{3}{Function} \hfil \bitbox{6}{Register} \hfil
\verb!\bitbox{2}{00}! \hfil
\verb!\end{bytefield}!
\end{verbatim}
Note the use of the optional argument to `\` to introduce three x-heights of additional whitespace between the two rows of bits.

**Rotated bit labels**  A problem with using very large bit numbers is that the labels run into each other, as in the following example:

\begin{verbatim}
\begin{bytefield}[endianness=big]{8}
  \bitheader[lsb=995]{995-1002} \\ \\
  \bitbox{4}{A} & \bitbox{4}{B}
\end{bytefield}
\end{verbatim}

One solution is to use the `bitformatting` option and the `graphicx` package’s `\rotatebox` command to rotate each bit label by 90°. Unfortunately, the naive use of `bitformatting` and `\rotatebox` does not typeset nicely:

\begin{verbatim}
\begin{bytefield}[endianness=big]{8}
  \bitheader[lsb=995, bitformatting={\tiny\rotatebox[origin=B]{90}}]{995-1002} \\ \\
  \bitbox{4}{A} & \bitbox{4}{B}
\end{bytefield}
\end{verbatim}

The two problems are that (1) the numbers are left-justified, and (2) the numbers touch the top margin of the word box. To address these problems we use `\makebox` to construct a right-justified region that is sufficiently wide to hold our largest number plus some additional space to shift the rotated numbers upwards:

\begin{verbatim}
\newlength{\bitlabelwidth}
\newcommand{\rothdr}{\tiny
  \settowidth{\bitlabelwidth}{\quad9999}\%
  \rotatebox[origin=B]{90}{\makebox[\bitlabelwidth][r]{#1}}%}
\end{verbatim}


**Unused bits**  Because \texttt{width} and \texttt{height} are defined within bit boxes (also word boxes), we can represent unused bits by filling a \texttt{bitbox} with a rule of size \texttt{width} \times \texttt{height}:

```
\begin{bytefield}
\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}
```

The effect is much better when the \texttt{color} package is used to draw the unused bits in color. (Light gray looks nice.)

```
\definecolor{lightgray}{gray}{0.8}
\begin{bytefield}
\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}
```
Aligning text on the baseline  Because bytefield internally uses \TeX's \texttt{picture} environment and that environment's \texttt{makebox} command to draw bit boxes and word boxes, the text within a box is centered vertically with no attention paid to the text's baseline. As a result, some bit-field labels appear somewhat askew:

\begin{bytefield}[bitwidth=1.5em]{2}
\bitbox{1}{M} & \bitbox{1}{y}
\end{bytefield}

\begin{tabular}{|c|c|}
\hline
M & y \\
\hline
\end{tabular}

A solution is to use the \texttt{boxformatting} option to trick \texttt{makebox} into thinking that all text has the same height and depth. Here we use \texttt{raisebox} to indicate that all text is as tall as a "W" and does not descend at all below the baseline:

\begin{enumerate}
\item \texttt{newlength{\maxheight}}
\item \texttt{setlength{\maxheight}{\heightof{W}}}
\item \texttt{newcommand{\baselinealign}[1]{{}}}
\item \texttt{centering}
\item \texttt{raisebox{0pt}{\maxheight}[0pt]{#1}}
\end{enumerate}

\begin{bytefield}[boxformatting=\baselinealign, bitwidth=1.5em]{2}
\bitbox{1}{M} & \bitbox{1}{y}
\end{bytefield}

\begin{tabular}{|c|c|}
\hline
M & y \\
\hline
\end{tabular}

Register contents  Sometimes, rather than listing the meaning of each bit field within each \texttt{bitbox} or \texttt{wordbox}, it may be desirable to list the contents, with the meaning described in an additional label above each bit number in the bit header. Although the register package is more suited to this form of layout, bytefield can serve in a pinch with the help of the \texttt{turnbox} macro from the rotating package:

\begin{enumerate}
\item \texttt{newcommand{\bitlabel}[2]{{}}}
\item \texttt{bitbox[][#1]{}}
\item \texttt{raisebox{0pt}{4ex}[Opt]{}}
\item \texttt{\rotatebox{45}{\fontsize{7}{7}\selectfont#2}{}}
\end{enumerate}

\begin{bytefield}
\bitbox{1}{M} & \bitbox{1}{y}
\end{bytefield}
2.5 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:
Omitted bit numbers  It is occasionally convenient to show a wide bit field in which the middle numbers are replaced with an ellipsis. The trick to typesetting such a thing with `bytefield` is to point the `bitformatting` option to a macro that conditionally modifies the given bit number before outputting it. One catch is that `bytefield` measures the height of the string “1234567890” using the current bit formatting, so that needs to be a valid input. (If `bitwidth` is set to “auto”, then “99i” also has to be a valid input, but we’re not using “auto” here.) The following example shows how to conditionally modify the bit number: If the number is 1234567890, it is used as is; numbers greater than 9 are increased by 48; numbers less than 4 are unmodified; the number 6 is replaced by an ellipsis; and all other numbers are discarded.

```latex
\newcommand{\fakesixtyfourbits}[1]{%\tiny
  \ifnum#1=1234567890
    #1
  \else
    \ifnum#1>9
      \count32=#1
      \advance\count32 by 48
      \the\count32
    \else
      \ifnum#1<4
        #1
      \else
        \ifnum#1=6
          \&\cdots
        \fi
      \fi
    \fi
  \fi
}
\begin{bytefield}{%bitwidth=\widthof{\tiny Fwd-},
  bitformatting=\fakesixtyfourbits,
  endianness=big}{16}
  \bitheader{0-15} \bp
  \bitbox{1}{\tiny F/E} \& \bitbox{1}{\tiny T0} \& \bitbox{1}{\tiny T1} \& \bitbox{12}{Data value}
\end{bytefield}
```

31 23 0

<table>
<thead>
<tr>
<th>Exponent</th>
<th>Mantissa</th>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>...</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>F/E</td>
<td>T0</td>
<td>T1</td>
<td>Fwd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data value

20
Memory-map diagrams  While certainly not the intended purpose of the bytefield package, one can utilize word boxes with empty \textit{sides} and word labels to produce memory-map diagrams:

\begin{bytefield}[bitheight=4]\begin{rightwordgroup}{Partition 4}
\bitbox{}{8}{\texttt{0xFFFFFFFF} \\
\texttt{0xC0000000}} & \bitbox{24}{\descbox{1\,GB area for VxDs, memory manager, file system code; shared by all processes.}{Read/writable.}} \\
\end{rightwordgroup} \\
\begin{rightwordgroup}{Partition 3}
\bitbox{}{8}{\texttt{0xBFFFFFFF} \\
\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.}} \\
\end{rightwordgroup} \\
\begin{rightwordgroup}{Partition 2}
\bitbox{}{8}{\texttt{0x7FFFFFFF} \\
\texttt{0x00400000}} & \bitbox{24}{\descbox{$\sim$2\,GB area private to process, process code, and data.}{Read/writable.}} \\
\end{rightwordgroup} \\
\begin{rightwordgroup}{Partition 1}
\bitbox{}{8}{\texttt{0x003FFFFF} \\
\texttt{0x00001000}} & \bitbox{24}{\descbox{4\,MB area for MS-DOS and Windows~3.1 compatibility.}{Read/writable.}} \\
\bitbox{}{8}{\texttt{0x00000FFF} \\
\texttt{0x00000000}} & \bitbox{24}{\descbox{4096-byte area for MS-DOS and Windows~3.1 compatibility.}{Protected---catches \textsc{null} pointers.}} \\
\end{rightwordgroup}
\end{bytefield}
<table>
<thead>
<tr>
<th>Address Range</th>
<th>Description</th>
<th>Partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFFFFFF</td>
<td>1 GB area for VxDs, memory manager, file system code; shared by all processes.</td>
<td>Partition 4</td>
</tr>
<tr>
<td>0xC0000000</td>
<td>Read/writable.</td>
<td></td>
</tr>
<tr>
<td>0xBFFFFFFF</td>
<td>1 GB area for memory-mapped files, shared system DLLs, file system code; shared by all processes.</td>
<td>Partition 3</td>
</tr>
<tr>
<td>0x80000000</td>
<td>Read/writable.</td>
<td></td>
</tr>
<tr>
<td>0x7FFFFFFF</td>
<td>~2 GB area private to process, process code, and data.</td>
<td>Partition 2</td>
</tr>
<tr>
<td>0x00400000</td>
<td>Read/writable.</td>
<td></td>
</tr>
<tr>
<td>0x003FFFFF</td>
<td>4 MB area for MS-DOS and Windows 3.1 compatibility.</td>
<td>Partition 1</td>
</tr>
<tr>
<td>0x00001000</td>
<td>Read/writable.</td>
<td></td>
</tr>
<tr>
<td>0x000000FF</td>
<td>4096 byte area for MS-DOS and Windows 3.1 compatibility.</td>
<td></td>
</tr>
<tr>
<td>0x00000000</td>
<td>Protected—catches NULL pointers.</td>
<td></td>
</tr>
</tbody>
</table>

The following variation uses variable-height regions in the memory map:

```latex
\newcommand{\memsection}[4]{% 
  \bytefieldsetup{bitheight=#3\baselineskip}% 
  \bitbox{16}{#4}% print box with caption 
}% 
\begin{bytefield}{24} 
  \memsection{ffff ffff}{0040 0000}{15}{-- free --}
  \memsection{003f ffff}{002f c000}{4}{Special Function}
\end{bytefield}
```

22
2.6 Putting it all together

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

```latex
\begin{bytefield}[bitheight=2.5\baselineskip]{32}
\bitheader{0,7,8,15,16,23,24,31} \\
\begin{rightwordgroup}{\parbox{6em}{\raggedright These words were taken verbatim from the TCP header definition (RFC-793).}}
23
```
Figure 3 shows the resulting protocol diagram.

### 2.7 Upgrading from older versions

bytefield’s user interface changed substantially with the introduction of version 2.0. Because documents written for bytefield v1.x will not build properly under later versions of the package, this section explains how to convert documents to the new interface.
Note that we can display, for example, a misaligned 64-bit value with clever use of the optional argument to `\wordbox` and `\bitbox`.

Total number of 16-bit data words that follow this header word, excluding the subsequent checksum-type value:

- Data 1
- Data 2
- Data 3
- Data 4
- ...
- Data \(N - 1\)
- Data \(N\)

\[ A5A5_H \oplus \left( \sum_{i=1}^{N} \text{Data}_i \right) \mod 2^{20} = 000010000110 \]

64-bit random number

Figure 3: Complex protocol diagram drawn with the `bytefield` package
These have been replaced with the \texttt{rightwordgroup} environment to make their invocation more \LaTeX-like. Use \texttt{\begin{rightwordgroup}} instead of \texttt{\wordgroupr} and \texttt{\end{rightwordgroup}} instead of \texttt{\endwordgroupr}.

These have been replaced with the \texttt{leftwordgroup} environment to make their invocation more \LaTeX-like. Use \texttt{\begin{leftwordgroup}} instead of \texttt{\wordgroupl} and \texttt{\end{leftwordgroup}} instead of \texttt{\endwordgroupl}.

Instead of changing bit widths with \texttt{\setlength{\bitwidth}{⟨width⟩}}, use \texttt{\bytefieldsetup{bitwidth=⟨width⟩}}.

Instead of changing bit heights with \texttt{\setlength{\byteheight}{⟨height⟩}}, use \texttt{\bytefieldsetup{bitheight=⟨height⟩}} (and note the change from “byte” to “bit” for consistency with \texttt{\bitwidth}).

Instead of using \texttt{\setlength{\curlyspace}{⟨dist⟩}} and \texttt{\setlength{\labelspace}{⟨dist⟩}} to alter the horizontal space that appears before and after a curly brace, use \texttt{\bytefieldsetup{curlyspace=⟨dist⟩}} and \texttt{\bytefieldsetup{labelspace=⟨dist⟩}}. Note that, as described in Section 2.2, left and right spacing can be set independently if desired.

Instead of using \texttt{\setlength{\curlyshrinkage}{⟨dist⟩}} to reduce the vertical space occupied by a curly brace, use \texttt{\bytefieldsetup{curlyshrinkage=⟨dist⟩}}. Note that, as described in Section 2.2, left and right curly-brace height can be reduced independently if desired.
The meaning of \texttt{\bitwidth}'s optional argument changed with bytefield v2.1. In older versions of the package, the optional argument was one of “1” or “b” for, respectively, little-endian or big-endian bit ordering. Starting with version 2.1, the optional argument can be any of the parameters described in Section 2.3 (but practically only \texttt{bitformatting}, \texttt{endianness}, and \texttt{lsb}). Hence, “1” should be replaced with \texttt{endianness=little} and “b” should be replaced with \texttt{endianness=big}. Although more verbose, these new options can be specified once for the entire document by listing them as package options or as arguments to \texttt{bytefieldsetup}.

As a crutch to help build older documents with minimal modification, bytefield provides a \texttt{compat1} package option that restores the old interface. This option, invoked with \texttt{\usepackage[compat1]{bytefield}}, may disappear in a future version of the package and should therefore not be relied upon as a long-term approach to using bytefield.

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 Section 2). All other macros are used only internally by bytefield.

3.1 Required packages

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

\begin{verbatim}
1 \RequirePackage{calc}[1998/07/07]
2 \RequirePackage{keyval}
\end{verbatim}

3.2 Utility macros

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

\texttt{\bf@newdimen} \newdimen defines new \texttt{\dimen}s globally. \texttt{\textbf{\bf@newdimen}} defines them locally. It simply merges \LaTeX{} 2ε’s \texttt{newdimen} and \texttt{alloca} macros while omitting \texttt{alloca}’s “\global” declaration.

\begin{verbatim}
3 \def\bf@newdimen#1{\advance\count11 by 1}
\end{verbatim}
\texttt{\bf@newdimen}\ e-\TeX \ provides \ many \ more \ \langle \text{dimen} \rangle s \ than \ the \ original \ \TeX \ 's \ 255. \ When \ running \ newer \ versions \ of \ e-\TeX \ we \ rebind \ \texttt{\bf@newdimen} \ to \ \texttt{\newdimen}. \ If \ the \ etex \ package \ is \ loaded, \ however, \ we \ instead \ rebind \ \texttt{\bf@newdimen} \ to \ \texttt{\locdimen} \ to \ keep \ the \ allocation \ local. \ Finally, \ if \ we're \ not \ running \ e-\TeX \ we \ leave \ \texttt{\bf@newdimen} \ defined \ as \ above \ to \ help \ reduce \ register \ pressure \ when \ only \ 255 \ \langle \text{dimen} \rangle s \ are \ available.

\AtBeginDocument{%
  \expandafter\ifx\csname e@alloc\endcsname\relax
    \expandafter\ifx\csname locdimen\endcsname\relax
      \else
        \let\bf@newdimen=\locdimen
    \fi
  \else
    \let\bf@newdimen=\newdimen
  \fi
}%

\bf@newdimen

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

\newlength{\bytefield@height}
\newif\ifcounting@words

\inc@bytefield@height

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

\newlength{bytefield@height\@increment}
\DeclareRobustCommand{\inc@bytefield@height}[1]{%\setlength{bytefield@height\@increment}{#1}\%\setlength{bytefield@height}{bytefield@height\@increment}%\%\global\advance\bytefield@height by \bytefield@height\@increment}%

\texttt{3.3 \ Top-level \ environment}

\entire@bytefield@picture

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

\texttt{\star}

The \ \texttt{bytefield} \ environment \ contains \ the \ layout \ of \ bits \ in \ a \ sequence \ of \ words. \ This \ is \ the \ main \ environment \ defined \ by \ the \ \texttt{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.)

26 \newenvironment{bytefield}{2}{}{{% 
27 \bf@bytefieldsetup{#1}%  
28 \renewcommand{\baselinestretch}{%  
29 \selectfont  
30 \def\bits@wide{#2}%  
31 \let\old@nl=\}\%  
32 \let\amp=&%  
33 \catcode\&=10  
34 \openup -1pt  
35 \setlength{\bytefield@height}{0pt}%  
36 \setlength{\unitlength}{1pt}%  
37 \global\counting@wordstrue  
38 \begin{lrbox}{\entire@bytefield@picture}%  
⋆\%  
39 \renewcommand{\}{1}[0pt]{%  
40 \unskip  
41 \vspace{##1}{}  
42 \amp\show@wordlabelr\cr  
43 \ignorespaces\global\counting@wordstrue\make@lspace\amp}\%  
44 \vbox\bgroup\ialign\bgroup##\amp##\amp##\cr\amp  
45 }{%  
46 \amp\show@wordlabelr\cr\egroup\egroup  
47 \end{lrbox}%  
48 \usebox{\entire@bytefield@picture}}%

3.4 Box-drawing macros

3.4.1 Drawing (proper)

\bf@bitformatting Format a bit number in the bit header. \bf@bitformatting may be redefined to take either a single argument (à la \textbf) or no argument (à la \small).  
49 \newcommand*{\bf@bitformatting}{\tiny}

\bf@boxformatting Format the text within a bit box or word box. \bf@boxformatting takes either a single argument (à la \textbf) or no argument (à la \small). The text that follows \bf@boxformatting is guaranteed to be a group that ends in \par, so if \bf@boxformatting accepts an argument, the macro should be defined with \long (e.g., with \newcommand but not with \newcommand*).

50 \newcommand*{\bf@boxformatting}{\centering}
Define 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 \texttt{setlength} this larger.

\begin{verbatim}
\newlength{\bf@bitwidth}
\settowidth{\bf@bitwidth}{\bf@bitformatting{99i}}
\end{verbatim}

This is the height of a single bit within the bit field.

\begin{verbatim}
\newlength{\bf@bitheight}
\setlength{\bf@bitheight}{4ex}
\end{verbatim}

These are scratch variables for storing the width and height (in points) of the box we're about to draw.

\begin{verbatim}
\newlength{\units@wide}
\newlength{\units@tall}
\end{verbatim}

\bitbox

\begin{verbatim}
\DeclareRobustCommand{\bitbox}[3][lrtb]{
\setlength{\units@wide}{\bf@bitwidth * #2}\
\bf@parse@bitbox@arg{#1}\
\draw@bit@picture{\strip@pt\units@wide}{\strip@pt\bf@bitheight}{#3}}
\end{verbatim}

\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). Uppercase letters suppress drawing the [L]eft, [R]ight, [T]op, and/or [B]ottom sides.

\begin{verbatim}
\DeclareRobustCommand{\wordbox}[3][lrtb]{
\setlength{\units@wide}{\bf@bitwidth * \bits@wide}\
\setlength{\units@tall}{\bf@bitheight * #2}\
\bf@parse@bitbox@arg{#1}\
\draw@bit@picture{\strip@pt\units@wide}{\strip@pt\units@tall}{#3}}
\end{verbatim}

\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 \texttt{parbox} to enable word-wrapping and explicit line breaks. In addition, we define \height, \depth, \totalheight, and \width (à la \texttt{makebox} and friends), so the user can utilize those for special effects (e.g., a \texttt{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 \texttt{unitlength} (i.e., \#1 and \#2, respectively).

\begin{verbatim}
\DeclareRobustCommand{\draw@bit@picture}[3]{%
\begin{picture}(#1,#2)\
\height\
\depth\
\totalheight\
\width\
\widthunits\
\heightunits\
\end{picture}}%
\end{verbatim}

First, we plot the user's text, with all sorts of useful lengths predefined.
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.

\ifbitbox@top
\put(0,#2){\line(1,0){#1}}\fi
\ifbitbox@bottom
\put(0,0){\line(1,0){#1}}\fi
\ifbitbox@left
\put(0,0){\line(0,1){#2}}\fi
\ifbitbox@right
\put(#1,0){\line(0,1){#2}}\fi
\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.

\ifcounting@words
\inc@bytefield@height{\unitlength * \real{#2}}\global\counting@wordsfalse
\fi
\ignorespaces}

\bitboxes
\bitboxes*

Put each token in \texttt{#3} into a box that’s a given number of bits (\texttt{#2}) wide and one byte tall. An optional argument (\texttt{#1}) specifies which lines to draw—[l]eft, [r]ight, [t]op, and/or [b]ottom (default: lrtb). Uppercase letters suppress drawing the [L]eft, [R]ight, [T]op, and/or [B]ottom sides. The *-form of the command omits interior left and right lines.

\DeclareRobustCommand{\bitboxes}{%  
\@ifstar{\bf@bitboxes@star}{\bf@bitboxes@no@star}  
}%
\bf@relax

Define a macro that expands to \texttt{\relax} for use with \texttt{\ifx} tests against \texttt{\bf@bitboxes@arg}, which can contain either tokens to typeset or \texttt{\relax}.

\def\bf@relax{\relax}
\bf@bitboxes@no@star Implement the unstarred version of \bitboxes.
102 \newcommand{\bf@bitboxes@no@star}[3][lrtb]{%
\bf@bitboxes@no@star@i Define a helper macro that walks the final argument of \bf@bitboxes@no@star token-by-token.
103 \def\bf@bitboxes@no@star@i##1{%
\bf@bitboxes@arg Store the current argument token in \bf@bitboxes@arg for use with \ifx.
104 \def\bf@bitboxes@arg{##1}%
105 \ifx\bf@bitboxes@arg\bf@relax
106 \let\next=\relax
107 \else
108 \bitbox[#1]{#2}{##1}%
109 \let\next=\bf@bitboxes@no@star@i
110 \fi
111 \next
112 }%
113 \bf@bitboxes@no@star@i#3\relax
114 \ignorespaces
115 }
\bf@bitboxes@star Implement the starred version of \bitboxes.
116 \newcommand{\bf@bitboxes@star}[3][lrtb]{%
\bf@bitboxes@star@i If the argument to \bitboxes* contains a single (or no) token, simply pass control to \bitbox and stop. Otherwise, suppress the box’s right border by appending “R” to \bitboxes*’s argument #1 and proceeding with the remaining tokens in #3.
117 \def\bf@bitboxes@star@i##1##2{%
\bf@bitboxes@arg@i\bf@bitboxes@arg@ii\bf@bitboxes@sides Store the next two argument tokens in \bf@bitboxes@arg@i and \bf@bitboxes@arg@ii for use with \ifx. We use those to set \bf@bitboxes@sides
118 \def\bf@bitboxes@sides{
119 \else
120 \bitbox[#1R]{#2}{##1}%
121 \def\next=\bf@bitboxes@star@ii{##2}%
122 \fi
123 \next
124 }%
\bf@bitboxes@sides Process all tokens in \bitboxes*’s argument #3 following the first argument. For each token, produce a box with the left side suppressed using “L”.
125 \def\bf@bitboxes@sides{#1#2}%
126 \next
127 }%
to \bitbox*’s argument #1 with the left side and, for the final token, the right side suppressed.

\def\bf@bitboxes@arg@i{##1}\
\def\bf@bitboxes@arg@ii{##2}\
\if\bf@bitboxes@arg@ii\bf@relax\
  \def\bf@bitboxes@sides{#1L}\
\else\
  \def\bf@bitboxes@sides{#1LR}\
\fi\
\if\bf@bitboxes@arg@i\bf@relax\
  \let\next=\relax\
\else\
  \expandafter\bitbox\expandafter[\bf@bitboxes@sides]{#2}{##1}\
\def\next{\bf@bitboxes@star@ii{##2}}\
\fi\
\next\
\bf@bitboxes@star@i#3\relax\relax\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, L, R, T, B\}. Lowercase letters display the left, right, top, or bottom side of a box; uppercase letters inhibit the display. The default is not to display any sides, but an uppercase letter can negate the effect of a prior lowercase letter.

\newif\ifbitbox@top
\newif\ifbitbox@bottom
\newif\ifbitbox@left
\newif\ifbitbox@right

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

\bf@parse@bitbox@arg

These macros are set to TRUE if we’re to draw the corresponding edge on the subsequent \bitbox or \wordbox.

\newif\ifbitbox@top
\newif\ifbitbox@bottom
\newif\ifbitbox@left
\newif\ifbitbox@right

\bf@parse@bitbox@sides

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

\begin{lstlisting}[language=TeX]
129 \def\bf@bitboxes@arg@i{##1}\
130 \def\bf@bitboxes@arg@ii{##2}\
131 \if\bf@bitboxes@arg@ii\bf@relax\
132   \def\bf@bitboxes@sides{#1L}\
133 \else\
134   \def\bf@bitboxes@sides{#1LR}\
135 \fi\
136 \if\bf@bitboxes@arg@i\bf@relax\
137   \let\next=\relax\
138 \else\
139   \expandafter\bitbox\expandafter[\bf@bitboxes@sides]{#2}{##1}\
140 \def\next{\bf@bitboxes@star@ii{##2}}\
141 \fi\
142 \next\
143 }\
144 \bf@bitboxes@star@i#3\relax\relax\ignorespaces

147 \newif\ifbitbox@top
148 \newif\ifbitbox@bottom
149 \newif\ifbitbox@left
150 \newif\ifbitbox@right

151 \def\bf@parse@bitbox@arg#1{\%
152 \bitbox@topfalse
153 \bitbox@bottomfalse
154 \bitbox@leftfalse
155 \bitbox@rightfalse
156 \bf@parse@bitbox@sides#1X}\%
157 \def\bf@parse@bitbox@sides#1{\%
\end{lstlisting}
3.5 Skipped words

\units@high\ This is the height of each diagonal line in the \skippedwords graphic. Note that \units@high = \units@tall – optional argument to \skippedwords.

\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).

\DeclareRobustCommand{\skippedwords}[1][2ex]{%
\setlength{\units@wide}{\bf@bitwidth * \bits@wide}%
\setlength{\units@high}{ipt * \ratio{\units@wide}{6.0pt}}%
\fi
\expandafter\bf@parse@bitbox@sides
\PackageWarning{bytefield}{Unrecognized box side `#1'}%
3.6 Bit-position labels

\bf@bit@endianness\ bytefield\ can\ label\ bit\ headers\ in\ either\ little-endian\ (0,\ 1,\ 2,\ \ldots,\ N-1)\ or\ big-endian\ (N-1,\ N-2,\ N-3,\ \ldots,\ 0)\ fashion.\ The\ \bf@bit@endianness\ macro\ specifies\ which\ to\ use,\ either\ “l”\ for\ little-endian\ (the\ default)\ or\ “b”\ for\ big-endian.

\newcommand*{\bf@bit@endianness}{l}

\bf@first@bit\ Normally,\ bits\ are\ numbered\ starting\ from\ zero.\ However,\ \bf@first@bit\ can\ be\ altered\ (usually\ locally)\ to\ begin\ numbering\ from\ a\ different\ value.

\newcommand*{\bf@first@bit}{0}

\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\ lsb\ option\ to\ reverse\ the\ labels’\ direction.

\DeclareRobustCommand{\bitheader}[2][{}]{%
\texttt{\bf@parse@range@list} This is helper function \#1 for \texttt{\bitheader}. It parses a comma-separated list of ranges, calling \texttt{\bf@parse@range} on each range.

\begin{verbatim}
230 \def\bf@parse@range@list#1,{% 
231 \ifx X#1 
232 \else 
233 \bf@parse@range@list#1-#1-#1\relax 
234 \expandafter\bf@parse@range@list 
235 \fi}
\end{verbatim}

Define some miscellaneous variables to be used internally by \texttt{\bf@parse@range}: x position of header, current label to output, and maximum label to output (+1).

\begin{verbatim}
236 \newlength{\header@xpos} 
237 \newcounter{header@val} 
238 \newcounter{max@header@val}
\end{verbatim}

\texttt{\bf@parse@range} This is helper function \#2 for \texttt{\bitheader}. It parses a hyphen-separated pair of numbers (or a single number) and displays the number at the correct bit position.

\begin{verbatim}
239 \def\bf@parse@range#1-#2-#3\relax{% 
240 \setcounter{header@val}{#1} 
241 \setcounter{max@header@val}{#2 + 1} 
242 \loop 
243 \ifnum\value{header@val}<\value{max@header@val}%% 
244 \if\bf@bit@endianness b%% 
245 \setlength{\header@xpos}{% 
246 \bf@bitwidth * (\bits@wide - \value{header@val} + \bf@first@bit - 1)}%% 
247 \else 
248 \setlength{\header@xpos}{\bf@bitwidth * (\value{header@val} - \bf@first@bit)}%% 
249 \fi 
250 \put(\strip@pt\header@xpos,0){%% 
251 \makebox(\strip@pt\bf@bitwidth,\strip@pt\units@tall){%% 
252 \bf@bitformatting{\theheader@val}}} 
253 \addtocounter{header@val}{1} 
254 \repeat}
\end{verbatim}

\texttt{\bf@process@bitheader@opts} This is helper function \#3 for \texttt{\bitheader}. It processes the optional argument to \texttt{\bitheader}.

\begin{verbatim}
\newcommand*{\bf@process@bitheader@opts}{% 
255 \let\KV@bytefield@l=\KV@bitheader@l 
256 \let\KV@bytefield@b=\KV@bitheader@b 
257 \let\KV@bytefield@l@default=\KV@bitheader@l@default 
258 \let\KV@bytefield@b@default=\KV@bitheader@b@default 
259 \setkeys{bytefield} 
260 \repeat}
\end{verbatim}

\texttt{\KV@bitheader@l} \texttt{\KV@bitheader@b} For backwards compatibility we also accept the (now deprecated) \texttt{l} as a synonym for \texttt{endianness=little} and \texttt{b} as a synonym for \texttt{endianness=big}. A typical document will specify an \texttt{endianness} option not as an argument to \texttt{\bitheader} but rather as a package option that applies to the entire document. If the
compat1 option was provided to bytefield (determined below by the existence of the \curlyshrinkage control word), we suppress the deprecation warning message.

\def\bf@bit@endianness{l}%

\def\bf@bit@endianness{b}%

\def\bf@leftcurlyshrinkage{5pt}%
\def\bf@rightcurlyshrinkage{5pt}%

\let\bf@leftcurly=\{
\let\bf@rightcurly=\}
\curly@box \bf@newdimen \curly@height \bf@newdimen \half@curly@height \bf@newdimen \curly@shift \bf@newdimen \old@axis

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

\newbox{\curly@box}

\store@rcurly \curly@height \half@curly@height \curly@shift \old@axis

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

\def\store@rcurly#1#2{%
\begingroup
\bf@newdimen \curly@height
\setlength{\curly@height}{#2 - \bf@rightcurlyshrinkage}%
\bf@newdimen \half@curly@height
\setlength{\half@curly@height}{0.5\curly@height}%
\bf@newdimen \curly@shift
\setlength{\curly@shift}{\bf@rightcurlyshrinkage}%
\setlength{\curly@shift}{\half@curly@height + 0.5\curly@shift}%
\global \sbox{#1}{\raisebox{\curly@shift}{%$
\xdef \old@axis{\the \fontdimen22 \textfont2}$%$
\fontdimen22 \textfont2=0pt$
\left.\vrule height \half@curly@height
width 0pt
depth \half@curly@height \right \bf@rightcurly$%$
\fontdimen22 \textfont2=\old@axis$}}%
\endgroup}%
\endgroup
}

\store@lcurly \half@curly@height \curly@shift

These are the same as \store@rcurly, etc. but using a "{" instead of a "}".

\def\store@lcurly#1#2{%
\begingroup
\bf@newdimen \curly@height
\setlength{\curly@height}{#2 - \bf@leftcurlyshrinkage}%
\bf@newdimen \half@curly@height
\setlength{\half@curly@height}{0.5\curly@height}%
\bf@newdimen \curly@shift
\setlength{\curly@shift}{\bf@leftcurlyshrinkage}%
\setlength{\curly@shift}{\half@curly@height + 0.5\curly@shift}%
\global \sbox{#1}{\raisebox{\curly@shift}{%$
\xdef \old@axis{\the \fontdimen22 \textfont2}$%$
\fontdimen22 \textfont2=0pt$
\left\bf@leftcurly$
\vrule height \half@curly@height
width 0pt
depth \half@curly@height \right.$%$
\fontdimen22 \textfont2=\old@axis$}%
\endgroup}%
\endgroup
}
3.7.2 Right-side labels

This macro is output in the third column of every row of the \ialigned bytefield table. It's normally a no-op, but \end{rightwordgroup} defines it to output the word label and then reset itself to a no-op.

\def\show@wordlabelr{}

\wordlabelr@start and \wordlabelr@end

Declare the starting and ending height (in points) of the set of rows to be labeled on the right.

\newlength{\wordlabelr@start}
\newlength{\wordlabelr@end}

\rightwordgroup

Label the words defined between \begin{rightwordgroup} and \end{rightwordgroup} on the right side of the bit field. 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.

\newenvironment{rightwordgroup}{}{\end{rightwordgroup}}

We begin by ending the group that \begin{rightwordgroup} created. This lets the \rightwordgroup environment span rows (because we're technically no longer within the environment).

\wordlabelr@start
\wordlabelr@text
\begin{rightwordgroup}

merely stores the starting height in \wordlabelr@start and the user-supplied text in \wordlabelr@text. \end{rightwordgroup} does most of the work.

\global{\wordlabelr@start=\bytefield@height}
\gdef{\wordlabelr@text={#1}}
\ignorespaces
\]

\wordlabelr@end

Because we already ended the group that \begin{rightwordgroup} created we now have to begin a group for \end{rightwordgroup} to end.

\begingroup
\global{\wordlabelr@end=\bytefield@height}
\endgroup

\show@wordlabelr

Redefine \show@wordlabelr to output \bf@rightcurlyspace space, followed by a large curly brace (in \curlybox), followed by \bf@rightlabelspace space, followed by the user's text (previously recorded in \wordlabelr@text). We typeset \wordlabelr@text within a \tabular environment, so \TeX will calculate its width automatically.

\gdef{\show@wordlabelr{}}
\sbox{\wordlabelr@box}{%}
\begin{tabular}{l}\wordlabelr@text\end{tabular} %
\setwidth{\label@box@width}{\usebox{\wordlabelr@box}}%
\setlength{\label@box@height}{\wordlabelr@end-\wordlabelr@start}%
\store@rcurly{\curly@box}{\label@box@height}%%
\bf@newdimen{total@box@width}}
The last thing \show@wordlabelr does is redefine itself back to a no-op.

\edef\show@wordlabelr{}}

\@currenvir
Because of our meddling with \begingroup and \endgroup, the current environment is all messed up. We therefore force the \end{rightwordgroup} to succeed, even if it doesn’t match the preceding \begin.

\def\@currenvir{rightwordgroup}%
\ignorespaces
}

3.7.3 Left-side labels

\wordlabell@start\wordlabell@end
Declare the starting and ending height (in points) of the set of rows to be labeled on the left.

\newlength{\wordlabell@start}
\newlength{\wordlabell@end}

\total@box@width
Declare the total width of the next label to typeset on the left of the bit field, that is, the aggregate width of the text box, curly brace, and spaces on either side of the curly brace.

\newlength{\total@box@width}

\make@lspace
This macro is output in the first column of every row of the \ialigned bytefield table. It’s normally a no-op, but \begin{leftwordgroup} defines it to output enough space for the next word label and then reset itself to a no-op.

\gdef\make@lspace{}

\leftwordgroup
This environment is essentially the same as the rightwordgroup environment but puts the label on the left. However, the following code is not symmetric to that of rightwordgroup. The problem is that we encounter \begin{leftwordgroup} 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 \end{leftwordgroup}, 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.\footnote{Question: Is there a way to push the label up to the top
of the subsequent row, perhaps with \texttt{vfill}?
}

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. \texttt{\end{leftwordgroup}} outputs the label immediately. Even though
\texttt{\end{leftwordgroup}} is called at the end of the second column, it \texttt{puts} the label
at a sufficiently negative \(x\) location for it to overlap the first column. Because there
will eventually be enough space to accommodate the label, we know that the label
won’t overlap the bit field or extend beyond the bit-field boundaries.

\begin{leftwordgroup}{leftwordgroup}\[1\]{}

We begin by ending the group that \texttt{\begin{rightwordgroup}} created. This lets
the \texttt{leftwordgroup} environment span rows (because we’re technically no longer
within the environment).

\end{leftwordgroup}

\begin{leftwordgroup}
\wordlabell@start
\wordlabell@text
\end{leftwordgroup}

We store the starting height and label text, which are needed by the
\begin{leftwordgroup}\[1\]{}

Next, we typeset a draft version of the label into \texttt{\word@label@box}, which we
measure (into \texttt{\total@lbox@width}) and then discard. We can’t typeset the final
version of the label until we reach the \texttt{\end{leftwordgroup}}, because that’s when
we learn the height of the word group. Without knowing the height of the word
group, we don’t 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.

\begin{leftwordgroup}
\texttt{sbox}{\word@label@box}{%}
\begin{tabular}{@{}l@{}}\wordlabell@text\end{tabular}}%
\settowidth{\label@box@width}{\usebox{\word@label@box}}%
\setlength{\total@lbox@width}{%\bf@leftcurlyspace + \widthof{\usebox{\curly@box}} + \bf@leftlabelspace + \label@box@width}\%}
\store@lcurly{\curly@box}{5cm}%
\global{\total@lbox@width=\total@lbox@width}
\make@lspace

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 \texttt{\make@lspace} to output \texttt{\total@lbox@width}’s worth of
space and then redefine itself back to a no-op.

\begin{leftwordgroup}\[1\]{}
\gdef\make@lspace{%}
Because we already ended the group that `\begin{leftwordgroup}` created we have to start the `\end{leftwordgroup}` by beginning a group for `\end{leftwordgroup}` to end.

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

Because of our meddling with `\begingroup` and `\end{leftwordgroup}`, the current environment is all messed up. We therefore force the `\end{leftwordgroup}` to succeed, even if it doesn’t match the preceding `\begin`.

3.7.4 Scratch space

Declare some scratch storage for the width, height, and contents of the word label we’re about to output.
3.8 Compatibility mode

bytefield’s interface changed substantially with the move to version 2.0. To give version 1.x users a quick way to build their old documents, we provide a version 1.x compatibility mode. We don’t enable this by default because it exposes a number of extra length registers (a precious resource) and because we want to encourage users to migrate to the new interface.

\newcommand{\bf@enter@compatibility@mode@i}{}% 
\newbytefield{bytefield} Redefine the bytefield environment in terms of the existing (new-interface) bytefield environment. The difference is that the redefinition utilizes all of the preceding lengths.

\let\newbytefield=\bytefield
\let\endnewbytefield=\endbytefield
\renewenvironment{bytefield}[1]{% 
\begin{newbytefield}[% 
bitwidth=\bitwidth, 
bitheight=\byteheight, 
curlyspace=\curlyspace, 
labelspace=\labelspace, 
curlyshrinkage=\curlyshrinkage}{##1}%
\end{newbytefield}% 
}%

\begin{rightwordgroup} % Define \wordgroupr, \endwordgroupr, \wordgroupl, and \endwordgroupl in terms of the new rightwordgroup and leftwordgroup environments.
\def\wordgroupr{\begin{rightwordgroup}}
\def\endwordgroupr{\end{rightwordgroup}}
\def\wordgroupl{\begin{leftwordgroup}}
\def\endwordgroupl{\end{leftwordgroup}}

\bytefieldsetup Disable bytefieldsetup in compatibility mode because it doesn’t work as expected. (Every use of the compatibility-mode bytefield environment overwrites all of the figure-formatting values.)
\renewcommand{\bytefieldsetup}[1]{% 
\PackageError{bytefield}{% 
The \protect{\bytefieldsetup}\space macro is not available in\MessageBreak 
version 1 compatibility mode\}% 
}%

Remove [compat1] from the \protect{\usepackage{bytefield}} line to
make \protect{\bytefieldsetup}\MessageBreak
available to this document.\space\space (The document may also need
to be modified to use\MessageBreak
the new bytefield interface.)
}%
}
}

\let\endwordgroupr=\wordgroupr
\let\wordgroupl=\wordgroupr
\let\endwordgroupl=\wordgroupr

\wordgroupr Issue a helpful error message for the commands that were removed in bytefield v2.0.
While this won’t help users whose first invalid action is to modify a no-longer-extant length register such as \bitwidth or \byteheight, it may benefit at least
a few users who didn’t realize that the bytefield interface has changed substantially
with version 2.0.
\endwordgroupr

\wordgroupl \newcommand{\wordgroupr}{% 
\PackageError{bytefield}{% 
Macros \protect{\wordgroupr}, \protect{\wordgroupl}, \protect{\endwordgroupr},
\MessageBreak
and \protect{\endwordgroupl}\space no longer exist\}% 
}%

Starting with version 2.0, bytefield uses \protect{\begin{wordgroupr}...\MessageBreak
\protect{\end{wordgroupr}} and \protect{\begin{wordgroupl}...\MessageBreak
\protect{\end{wordgroupl}} to specify word groups and a new \protect{\bytefieldsetup}\space macro to
\MessageBreak
change bytefield’s various formatting parameters.%
\endwordgroupl

\let\endwordgroupr=\wordgroupr
\let\wordgroupl=\wordgroupr
\let\endwordgroupl=\wordgroupr

3.9 Option processing

We use the \texttt{keyval} package to handle option processing. Because all of bytefield’s
options have local impact, options can be specified either as package arguments
or through the use of the \texttt{\bytefieldsetup} macro.

\begin{verbatim}
\KV@bytefield@bitwidth
bf@bw@arg
bf@auto
\end{verbatim}

Specify the width of a bit number in the bit header. If the special value “\texttt{auto}” is
given, set the width to the width of a formatted “\texttt{99i}”. 
\begin{verbatim}
\define@key{bytefield}{bitwidth}{% 
\def\bf@bw@arg{#1}% 
\def\bf@auto{auto}% 
\end{verbatim}
\newcommand{\KV@bytefield@bf@bitheight}{\setlength{\bf@bitheight}{#1}}

\newcommand{\KV@bytefield@bitformatting}{\def\bf@bitformatting{#1}}

\newcommand{\KV@bytefield@boxformatting}{\def\bf@boxformatting{#1}}

\newcommand{\KV@bytefield@leftcurly}{\def\bf@leftcurly{#1}}
\newcommand{\KV@bytefield@rightcurly}{\def\bf@rightcurly{#1}}

\newcommand{\KV@bytefield@leftcurlyspace}{\def\bf@leftcurlyspace{#1}}
\newcommand{\KV@bytefield@rightcurlyspace}{\def\bf@rightcurlyspace{#1}}
\newcommand{\KV@bytefield@curlyspace}{\def\bf@leftcurlyspace{#1}\def\bf@rightcurlyspace{#1}}

\newcommand{\KV@bytefield@leftlabelspace}{\def\bf@leftlabelspace{#1}}
\newcommand{\KV@bytefield@rightlabelspace}{\def\bf@rightlabelspace{#1}}
\newcommand{\KV@bytefield@labelspace}{\def\bf@leftlabelspace{#1}\def\bf@rightlabelspace{#1}}

\newcommand{\KV@bytefield@leftcurlyshrinkage}{\def\bf@leftcurlyshrinkage{#1}}
\newcommand{\KV@bytefield@rightcurlyshrinkage}{\def\bf@rightcurlyshrinkage{#1}}
\newcommand{\KV@bytefield@curlyshrinkage}{\def\bf@leftcurlyshrinkage{#1}\def\bf@rightcurlyshrinkage{#1}}

\setlength{\bf@bitwidth}{\bf@auto}%
\settowidth{\bf@bitwidth}{\bf@bitformatting{99i}}%
\else
\setlength{\bf@bitwidth}{#1}%
\fi

\define@key{bytefield}{bitheight}{\setlength{\bf@bitheight}{#1}}
\define@key{bytefield}{bitformatting}{\def\bf@bitformatting{#1}}
\define@key{bytefield}{boxformatting}{\def\bf@boxformatting{#1}}
\define@key{bytefield}{leftcurly}{\def\bf@leftcurly{#1}}
\define@key{bytefield}{rightcurly}{\def\bf@rightcurly{#1}}
\define@key{bytefield}{leftcurlyspace}{\def\bf@leftcurlyspace{#1}}
\define@key{bytefield}{rightcurlyspace}{\def\bf@rightcurlyspace{#1}}
\define@key{bytefield}{curlyspace}{\def\bf@leftcurlyspace{#1}\def\bf@rightcurlyspace{#1}}
\define@key{bytefield}{leftlabelspace}{\def\bf@leftlabelspace{#1}}
\define@key{bytefield}{rightlabelspace}{\def\bf@rightlabelspace{#1}}
\define@key{bytefield}{labelspace}{\def\bf@leftlabelspace{#1}\def\bf@rightlabelspace{#1}}
\define@key{bytefield}{leftcurlyshrinkage}{\def\bf@leftcurlyshrinkage{#1}}
\define@key{bytefield}{rightcurlyshrinkage}{\def\bf@rightcurlyshrinkage{#1}}
\define@key{bytefield}{curlyshrinkage}{\def\bf@leftcurlyshrinkage{#1}\def\bf@rightcurlyshrinkage{#1}}

\setlength{\bf@bitheight}{#1} Specify the height of a bit in a \bitbox or \wordbox.
\setlength{\bf@bitheight}{#1} \define@key{bytefield}{bitheight}{\setlength{\bf@bitheight}{#1}}

\setlength{\bf@bitwidth}{#1} Specify the style of a bit number in the bit header. This should be passed an expression that takes either one argument (e.g., \textit) or no arguments (e.g., \\small\bfseries).
\setlength{\bf@bitwidth}{#1} \define@key{bytefield}{bitformatting}{\def\bf@bitformatting{#1}}

\setlength{\bf@bitwidth}{#1} Specify a style to be applied to the contents of every bit box and word box. This should be passed an expression that takes either one argument (e.g., \textit) or no arguments (e.g., \\small\bfseries).
\setlength{\bf@bitwidth}{#1} \define@key{bytefield}{boxformatting}{\def\bf@boxformatting{#1}}

\def\bf@leftcurly{#1} \def\bf@rightcurly{#1} \define@key{bytefield}{leftcurly}{\def\bf@leftcurly{#1}} \define@key{bytefield}{rightcurly}{\def\bf@rightcurly{#1}}
\def\bf@leftcurlyspace{#1} \def\bf@rightcurlyspace{#1} \define@key{bytefield}{leftcurlyspace}{\def\bf@leftcurlyspace{#1}} \define@key{bytefield}{rightcurlyspace}{\def\bf@rightcurlyspace{#1}}
\def\bf@leftlabelspace{#1} \def\bf@rightlabelspace{#1} \define@key{bytefield}{leftlabelspace}{\def\bf@leftlabelspace{#1}} \define@key{bytefield}{rightlabelspace}{\def\bf@rightlabelspace{#1}}
\def\bf@curlyshrinkage{#1} \def\bf@leftcurlyshrinkage{#1} \def\bf@rightcurlyshrinkage{#1} \define@key{bytefield}{curlyshrinkage}{\def\bf@curlyshrinkage{#1}} \define@key{bytefield}{leftcurlyshrinkage}{\def\bf@leftcurlyshrinkage{#1}} \define@key{bytefield}{rightcurlyshrinkage}{\def\bf@rightcurlyshrinkage{#1}}
\bytefield{endianness}{\{little\} or \{big\}}

\bytefield{lsb}{\{first\} bit of a word}

\bytefieldsetup{Reconfigure values for various bytefield parameters. Internally to the package we use the \bytefieldsetup macro instead of \bytefieldsetup. This enables us to redefine \bytefieldsetup when entering version 1 compatibility mode without impacting the rest of bytefield.}

\bf{compat1} instructs bytefield to enter version 1 compatibility mode—at the cost of a number of additional length registers and the inability to specify parameters in the argument to the bytefield environment.

\bf{options} We want to use \bytefieldsetup to process bytefield package options. Unfortunately, \bf{options} doesn’t handle \(\text{key}=\text{value}\) arguments. Hence,
we use `\DeclareOption*` to catch all options, each of which it appends to `\bf@package@options`. `\bf@package@options` is passed to `\bf@bytefieldsetup` only at the beginning of the document so that the options it specifies (a) can refer to ex-heights and (b) override the default values, which are also set at the beginning of the document.

```
526 \def\bf@package@options{}
527 \DeclareOption*{%
528 \edef\next{%
529 \noexpand\g@addto@macro\noexpand\bf@package@options{,\CurrentOption}%
530 }%
531 \next
532 }
533 \ProcessOptions\relax
534 \expandafter\bf@bytefieldsetup\expandafter{\bf@package@options}
```

## 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. For example, `bytefield` is somewhat wasteful of (\texttt{dimen}) registers (although it did get a lot better with version 1.1 and again with version 1.3). The package should really get a major overhaul now that I’ve gotten better at \TeX/\LaTeX programming. One minor improvement I’d like to make in the package is to move left, small curly braces closer to the bit field. In the following figure, notice how distant the small curly appears from the bit-field body:

![Figure showing the problem with small curly braces in `bytefield`]

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

## Change History

v1.0

- General: Initial version ........... 1

v1.1

- \allocationnumber: Bug fix:
  - Added `\bf@newdimen` to

- \bf@parse@range@list: Bug fix:

  - greatly reduce the likelihood of “No room for a new \dimen” errors (reported by Vitaly A. Repin) ................. 27
Swapped order of arguments to \ifx test (suggested by Hans-Joachim Widmaier) ... 36
General: Restructured the .dtx file ... 1

v1.2
\curly@box: Bug fix: Defined \curly@box globally (suggested by Stefan Ulrich) ... 38

v1.2a
General: Specified an explicit package date when loading the calc package to avoid loading an outdated version. Thanks to Kevin Quick for discovering that outdated versions of calc are still being included in TeX distributions. ............ 27

v1.3
\bf@newdimen: Added support for \TeX's larger local ⟨dimen⟩ pool (code provided by Heiko Oberdiek) ............ 28

v1.4
\bf@bitformatting: Introduced this macro at Steven R. King's request to enable users to alter the bit header's font size ... 29
General: Made assignments to \counting@words global to prevent vertical-spacing problems with back-to-back word groups (bug fix due to Steven R. King) .................. 29
Split \curlyspace, \labelspace, and \curlyshrinkage into left and right versions ........... 1

v2.0
\bytefieldsetup: Introduced this macro to provide a more convenient way of configuring bytefield's parameters ........ 1
General: Made a number of non-backwards-compatible changes, including replacing \wordgroup and \endwordgroup with a leftwordgroup environment and \wordgroup and \endwordgroup with a rightwordgroup environment

v2.1
\: Augmented the definition of \ to accept an optional argument, just like in a \tabular environment ......... 29
\bf@parse@range: Added code due to Renaud Pacalet for shifting the bit header by a distance corresponding to \bf@first@bit, used for typesetting registers split across rows .................. 36
\bitheader: Changed the optional argument to accept ⟨key⟩=⟨value⟩ pairs instead of just “1” and “b” ............ 35
General: Included in the documentation a variable-height memory-map example suggested by Martin Demling .......... 22

v2.2
\bitboxes: Added this macro based on an idea proposed by Andrew Mertz ......... 31

v2.3
\bf@newdimen: Rewrote the macro based on discussions with David Carlisle to avoid producing “No room for a new \dimen” errors in newer versions of \TeX (cf. \url{http://tex.stackexchange.com/q/275042}) ............. 28

v2.4
bytefield: Make the code resilient to changes in \baselinestretch. Thanks to Karst Koymans for the bug report ............... 28
Index

Numbers written in italic 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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>@currenvir</td>
<td>361, 405</td>
</tr>
<tr>
<td>@ifstar</td>
<td>99</td>
</tr>
<tr>
<td>|</td>
<td>31, 35</td>
</tr>
<tr>
<td>}</td>
<td>288, 289</td>
</tr>
<tr>
<td>A</td>
<td>allocationnumber 3</td>
</tr>
<tr>
<td>&amp;</td>
<td>3</td>
</tr>
<tr>
<td>@</td>
<td>26, 42, 43, 44, 46</td>
</tr>
<tr>
<td>\AtBeginDocument</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>baselinestretch 28</td>
</tr>
<tr>
<td>\bf@arg</td>
<td>505, 506, 509</td>
</tr>
<tr>
<td>\bf@auto</td>
<td>469</td>
</tr>
<tr>
<td>\bf@big</td>
<td>504, 509</td>
</tr>
<tr>
<td>\bf@bit@endianness</td>
<td>213, 244, 270, 280, 507, 510</td>
</tr>
<tr>
<td>\bf@bitboxes@arg</td>
<td>104, 118</td>
</tr>
<tr>
<td>\bf@bitboxes@arg@i</td>
<td>129</td>
</tr>
<tr>
<td>\bf@bitboxes@arg@ii</td>
<td>129</td>
</tr>
<tr>
<td>\bf@bitboxes@no@star</td>
<td>99, 102</td>
</tr>
<tr>
<td>\bf@bitboxes@no@star@i</td>
<td>103</td>
</tr>
<tr>
<td>\bf@bitboxes@sides</td>
<td>129</td>
</tr>
<tr>
<td>\bf@bitboxes@star</td>
<td>99, 116</td>
</tr>
<tr>
<td>\bf@bitboxes@star@i</td>
<td>117</td>
</tr>
<tr>
<td>\bf@bitboxes@star@ii</td>
<td>124, 128</td>
</tr>
<tr>
<td>\bf@bitformatting</td>
<td>49, 52, 218, 252, 473, 479</td>
</tr>
<tr>
<td>\bf@bitheight</td>
<td>63, 60, 63, 478</td>
</tr>
<tr>
<td>\bf@bitwidth</td>
<td>51, 58, 62, 197</td>
</tr>
<tr>
<td>\bf@boxformatting</td>
<td>50, 79, 480</td>
</tr>
<tr>
<td>\bf@bw@arg</td>
<td>469</td>
</tr>
<tr>
<td>\bf@bytefieldsetup</td>
<td>27, 523, 534</td>
</tr>
<tr>
<td>\bf@enter@compatibility@mode@i</td>
<td>510, 525</td>
</tr>
<tr>
<td>\bf@first@bit</td>
<td>213, 246, 248, 522</td>
</tr>
<tr>
<td>\bf@first@curly</td>
<td>288, 322, 481</td>
</tr>
<tr>
<td>\bf@first@curly@shrinkage</td>
<td>252, 313, 317, 495, 498</td>
</tr>
<tr>
<td>\bf@leftcurly@space</td>
<td>284, 297, 312, 314, 340, 389</td>
</tr>
<tr>
<td>\bf@leftcurly@space</td>
<td>284, 297, 312, 314, 340, 389</td>
</tr>
<tr>
<td>\bf@leftcurly@space</td>
<td>284, 297, 312, 314, 340, 389</td>
</tr>
<tr>
<td>\bf@leftlabel@space</td>
<td>284, 379, 401, 489, 492</td>
</tr>
<tr>
<td>\bf@little</td>
<td>503, 506</td>
</tr>
<tr>
<td>\bf@newdimen</td>
<td>3</td>
</tr>
<tr>
<td>\bf@package@options</td>
<td>526</td>
</tr>
<tr>
<td>\bf@parse@bitbox@arg</td>
<td>59, 64, 119, 216</td>
</tr>
<tr>
<td>\bf@parse@bitbox@sides</td>
<td>156, 157</td>
</tr>
<tr>
<td>\bf@parse@endianness</td>
<td>501</td>
</tr>
<tr>
<td>\bf@parse@range</td>
<td>233, 239</td>
</tr>
<tr>
<td>\bf@parse@range@list</td>
<td>222, 230</td>
</tr>
<tr>
<td>\bf@process@bitheader@opts</td>
<td>220, 230</td>
</tr>
<tr>
<td>\bf@relax</td>
<td>101, 105, 119, 131, 136</td>
</tr>
<tr>
<td>\bf@rightcurly@shrinkage</td>
<td>288, 306, 482</td>
</tr>
<tr>
<td>\bf@rightcurly@space</td>
<td>283, 294, 298, 406, 499</td>
</tr>
<tr>
<td>\bf@rightcurly@space</td>
<td>283, 294, 298, 406, 499</td>
</tr>
<tr>
<td>\bf@rightcurly@space</td>
<td>283, 294, 298, 406, 499</td>
</tr>
<tr>
<td>\bf@rightcurly@space</td>
<td>283, 348, 354, 484, 487</td>
</tr>
<tr>
<td>\bf@rightcurly@space</td>
<td>283, 348, 354, 484, 487</td>
</tr>
<tr>
<td>\bf@rightcurly@space</td>
<td>283, 348, 354, 484, 487</td>
</tr>
<tr>
<td>\bf@rightcurly@space</td>
<td>283, 348, 354, 484, 487</td>
</tr>
<tr>
<td>\bf@right@label@space</td>
<td>283, 350, 356, 490, 493</td>
</tr>
<tr>
<td>\bf@right@label@space</td>
<td>283, 350, 356, 490, 493</td>
</tr>
<tr>
<td>\bytebox@bottomfalse</td>
<td>153, 176</td>
</tr>
<tr>
<td>\bytebox@bottomtrue</td>
<td>153, 176</td>
</tr>
<tr>
<td>\bytebox@leftfalse</td>
<td>154, 179</td>
</tr>
<tr>
<td>\bytebox@lefttrue</td>
<td>154, 179</td>
</tr>
<tr>
<td>\bytebox@rightfalse</td>
<td>167</td>
</tr>
<tr>
<td>\bytebox@righttrue</td>
<td>167</td>
</tr>
<tr>
<td>\bytebox@topfalse</td>
<td>152, 173</td>
</tr>
<tr>
<td>\bytebox@toptrue</td>
<td>161</td>
</tr>
<tr>
<td>\byteboxes</td>
<td>4</td>
</tr>
<tr>
<td>\byteboxes*</td>
<td>98</td>
</tr>
<tr>
<td>bitformatting (option)</td>
<td>4, 10, 16</td>
</tr>
<tr>
<td>bitheader</td>
<td>4</td>
</tr>
<tr>
<td>bitheight (option)</td>
<td>9</td>
</tr>
<tr>
<td>bits@wide</td>
<td>20, 21, 246, 249, 391</td>
</tr>
<tr>
<td>bitwidth (option)</td>
<td>7</td>
</tr>
<tr>
<td>boxformatting (option)</td>
<td>9, 18</td>
</tr>
<tr>
<td>bytefield</td>
<td>422</td>
</tr>
</tbody>
</table>