The \texttt{bytefield} package\textsuperscript{*}

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

October 31, 2020

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.

\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.

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

- 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, and
- “skipped words” within fields.

\textsuperscript{*}This document corresponds to \texttt{bytefield} v2.6, dated 2020/10/31.
Because `bytefield` draws its figures using only the \LaTeX picture environment, these figures are not specific to any particular backend, do not require PostScript or PDF 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, \texttt{\ref} and \texttt{\cite} 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:

![RTP Packetization Illustration](image)

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

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

The following `bytefield` code shows how straightforward it is to typeset that illustration using the `bytefield` package:
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.

<table>
<thead>
<tr>
<th>V=2</th>
<th>P</th>
<th>X</th>
<th>CC</th>
<th>M</th>
<th>PT</th>
<th>sequence number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RTP Header

synchronization source (SSRC) identifier

contributing source (CSRC) identifiers

\ldots

MPEG-4 Visual stream (byte aligned)

\ldots\textit{optional} RTP padding

Figure 1: Sample bytefield output

2.2 Basic commands

This section explains how to use the bytefield package. It lists all of the exported macros and environments in approximately decreasing order of usefulness.

\begin{bytefield} \[⟨options⟩] \{(bit-width)\} \langle 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 comma-separated list of \langle key\rangle=\langle value\rangle pairs, described in Section 2.3 for formatting the bit-field’s layout. One can think of a bytefield as being analogous to a \texttt{tabular}: words are separated by “\textbackslash”, and fields within a word are separated by “\&”. As in a \texttt{tabular}, “\textbackslash” 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 \texttt{bytefield} environment are \texttt{\bitbox} and \texttt{\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 first, optional, argument, \langle\textit{sides}\rangle, is a list of letters specifying which sides of the field box to draw—\texttt{l}eft, \texttt{r}ight, \texttt{t}op, and/or \texttt{b}ottom. The default is \texttt{"lrtb"} (i.e., all sides are drawn). The second, required, argument is the width in bits of a bit box or the height in words of a word box. The third argument is an optional, comma-separated list of \langle\textit{key}\rangle=\langle\textit{value}\rangle pairs, described in Section 2.3. The fourth, required, argument is the text to typeset within the box. It is typeset horizontally centered within a vertically centered \texttt{\parbox}. Hence, words will wrap, and \texttt{"\\"\"} can be used to break lines manually.

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

\begin{verbatim}
\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}
\end{verbatim}

The resulting bit field looks like this:

\begin{tabular}{|c|c|}
\hline
A 16-bit field & \\
8 bits & 8 more bits \\
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 \texttt{bytefield} environment); \texttt{bytefield} does not currently check for under- or overruns.

Here’s an example of using the \texttt{bgcolor} option to fill each box with a different color:

\begin{verbatim}
\definecolor{lightcyan}{rgb}{0.84,1,1}
\definecolor{lightgreen}{rgb}{0.64,1,0.71}
\definecolor{lightred}{rgb}{1,0.7,0.71}
\begin{bytefield}[bitheight=\widthof{"Sign"},
boxformatting={\centering\small}]{32}
\end{verbatim}

\footnote{Uppercase \texttt{L, R, T, and B} undo a prior \texttt{l, r, t, or b} and may be useful for writing wrapper commands around \texttt{\bitbox} and \texttt{\wordbox}.}
Within a \texttt{bitbox} or \texttt{wordbox}, the bytefield package defines \texttt{height}, \texttt{depth}, \texttt{totalheight}, and \texttt{width} to the corresponding dimensions of the box. Section 2.4 gives an example of how these lengths may be utilized.

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

With \texttt{bitboxes} this can be abbreviated to

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

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

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

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

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

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

To make the bit field more readable, it helps to label bit positions across the top. The \bitheader command provides a flexible way to do that. The optional argument is a comma-separated list of \langle key \rangle=\langle value \rangle pairs 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:

\begin{bytefield}{32}
  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
\end{bytefield}
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 (text). Similarly, all words defined between \begin{leftwordgroup} and \end{leftwordgroup} will be labeled on the left with (text). \begin{(side)wordgroup} must lie at the beginning of a row (i.e., right after a “\”), and \end{(side)wordgroup} must lie right before the end of the row (i.e., right before a “\”).

The optional argument is a comma-separated list of \text{\text{key}}=\text{\text{value}} pairs from the set described in Section 2.3. In practice, only curlystyle, leftcurlystyle, and rightcurlystyle, make sense within the context of a \begin{(side)wordgroup}.

Unlike other \LaTeX{} 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{rightwordgroup}...\end{rightwordgroup} is a valid sequence, but \begin{rightwordgroup}...\begin{rightwordgroup}...\end{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:
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}
  \end{bytefield}

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 justapoistion of “\\” to the various word-grouping commands in the above.

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

\skippedwords

Draw a graphic representing a number of words that are not shown. \texttt{\skippedwords} is intended to work with the \texttt{(sides)} argument to \texttt{\wordbox}, as in the following example:

\begin{bytefield}{16}
  \wordbox{1}{Some data} \\
  \wordbox[1rt]{1}{Lots of data} \\
  \skippedwords
\end{bytefield}
Alter the formatting of all subsequent bit fields. Section 2.3 describes the possible values for each \(\text{\textlangle key\rangle=\text\langle value\rangle}\) pair 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.

### 2.3 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 six locations:

- as package options (i.e., in the \texttt{\usepackage[\langle options\rangle]\{bytefield\}} line), which affects all \texttt{bytefield} environments in the entire document,
- anywhere in the document using the \texttt{\bytefieldsetup} command, which affects all subsequent \texttt{bytefield} environments in the current scope,
- as the optional argument to a \texttt{\begin{bytefield}}, which affects only that single bit-field figure, or
- as the optional argument to a \texttt{\bitheader}, which affects only that particular header. (Only a few parameters are meaningful in this context.)
- as the optional argument to a \texttt{\begin{leftwordgroup}} or \texttt{\begin{rightwordgroup}}, which affects only that particular word group. (Only a few parameters are meaningful in this context.)
• as the second optional argument to a \bitbox, \wordbox, or \bitboxes, which affects only that particular box. (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[\langle options\rangle]{bytefield} line.

\begin{verbatim}
bitwidth = \langle length\rangle
bitheight = \langle length\rangle
\end{verbatim}

The above parameters represent the width and height of each bit in a bit field. 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. The default value of bitheight is 2ex, which should allow a normal piece of text to appear within a \bitbox or \wordbox without abutting the box’s top or bottom edge.

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

\begin{verbatim}
endianness = little or big
\end{verbatim}

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 endianness=little, and the second formats the same bit field in big-endian ordering using endianness=big.

\begin{verbatim}
\begin{bytefield}[endianness=little,bitwidth=0.11111\linewidth]{{8}}
  \bitheader{0-7} \$\\
  \bitbox{1}{Res} & \bitbox{1}{BE} & \bitbox{1}{CF} & \bitbox{3}{$\mbox{Name\_Len}-1$} & \bitbox{2}{Len\_Len} \$
  \end{bytefield}
\end{verbatim}

\begin{verbatim}
0 1 2 3 4 5 6 7
Res BE CF Name\_Len − 1 Len\_Len
\end{verbatim}
The numbers that appear in a bit header are typeset in the `bitformatting` style, which defaults to `\tiny`. To alter the style of bit numbers in the bit header, set `bitformatting` to a macro that takes a single argument (like `\textbf`) or no arguments (like `\small`). Groups of commands (e.g., `{\large\itshape}`) are also acceptable.

When `bitformatting` is set, `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 `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 `bitformatting` and `bitwidth` to format a bit header with small, boldface text:

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

The resulting bit field looks like this:

<table>
<thead>
<tr>
<th>F/E</th>
<th>T0</th>
<th>T1</th>
<th>Fwd</th>
<th>Data value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 \textbf{boxformatting} to a macro that takes a single argument (like \textbf{but not \textit{—see below}) or no arguments (like \textbf{small}). Groups of commands (e.g., {\large\textit{shape}}) are also acceptable.

If \textbf{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 \textbf{\newcommand} are automatically made long, but commands defined with \textbf{\newcommand*} are not. \LaTeX{}’s \textbf{... formatting commands} (e.g., \textbf{\textit{But not \textbf{—see below}) or no arguments (like \textbf{small}). Groups of commands (e.g., {\large\textit{shape}}) are also acceptable.}

The following example shows how to use \textbf{boxformatting} to format the text within each box horizontally centered and italicized:

\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:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>F/E</td>
<td>T0</td>
<td>T1</td>
<td>Fwd</td>
<td>Data value</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></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\textbf{bgcolor = \langle color\rangle}

Bit and word boxes are normally left unfilled. The \textbf{bgcolor} option fills them with a specified background color. A document will need to include the \textbf{color}, \textbf{xcolor}, or similar package to expose color names to \textbf{bytefield}. The \textbf{boxformatting} option described above can be used to set the foreground color.

\textbf{leftcurly = \langle delimiter\rangle} \\
\textbf{rightcurly = \langle delimiter\rangle}

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 \textbf{left} or \textbf{right} in math mode) via a suitable redefinition of \textbf{leftcurly} or \textbf{rightcurly}. As in math mode, “.” means “no symbol”, as in the following example (courtesy of Steven R. King):

\begin{bytefield}[rightcurly=., rightcurlyspace=0pt]{32} \\
\bitheader{endianness=big}{0,7,8,15,16,23,24,31} \\
\begin{rightwordgroup}{0Ch}

12
\begin{bytefield}
\begin{rightwordgroup}{08h}
\begin{rightwordgroup}{04h}
\begin{rightwordgroup}{00h}
\begin{rightwordgroup} \tiny (highest address)
\end{bytefield}

\begin{tabular}{|l|l|l|l|}
\hline
\textbf{Byte 15} & \textbf{Byte 14} & \textbf{Byte 13} & \textbf{Byte 12} \\
\hline
\textbf{Long 0} & & & 0Ch \\
\hline
\textbf{Word 1} & \textbf{Word 0} & & \\
\hline
\textbf{Byte 3} & \textbf{Byte 2} & \textbf{Byte 1} & \textbf{Byte 0} \\
\hline
\end{tabular}

\textbf{leftcurlyspace} = \langle \text{length} \rangle
\textbf{rightcurlyspace} = \langle \text{length} \rangle
\textbf{curlyspace} = \langle \text{length} \rangle

\textbf{leftcurlyspace} and \textbf{rightcurlyspace} specify the space to insert between the bit field and the curly brace in a left or right word group (default: 1ex). Setting \textbf{curlyspace} is a shortcut for setting both \textbf{leftcurlyspace} and \textbf{rightcurlyspace} to the same value.

\textbf{leftlabelspace} = \langle \text{length} \rangle
\textbf{rightlabelspace} = \langle \text{length} \rangle
\textbf{labelspace} = \langle \text{length} \rangle

\textbf{leftlabelspace} and \textbf{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 \textbf{labelspace} is a shortcut for setting both \textbf{leftlabelbrace} and \textbf{rightlabelspace} to the same value.

13
Figure 2 illustrates the juxtaposition of \texttt{rightcurlyspace} and \texttt{rightlabelspace} to a word group and its label. The \texttt{leftcurlyspace} and \texttt{leftlabelspace} parameters are symmetric.

\begin{figure}
\centering
\begin{verbatim}
\begin{bytefield}
\bitheader{0-15} \begin{rightwordgroup}{Sign-extended} \bitbox{4}{Tag} & \bitbox{12}{Data} \end{rightwordgroup} \end{bytefield}
\end{verbatim}
\caption{Role of \texttt{rightcurlyspace} and \texttt{rightlabelspace}}
\end{figure}

\begin{verbatim}
leftcurlyshrinkage = (\texttt{length})
rightcurlyshrinkage = (\texttt{length})
curlyshrinkage = (\texttt{length})
\end{verbatim}

In \LaTeX, 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. \texttt{leftcurlyshrinkage/rightcurlyshrinkage} is an amount by which to reduce the height of the curly brace in a left/right word group’s label. Setting \texttt{curlyshrinkage} is a shortcut for setting both \texttt{leftcurlyshrinkage} and \texttt{rightcurlyshrinkage} to the same value. Shrinkages default to 5pt, and it is extremely unlikely that one would ever need to change them. Nevertheless, these parameters are included here in case a document is typeset with a math font containing radically different curly braces from the ones that come with \LaTeX or that replaces the curly braces (using \texttt{leftcurly/rightcurly}, described above) with symbols of substantially different heights.

\begin{verbatim}
leftcurlystyle = (\texttt{command})
rightcurlystyle = (\texttt{command})
curlystyle = (\texttt{command})
\end{verbatim}

Provide a macro that will be invoked before the code that draws left, right, or both curly braces. The macro must accept either zero or one argument. It can be used, for example, to color the curly brace:

\begin{verbatim}
\begin{bytefield}{curlystyle=\color{blue}}{16}
 \bitheader{0-15} \begin{rightwordgroup}{Sign-extended} \bitbox{4}{Tag} & \bitbox{12}{Data} \end{rightwordgroup} \end{bytefield}
\end{verbatim}
Designate the least significant bit (LSB) in the bit header. By default, the LSB is zero, which means that the first bit position in the header corresponds to bit 0. Specifying a different LSB shifts the bit header such that the first bit position instead corresponds to \( \langle \text{integer} \rangle \). Note that the `lsb` option affects bit positions regardless of whether these positions are labeled, as demonstrated by the following two examples:

\begin{bytefield}{32}
  \bitheader[lsb=0]{4,12,20,28} \\n  \bitbox{16}{ar\$hrd} & \bitbox{16}{ar\$pro} \\n  \bitbox{8}{ar\$hln} & \bitbox{8}{ar\$pln} & \bitbox{16}{ar\$op} \\n\end{bytefield}

\begin{bytefield}{32}
  \bitheader[lsb=4]{4,12,20,28} \\n  \bitbox{16}{ar\$hrd} & \bitbox{16}{ar\$pro} \\n  \bitbox{8}{ar\$hln} & \bitbox{8}{ar\$pln} & \bitbox{16}{ar\$op} \\n\end{bytefield}
Provide a macro that will be invoked once for each word in a word box after the regular content is rendered. The macro will be passed two arguments: the word number (starting from 0) and the total number of words in the word box. Furthermore, the macro will be invoked within a one-word-wide box positioned at the base of the word. \texttt{perword} can therefore be used for delineating words within a word box, numbering words, or performing other such annotations. As a simple example, the following code draws a gray line at the bottom of each word in the “Descriptive text” word box:

\begin{verbatim}
\newcommand{\wordline}[2]{\color[rgb]{0.7,0.7,0.7}\hrulefill}
\begin{bytefield}[bitwidth=4em]{8}
  \bitheader[lsb=1,bitformatting=\small]{1-8} \\
  \wordbox[lrt]{7}{\perword=\wordline}{Descriptive text (60 bytes)} \\
  \bitbox[lrb]{4}{} & \bitbox{4}{subsys data offset} \\
  \bitbox{4}{subsys data offset} & \bitbox{2}{version} & \bitbox{2}{endian indicator} \\
\end{bytefield}
\end{verbatim}

\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
\hline
Descriptive text (60 bytes) \\
\hline
subsys data offset & subsystem data offset & version & endian indicator \\
\hline
\end{tabular}
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 \langle sides\rangle 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}

\begin{center}
\begin{tabular}{cccc}
0 & 7 & 8 & 15 \\
\hline
8-bit field \\
\hline
24-bit field
\end{tabular}
\end{center}

Ellipses  To skip words that appear the middle of enumerated data, put some \vdots in a \wordbox with empty \langle sides\rangle:

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

\begin{center}
\begin{tabular}{lc}
Type & \# of nodes \\
\hline
Node 1 \\
Node 2 \\
\vdots \\
Node $N$
\end{tabular}
\end{center}

The extra \text{1ex} of vertical space helps vertically 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):

\begin{tabular}{|l|l|}
  \hline
  Default: & OK \hline
  \bytefieldsetup{%
    \renewcommand{\bitwidth}{\widthof{OK}}:
  }
  & OK \hline
  \tiny OK: & OK \hline
  \tiny 0 \ \ & Data \hline
  \rotatebox{90}{\small OK}: & OK \hline
  \let\bw=\width
  \resizebox{\bw}{!}{~OK~} & Data \hline
\end{tabular}

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 bytefield based on code provided by Renaud Pacalet.) The trick behind the typesetting of multi-line bit fields is to pass the lsb option to \bitheader to change the starting bit number used in each bit header:

\begin{bytefield}{16}
  \bitheader{lsb=16}{16-31} \par
  \bitbox{1}{\tiny Enable} & \bitbox{7}{Reserved} \\
  \bitbox{8}{Bus} \\
  \bitheader{0-15} \par
  \bitbox{5}{Device} & \bitbox{3}{Function} & \bitbox{6}{Register} \\
  \bitbox{2}{00}
\end{bytefield}

16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
\hline
Enable & Reserved & Bus \hline
\end{tabular}

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
\hline
Device & Function & Register & 00 \hline
\end{tabular}

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

\begin{tabular}{cc}
  \tiny A & \tiny B \\
\end{tabular}

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

\begin{tabular}{cc}
  \tiny A & \tiny B \\
\end{tabular}

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:

\newlength{bitlabelwidth}
\newcommand{\rotbitheader}[1]{% 
  \tiny 
  \settowidth{bitlabelwidth}{\quad 9999} \\
  \rotatebox[origin=B]{90}{\makebox[bitlabelwidth][r]{#1}}}

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

\begin{tabular}{cc}
  \tiny A & \tiny B \\
\end{tabular}
**Unused bits**  The `bgcolor` option can be used to represent unused bits by specifying a background fill color—light gray looks nice—and empty text:

```latex
\definecolor{lightgray}{gray}{0.8}
\begin{bytefield}{32}
  \bitheader{0,4,8,12,16,20,24,28} \ \n  \bitbox{8}{Tag} & \bitbox{8}{Value} & \bitbox{4}{[bgcolor=lightgray]} & \bitbox{12}{Mask} \ \n  \wordbox{1}{Key}
\end{bytefield}
```

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>20</th>
<th>24</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Value</td>
<td></td>
<td>Mask</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Aligning text on the baseline**  Because `bytefield` internally uses \TeX’s `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:

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

```
M y
```

A solution is to use the `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:

```latex
\newlength{\maxheight}
\setlength{\maxheight}{\heightof{W}}
\newcommand{\baselinealign}[1]{\centering \raisebox{0pt}[\maxheight][0pt]{#1}}
\begin{bytefield}[boxformatting=\baselinealign, bitwidth=1.5em]{2}
  \bitbox{1}{M} & \bitbox{1}{y}
\end{bytefield}
```

20
Register contents  Sometimes, rather than listing the meaning of each bit field within each \bitbox or \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 \turnbox macro from the rotating package:

\newcommand{\bitlabel}[2]{%  
\bitbox{#1}{%  \raisebox{0pt}{\[4ex\][0pt]{%  \turnbox{45}{\fontsize{7}{7}\selectfont#2}  
\}}%  }
}

\begin{bytefield}[bitwidth=1em]{16}
\begin{tabular}{cccccccccccccc}
\bitlabel{}{Carry} & \bitlabel{}{Reserved} & \bitlabel{}{Parity} & \bitlabel{}{Reserved} & \bitlabel{}{Adjust} & \bitlabel{}{Reserved} & \bitlabel{}{Zero} & \bitlabel{}{Sign} & \bitlabel{}{Trap} & \bitlabel{}{Interrupt enable} & \bitlabel{}{Direction} & \bitlabel{}{Overflow} & \bitlabel{2}{I/O privilege level (12–13)} & \bitlabel{}{Nested task} & \bitlabel{}{Reserved}
\hline
\bitbox{}{0} & \bitbox{}{1} & \bitbox{}{0} & \bitbox{}{0} & \bitbox{}{0} & \bitbox{}{0} & \bitbox{}{0} & \bitbox{}{1} & \bitbox{}{0} & \bitbox{}{1} & \bitbox{}{0} & \bitbox{}{0} & \bitbox{}{0} & \bitbox{}{0} & \bitbox{}{0}
\end{tabular}
\end{bytefield}

\begin{center}
\begin{tabular}{cccccccccccccccc}
Carry & Reserved & Adjust & Reserved & Parity & Reserved & Zero & Sign & Trap & Interrupt enable & Direction & Overflow & I/O privilege level (12–13) & Nested task & Reserved
\hline
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{tabular}
\end{center}

2.5 Not-so-common tricks

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 \texttt{bytefield} is to point the \texttt{bitformatting} option to a macro that conditionally modifies the given bit number before outputting it. One catch is that \texttt{bytefield} measures the height of the string “1234567890” using the current bit formatting, so that needs to be a valid input. (If \texttt{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 \textit{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.

\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,
derianness=big]{16}
\bitheader{0-15} \\ 
\bitbox{1}{\tiny F/E} & \bitbox{1}{\tiny T0} & \bitbox{1}{\tiny T1} & \bitbox{1}{\tiny Fwd} & \bitbox{12}{Data value} \\
\end{bytefield}

\textbf{Memory-map diagrams}  While certainly not the intended purpose of the \texttt{bytefield} package, one can utilize word boxes with empty \texttt{\langle sides\rangle} and word labels to produce memory-map diagrams:
<table>
<thead>
<tr>
<th>Partition</th>
<th>Memory Range</th>
<th>Description</th>
<th>Accessible</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x003FFFFF - 0x00001000</td>
<td>4 MB area for MS-DOS and Windows 3.1 compatibility.</td>
<td>Read/writable.</td>
</tr>
<tr>
<td></td>
<td>0x00000FFF - 0x00000000</td>
<td>4096 byte area for MS-DOS and Windows 3.1 compatibility.</td>
<td>Protected --- catches null pointers.</td>
</tr>
<tr>
<td>2</td>
<td>0x7FFFFFFF - 0x00400000</td>
<td>~2 GB area private to process, process code, and data.</td>
<td>Read/writable.</td>
</tr>
<tr>
<td>3</td>
<td>0xBFFFFFFF - 0x80000000</td>
<td>1 GB area for memory-mapped files, shared system dlls, file system code; shared by all processes.</td>
<td>Read/writable.</td>
</tr>
<tr>
<td>4</td>
<td>0xFFFFFFFF - 0xC0000000</td>
<td>~1 GB area for VxDs, memory manager, file system code; shared by all processes.</td>
<td>Read/writable.</td>
</tr>
<tr>
<td>Address</td>
<td>Description</td>
<td>Partition</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFF</td>
<td>1 GB area for VxDs, memory manager, file system code; shared by all processes.</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>0xC0000000</td>
<td>Read/writable.</td>
<td></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>3</td>
<td></td>
</tr>
<tr>
<td>0x80000000</td>
<td>Read/writable.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x7FFFFFFF</td>
<td>~2 GB area private to process, process code, and data.</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0x00400000</td>
<td>Read/writable.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x003FFFFF</td>
<td>4 MB area for MS-DOS and Windows 3.1 compatibility.</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0x00001000</td>
<td>Read/writable.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000FFF</td>
<td>4096 byte area for MS-DOS and Windows 3.1 compatibility.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000000</td>
<td>Protected—catches NULL pointers.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

```latex
\newcommand{\memsection}{\parbox[c][3.8\baselineskip]{0.95\width}{%.facilitates the creation of memory maps. Start address at the bottom,\
end address at the top.}\
syntax: \memsection{end address}{start address}{height in lines}{text in box}\
newcommand{\memsection}[4]{%\
\bytefieldsetup{bitheight=#3\baselineskip}\
\bitbox{10}{%\
\texttt{#1}}% print end address\
\vspace{#3\baselineskip}\vspace{-2\baselineskip}\vspace{-#3pt}\
\texttt{#2}}% print start address}\
\bitbox{16}{#4} % print box with caption}
```

```latex
\begin{bytefield}{24}
\memsection{ffff ffff}{0040 0000}{15}{-- free --}\begin{rightwordgroup}{internal memory}\
\memsection{003f ffff}{002f c000}{4}{Special Function}
```

24
2.6 Putting it all together

The following code showcases most of bytefield’s features in a single figure.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data offset</td>
<td>Data offset for alignment</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved space for future needs</td>
</tr>
<tr>
<td>Urgent</td>
<td>Urgent flag for network control</td>
</tr>
<tr>
<td>ACK</td>
<td>Acknowledgment flag</td>
</tr>
<tr>
<td>PSH</td>
<td>Push flag for network control</td>
</tr>
<tr>
<td>RST</td>
<td>Reset flag for network control</td>
</tr>
<tr>
<td>SYN</td>
<td>Synchronize flag for network control</td>
</tr>
<tr>
<td>FIN</td>
<td>Finish flag for network control</td>
</tr>
<tr>
<td>Window</td>
<td>Window size for network control</td>
</tr>
<tr>
<td>Checksum</td>
<td>Checksum for data integrity check</td>
</tr>
<tr>
<td>Urgent pointer</td>
<td>Pointer for urgent data</td>
</tr>
<tr>
<td>Data octets</td>
<td>Data octets following the header</td>
</tr>
<tr>
<td>Source</td>
<td>Source address of the sender</td>
</tr>
<tr>
<td>Destination</td>
<td>Destination address of the receiver</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Timestamp for message lifetime</td>
</tr>
</tbody>
</table>

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.
These words were taken verbatim from the TCP header definition (RFC 793).

Why two Length fields? No particular reason.

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

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

64-bit random number

Figure 3: Complex protocol diagram drawn with the \texttt{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}{\langle width\rangle}}, use \texttt{\bytefieldsetup{bitwidth=\langle width\rangle}}.

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

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

Instead of using \texttt{\setlength{\curlyshrinkage}{\langle dist\rangle}} to reduce the vertical space occupied by a curly brace, use \texttt{\bytefieldsetup{curlyshrinkage=\langle dist\rangle}}. Note that, as described in Section 2.2, left and right curly-brace height can be reduced independently if desired.
The meaning of `\bitwidth`'s optional argument changed with `bytefield` v2.1. In older versions of the package, the optional argument was one of "l" 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 `bitformatting`, `endianness`, and `lsb`). Hence, "l" should be replaced with `endianness=little` and "b" should be replaced with `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 `\bytefieldsetup`.

As a crutch to help build older documents with minimal modification, `bytefield` provides a `compat1` package option that restores the old interface. This option, invoked with `\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 `\widthof` and `\heightof` were introduced in June 1998, `tex\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.

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

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` `\allocationnumber`

`\newdimen` defines new `(dimen)`s globally. `\bf@newdimen` defines them locally. It simply merges `\TeX` 2\'s `\newdimen` and `\alloc@` macros while omitting `\alloc@`'s "global" declaration.

3 `\def\bf@newdimen#1{\advance\count11 by 1}`
\count11 \allocationnumber = \count11
\dimendef#1 = \allocationnumber
wlog{\string#1 = \string\dimen\the\allocationnumber\space (locally)}\%
\bf@newdimen ε-TEX provides many more \textit{dimen}s than the original \TeX’s 255. When running newer versions of ε-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 ε-TEX we leave \texttt{\bf@newdimen} defined as above to help reduce register pressure when only 255 \textit{dimen}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
}
\bytefield@height
\ifcounting@words When \texttt{\ifcounting@words} is \texttt{TRUE}, add the height of the next 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
\ifcounting@words
\newlength{\bytefield@height@increment}
\DeclareRobustCommand{\inc@bytefield@height}[1]{%
\setlength{\bytefield@height@increment}{#1}%
\global\advance\bytefield@height by \bytefield@height@increment
}
\newsavebox{\entire@bytefield@picture}
\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 bit field, put a box around it, and do other operations on the entire figure.
The `bytefield` environment contains the layout of bits in a sequence of words. 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.)

```latex
\begin{bytefield}[2][12]
  1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12
\end{bytefield}
```

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`).

`\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 \texttt{\textbackslash{bf@boxformatting}} accepts an argument, the macro should be defined with \texttt{\textbackslash{long}} (e.g., with \texttt{\textbackslash{newcommand}} but not with \texttt{\textbackslash{newcommand}*}).

\texttt{\textbackslash{bf@call@box@cmd}}

\texttt{\textbackslash{bf@call@box@func}}

\texttt{\textbackslash{bitbox}}

\texttt{\textbackslash{bf@bitwidth}} 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{\textbackslash{setlength}} this larger.

\texttt{\textbackslash{bf@bitheight}} This is the height of a single bit within the bit field.

\texttt{\textbackslash{units@wide}} \texttt{\textbackslash{units@tall}} These are scratch variables for storing the width and height (in points) of the box we're about to draw.

\texttt{\textbackslash{bf@call@box@cmd@i}} Store the set of lines and the bit/word count and invoke \texttt{\textbackslash{bf@call@box@cmd@ii}}.

\texttt{\textbackslash{bf@call@box@cmd@ii}} Store the key/value parameters and the text to typeset then invoke the macro originally passed to \texttt{\textbackslash{bf@call@box@cmd}}.

\texttt{\textbackslash{bf@call@box@cmd@iii}} \texttt{\textbackslash{bf@call@box@cmd@iv}}

\texttt{\textbackslash{units@wide}} \texttt{\textbackslash{units@tall}}

\texttt{\textbackslash{bf@bitwidth}}\texttt{\textbackslash{bf@bitheight}}

\texttt{\textbackslash{units@wide}}\texttt{\textbackslash{units@tall}}

\texttt{\textbackslash{bf@call@box@cmd}}

\texttt{\textbackslash{bf@call@box@func}}

\texttt{\textbackslash{bitbox}} Put some text (\#4) 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). Uppercase letters provide the opposite effect: They suppress drawing the [L]eft, [R]ight, [T]op, and/or [B]ottom
sides. Additional drawing parameters can be provided via another optional argument (#3).

\DeclareRobustCommand{\bitbox}{\bf@call@box@cmd{\bf@bitbox}}

\bf@bitbox
Implement all of the \bitbox logic.

\def\bf@bitbox{%
\bgroup
\expandafter\bf@parse@bitbox@arg\expandafter{\bf@call@box@arg@i}%
\setlength{\units@wide}{\bf@bitwidth * \bf@call@box@arg@ii}%
\expandafter\bf@bytefieldsetup\expandafter{\bf@call@box@arg@iii}%
@ifundefined{bf@bgcolor}{%
If bgcolor was specified, draw a colored rule of the full size of the box.
\rlap{%
\draw@bit@picture{\strip@pt\units@wide}{\strip@pt\bf@bitheight}{%
\color{bf@bgcolor}%
\rule{\width}{\height}%
}%}
%}
Draw the user-provided text on top of the rule (if any).
\draw@bit@picture{\strip@pt\units@wide}{\strip@pt\bf@bitheight}{%
\bf@call@box@arg@iv%
}%
\egroup
\ignorespaces
}%}

\wordbox
Put some text (#4) 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 provide the opposite effect: They suppress drawing the [L]eft, [R]ight, [T]op, and/or [B]ottom sides. Additional drawing parameters can be provided via another optional argument (#3).

\DeclareRobustCommand{\wordbox}{\bf@call@box@cmd{\bf@wordbox}}

\bf@wordbox
Implement all of the \wordbox logic.

\def\bf@wordbox{%
\bgroup
\expandafter\bf@parse@bitbox@arg\expandafter{\bf@call@box@arg@i}%
\setlength{\units@wide}{\bf@bitwidth * \bits@wide}%
\setlength{\units@tall}{\bf@bitheight * \bf@call@box@arg@ii}%
\expandafter\bf@bytefieldsetup\expandafter{\bf@call@box@arg@iii}%
@ifundefined{bf@bgcolor}{%
If bgcolor was specified, draw a colored rule of the full size of the box.
\rlap{%
\draw@bit@picture{\strip@pt\units@wide}{\strip@pt\bf@bitheight}{%
\bf@call@box@arg@iv%
\draw@bit@picture \bf@call@box@arg@iv
\@ifundefined{bf@per@word}{}{\bf@invoke@per@word{\bf@call@box@arg@ii}}
\egroup
\ignorespaces
} \draw@bit@picture
\put(0,0){\makebox(#1,#2){\parbox{#1\unitlength}{\bf@set@user@dimens{#1}{#2}\bf@boxformatting{#3\par}}}}
\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

\bf@invoke@per@word
Invoke \bf@per@word once per word, passing it the (0-indexed) word number and total number of words.

\newcommand{\bf@invoke@per@word}[1]{%
  \begin{picture}(0,0)%
  \@tempcnta=0
  \@tempdima=#1\bf@bitheight
  \loop
    \advance\@tempdima by -\bf@bitheight
    \bgroup
      \put(-\strip@pt\units@wide, \strip@pt\@tempdima){%\advance@tempcnta by 1\relax
  \egroup
  \ifnum#1>\@tempcnta
    \repeat
  \end{picture}%
}

\bf@set@user@dimens
Given a width in bits (#1) and a height in words (#2), make a number of box dimensions available to the author: \width, \height, \depth, \totalheight. Additionally, make the arguments available to the author via the \widthunits and \heightunits macros.

\newcommand{\bf@set@user@dimens}[2]{%
  \bf@newdimen\width
  \bf@newdimen\height
  \bf@newdimen\depth
  \bf@newdimen\totalheight
  \width=#1\unitlength
  \height=#2\unitlength
  \depth=0pt
  \totalheight=#2\unitlength
  \def\widthunits{#1}
  \def\heightunits{#2}
}
\bitboxes Put each token in \#3 into 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). 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@call@box@cmd{\bf@bitboxes@star}}%
{\bf@call@box@cmd{\bf@bitboxes@no@star}}%
}
\bf@relax
Define a macro that expands to \relax for use with \ifx tests against \bf@bitboxes@arg, which can contain either tokens to typeset or \relax.
\def\bf@relax{\relax}
\bf@bitboxes@no@star
Implement the unstarred version of \bitboxes. This macro simply expands its text argument into a list of tokens followed by \relax then invokes \bf@bitboxes@no@star@i.
\def\bf@bitboxes@no@star{\bf@bitboxes@no@star@i\bf@call@box@arg@iv\relax}
\bf@bitboxes@no@star@i
Walk the subsequent tokens one-by-one until \relax is encountered. For each token, invoke \bf@bitbox (the internal version of \bitbox for which \bf@call@box@arg@i\texttt{(number)} are all defined.
\def\bf@bitboxes@no@star@i#1{\def\bf@call@box@arg@i{#1}\ifx\bf@call@box@arg@i\bf@relax \let\next=\relax \else \bf@bitbox \let\next=\bf@bitboxes@no@star@i \fi}
\bf@bitboxes@star
Implement the starred version of \bitboxes. This macro simply stores the original \texttt{(sides)} argument in \bf@bitboxes@sides, expands its text argument into a list of tokens followed by two \relaxes, and invokes \bf@bitboxes@star@i.
\def\bf@bitboxes@star{\bf@bitboxes@sides{\bf@call@box@arg@i}\bf@call@box@arg@i\bf@call@box@arg@i\relax\relax\relax\ignorespaces}
\bf@bitboxes@sides

36
Process the first token in the text argument passed to `\bitboxes*`. If it’s also the last token (indicated by its being followed by `\relax`), draw an ordinary bit box with all sides present. If it’s not the last token, draw a bit box with the right side suppressed and invoke `\bitboxes@star@ii` on the remaining tokens.

```
\bf@bitboxes@star@i
\bf@call@box@arg@iv
\bf@bitboxes@arg@ii
\next
\def\bf@bitboxes@star@i#1#2{% 
  \def\bf@call@box@arg@iv{#1}% 
  \def\bf@bitboxes@arg@ii{#2}% 
  \ifx\bf@bitboxes@arg@ii\bf@relax 
    \bf@bitbox 
    \let\next=\relax 
  \else 
    \edef\bf@call@box@arg@i{\bf@bitboxes@sides R}% 
    \bf@bitbox 
    \def\next{\bf@bitboxes@star@ii{#2}}% 
  \fi 
  \next 
}
```

Process the second and subsequent tokens in the text argument passed to `\bitboxes*`. If the next token in the stream is the final one (indicated by its being followed by `\relax`), draw a bit box with the left side suppressed. If it’s not the final token, draw a bit box with both the left and right sides suppressed and invoke itself recursively on the remaining tokens.

```
\bf@bitboxes@star@ii
\bf@call@box@arg@iv
\bf@bitboxes@arg@ii
\next
\def\bf@bitboxes@star@ii#1#2{% 
  \def\bf@call@box@arg@iv{#1}% 
  \def\bf@bitboxes@arg@ii{#2}% 
  \ifx\bf@bitboxes@arg@ii\bf@relax 
    \edef\bf@call@box@arg@i{\bf@bitboxes@sides L}% 
    \else 
    \edef\bf@call@box@arg@i{\bf@bitboxes@sides LR}% 
    \fi 
    \ifx\bf@call@box@arg@iv\bf@relax 
    \let\next=\relax 
  \else 
    \bf@bitbox 
    \def\next{\bf@bitboxes@star@ii{#2}}% 
  \fi 
  \next 
}
```

### 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.
These macros are set to \texttt{true} if we’re to draw the corresponding edge on the subsequent \texttt{bitbox} or \texttt{wordbox}.

\begin{verbatim}
\def\bf@parse@bitbox@arg#1{%
  \bitbox@topfalse\bitbox@bottomfalse\bitbox@leftfalse\bitbox@rightfalse
  \bf@parse@bitbox@sides#1X%}
\end{verbatim}

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

\begin{verbatim}
\def\bf@parse@bitbox@sides#1{%
  \ifx#1X%
  \else
    \ifx#1t%
      \bitbox@toptrue
    \else
      \ifx#1b%
        \bitbox@bottomtrue
      \else
        \ifx#1l%
          \bitbox@lefttrue
        \else
          \ifx#1r%
            \bitbox@righttrue
          \else
            \PackageWarning{bytefield}{Unrecognized box side ‘#1’}%
          \fi
        \fi
      \fi
    \fi
  \fi
\end{verbatim}

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

\begin{verbatim}
\def\bf@parse@bitbox@sides#1{%
  \ifx#1t%
    \bitbox@toptrue
  \else
    \ifx#1b%
      \bitbox@bottomtrue
    \else
      \ifx#1l%
        \bitbox@lefttrue
      \else
        \ifx#1r%
          \bitbox@righttrue
        \else
          \PackageWarning{bytefield}{Unrecognized box side ‘#1’}%
        \fi
      \fi
    \fi
  \fi
\end{verbatim}
3.5 Skipped words

\texttt{\textbackslash units@high}\hspace{1em}This is the height of each diagonal line in the \texttt{\skippedwords} graphic. Note that \texttt{\units@high = \units@tall} \textit{optional argument to \skippedwords}.

\texttt{\newlength{\units@high}}

\texttt{\skippedwords}\hspace{1em}Output a fancy graphic representing skipped words. The optional argument is the vertical space between the two diagonal lines (default: 2ex).

\texttt{\DeclareRobustCommand{\skippedwords}[1][2ex]{%}
\texttt{\setlength{\units@wide}{\bf@bitwidth * \bits@wide}}
\texttt{\setlength{\units@high}{1pt * \ratio{\units@wide}{6.0pt}}%}
\texttt{\setlength{\units@tall}{#1 + \units@high}}\%}
\texttt{\edef\num@wide{\strip@pt\units@wide} \%}
\texttt{\edef\num@tall{\strip@pt\units@tall} \%}
\texttt{\edef\num@high{\strip@pt\units@high} \%}
\texttt{\begin{picture}(\num@wide,\num@tall)}
\texttt{\put(0,\num@tall){\line(6,-1){\num@wide}}\%}
\texttt{\put(\num@wide,0){\line(-6,1){\num@wide}}\%}
\texttt{\put(0,0){\line(0,1){\num@high}}\%}
\texttt{\put(\num@wide,\num@tall){\line(0,-1){\num@high}}\%}
\texttt{\end{picture}}\%
\texttt{\ifcounting@words\%}
\texttt{\inc@bytefield@height{\unitlength * \real{\num@tall}}}\%
\texttt{\global\counting@wordsfalse} \%
\texttt{\fi} \%
\texttt{}}

3.6 Bit-position labels

\texttt{\bf@bit@endianness} \texttt{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 \texttt{\bf@bit@endianness} macro specifies which to use, either “l” for little-endian (the default) or “b” for big-endian.

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

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

\texttt{\newcommand{\bf@first@bit}{0}}
\bitheader

Output a header of numbered bit positions. The optional argument \( (#1) \) is “1” 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 \texttt{lsb} option to reverse the labels’ direction.

\begin{verbatim}
\DeclareRobustCommand{\bitheader}[2][]{\bf@parse@bitbox@arg{lrtb}\setlength{\units@wide}{\bf@bitwidth * \bits@wide}\setlength{\units@tall}{\heightof{\bf@bitformatting{1234567890}}}\setlength{\units@high}{\units@tall * -1}\bf@process@bitheader@opts{#1}\begin{picture}(\strip@pt\units@wide,\strip@pt\units@tall)(0,\strip@pt\units@high)\bf@parse@range@list#2,X,\end{picture}\ifcounting@words\inc@bytefield@height{\unitlength * \real{\strip@pt\units@tall}}\global\counting@wordsfalse\fi} \bf@parse@range@list

This is helper function \#1 for \bitheader. It parses a comma-separated list of ranges, calling \bf@parse@range on each range.
\end{verbatim}

\bf@parse@range@list

\def{\bf@parse@range@list}{\bf@parse@range{#1,\relax}{\relax}}

\bf@parse@range

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

\begin{verbatim}
\def{\bf@parse@range}{\setcounter{header@val}{#1}\setcounter{max@header@val}{#2 + 1}\loop\ifnum\value{header@val}<\value{max@header@val}\bf@parse@range@list{#1-#2-\relax}{\relax}\expandafter\bf@parse@range@list\fi} \header@xpos \texttt{x} position of header, current label to output, and maximum label to output (+1).
\end{verbatim}
This is helper function #3 for \bitheader. It processes the optional argument to \bitheader.

For backwards compatibility we also accept the (now deprecated) l as a synonym for endianness=little and b as a synonym for endianness=big. A typical document will specify an endianness option not as an argument to \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.

This warning occurred

This warning occurred

This warning occurred

This warning occurred

This warning occurred

This warning occurred
3.7 Word labels

3.7.1 Curly-brace manipulation

Reduce the height of a left (right) curly brace by \texttt{\@leftcurlyshrinkage} (\texttt{\@rightcurlyshrinkage}) 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, \texttt{\@leftcurlyshrinkage} and \texttt{\@rightcurlyshrinkage} can be modified.

\begin{verbatim}
def\bf@leftcurlyshrinkage{5pt}
def\bf@rightcurlyshrinkage{5pt}
\end{verbatim}

Define the amount of space to insert before a curly brace and before a word label (i.e., after a curly brace).

\begin{verbatim}
def\bf@leftcurlyspace{1ex}
def\bf@rightcurlyspace{1ex}
def\bf@leftlabelspace{0.5ex}
def\bf@rightlabelspace{0.5ex}
\end{verbatim}

Define the symbols to use as left and right curly braces. These symbols must be extensible math symbols (i.e., they will immediately follow \texttt{\left} or \texttt{\right} in math mode).

\begin{verbatim}
let\bf@leftcurly=\{
let\bf@rightcurly=\}
\end{verbatim}

Define the default formatting for left and right curly braces as “do nothing special”.

\begin{verbatim}
let\bf@leftcurlystyle=\relax
let\bf@rightcurlystyle=\relax
\end{verbatim}

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

\begin{verbatim}
newbox{\curly@box}
\end{verbatim}

Store a “}” that’s \#2 tall in box \#1. The only unintuitive thing here is that we have to redefine \texttt{\fontdimen22—the 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.

\begin{verbatim}
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}{\half@curly@height + 0.5\curly@shift}\global\setbox{#1}{\raisebox{\curly@shift}{}}\xdef\old@axis{\the\fontdimen22\textfont2}}\endgroup
\end{verbatim}
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}{\half@curly@height + 0.5\curly@shift}%
\global\sbox{#1}{\raisebox{\curly@shift}{%
$\xdef\old@axis{\the\fontdimen22\textfont2}$%
\bf@leftcurlystyle{%
$\fontdimen22\textfont2=0pt$
\left\bf@leftcurly
\vrule height\half@curly@height
width 0pt
depth\half@curly@height\right\bf@leftcurly$%
}\fontdimen22\textfont2=\old@axis$}}%
\endgroup
}

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 \end{rightwordgroup} defines it to output the word label and then reset itself to a no-op.

\def\show@wordlabelr{}

\wordlabelr@start
\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 first, optional, argument is a list of parameters, as defined in Section 2.3. The second,
mandatory, 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}{\[2\%}{\}

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).

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

\global\wordlabelr@start=\bytefield@height
\gdef\wordlabelr@params{#1}%
\gdef\wordlabelr@text{#2}%
\ignorespaces
%
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
\show@wordlabelr
Redefine \show@wordlabelr to output \bf@rightcurlyspace space, followed by
a large curly brace (in \curlybox), followed by \bf@rightlabelspace space, fol-
lowed by the user’s text (previously recorded in \wordlabelr@text). We typeset
\wordlabelr@text within a tabular environment, so \LaTeX will calculate its
width automatically.

\gdef\show@wordlabelr{%
\sbox{\word@label@box}{%
\begin{tabular}[b]{@{}l@{}}\wordlabelr@text\end{tabular}%
}%
\settowidth{\label@box@width}{\usebox{\word@label@box}}%
\setlength{\label@box@height}{\wordlabelr@end-\wordlabelr@start}%
Evaluate any parameters passed to \begin{rightwordgroup} right before we ren-
der the curly brace.

\expandafter{\bf@bytefieldsetup}\expandafter{\wordlabelr@params}%
\store@curly{\curlybox}{\label@box@width}%
\bf@newdimen{\total@box@width}%
\setlength{\total@box@width}{%}
\bf@rightcurlyspace +
\widthof{\usebox{\curlybox}} +
\bf@rightlabelspace +
\label@box@width%
}%
\begin{picture}(\strip@pt{\total@box@width},0)
\put(0,0){%
\hspace{\bf@rightcurlyspace}%
}
The last thing \show@wordlabelr does is redefine itself back to a no-op.
\gdef\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}
\ignorespacesafterend
}

3.7.3 Left-side labels

\wordlabell@start Declare the starting and ending height (in points) of the set of rows to be labeled on the left.
\wordlabell@end
\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 \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. \end{leftwordgroup} outputs the label immediately. Even though
\end{leftwordgroup} 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
will eventually be enough space to accomodate the label, we know that the label
won’t overlap the bit field or extend beyond the bit-field boundaries.

We store the starting height, optional parameters (see Section 2.3), and label text,
all of which are needed by the \end{leftwordgroup}. We immediately parse the
parameters because they may affect the \store@lcurly invocation below.

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 \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 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.

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.

We now end the group that \begin{rightwordgroup} created. This lets the
leftwordgroup environment span rows (because we’re technically no longer within
the environment).
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. Before typesetting the curly brace we re-parse the optional parameters because we’re in a new group from the one in which we parsed them before, and the parameters can affect the second \store@lcurly invocation just they could have affected the first.

\global\wordlabell@end=\bytefield@height
\bf@newdimen\starting@point
\setlength{\starting@point}{\-\total@lbox@width - \bf@bitwidth*\bits@wide}
\sbox{\word@label@box}{\begin{tabular}[b]{@{}l@{}}\wordlabell@text\end{tabular}}
\settowidth{\label@box@width}{\usebox{\word@label@box}}
\setlength{\label@box@height}{\wordlabell@end-\wordlabell@start}
\expandafter\bf@bytefieldsetup\expandafter{\wordlabell@params}
\store@lcurly{\curly@box}{\label@box@height}
\begin{picture}(0,0)
\put(\strip@pt\starting@point,0){\makebox(\strip@pt\label@box@width,\strip@pt\label@box@height){\usebox{\word@label@box}}\hspace*{\bf@leftlabelspace}\usebox{\curly@box}\hspace*{\bf@leftcurlyspace}}\end{picture}

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

\def\@currenvir{leftwordgroup}%
\ignorespacesafterend}

\@currenvir

3.7.4 Scratch space

\label@box@width\label@box@height\word@label@box

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

Define a handful of lengths that the user was allowed to \setlength explicitly in bytefield 1.x.

\setlength{\bitwidth}{\tiny 99i}\%  
\setlength{\byteheight}{4ex}\%  
\setlength{\curlyspace}{1ex}\%  
\setlength{\labelspace}{0.5ex}\%  
\setlength{\curlyshrinkage}{5pt}\%

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.

\begin{newbytefield}[\bitwidth,\byteheight,\curlyspace,\labelspace,\curlyshrinkage]  
\end{newbytefield}

Define \wordgroupr, \endwordgroupr, \wordgroupl, and \endwordgroupl in terms of the new rightwordgroup and leftwordgroup environments.
\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.)

538 \renewcommand{\bytefieldsetup}[1]{%
539 \PackageError{bytefield}{% 540 The \protect\bytefieldsetup\space macro is not available in\MessageBreak 541 version 1 compatibility mode% 542 }{% 543 Remove [compat1] from the \protect\usepackage{bytefield} line to 544 make \protect\bytefieldsetup\MessageBreak 545 available to this document.\space\space (The document may also need 546 to be modified to use\MessageBreak 547 the new bytefield interface.)% 548 }%
549 }%
550 }

\wordgroupr Issue a helpful error message for the commands that were removed in bytefield v2.0.
\endwordgroupr 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.

551 \newcommand{\wordgroupr}{%
552 \PackageError{bytefield}{% 553 Macros \protect\wordgroupr, \protect\wordgroupl, \protect\endwordgroupr, 554 \MessageBreak 555 and \protect\endwordgroupl\space no longer exist% 556 }%
557 }% 558 Starting with version 2.0, bytefield uses \protect\begin{wordgroupr}... 559 \MessageBreak 560 \protect\end{wordgroupr} and \protect\begin{wordgroupl}...% 561 \protect\end{wordgroupl}\MessageBreak 562 to specify word groups and a new \protect\bytefieldsetup\space macro to 563 \MessageBreak 564 change bytefield’s various formatting parameters.% 565 }% 566 }% 567 \let\endwordgroupr=\wordgroupr 568 \let\wordgroupl=\wordgroupr 569 \let\endwordgroupl=\wordgroupr

3.9 Option processing

We use the 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 \bytefieldsetup macro.

\KV@bytefield@bitwidth Specify the width of a bit number in the bit header. If the special value “auto” is given, set the width to the width of a formatted “991”.

49
\define@key{bytefield}{bitwidth}{\%}
\def\bf@bw@arg{#1}\%
\def\bf@auto{auto}\%
\ifx\bf@bw@arg\bf@auto\%
\settowidth{\bf@bitwidth}{\bf@bitformatting{99i}}\%
\else\%
\setlength{\bf@bitwidth}{#1}\%
\fi\%
\fi\%

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

\KV@bytefield@bitformatting
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}).
\define@key{bytefield}{bitformatting}{\def\bf@bitformatting{#1}}

\KV@bytefield@boxformatting
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}).
\define@key{bytefield}{boxformatting}{\def\bf@boxformatting{#1}}

\KV@bytefield@leftcurly
\KV@bytefield@rightcurly
\bf@leftcurly
\bf@rightcurly
Specify the symbol to use for bracketing a left or right word group. This must be an extensible math delimiter (i.e., something that can immediately follow \left or \right in math mode).
\define@key{bytefield}{leftcurly}{\def\bf@leftcurly{#1}}
\define@key{bytefield}{rightcurly}{\def\bf@rightcurly{#1}}

\KV@bytefield@leftcurlyspace
\KV@bytefield@rightcurlyspace
\bf@leftcurlyspace
\bf@rightcurlyspace
Specify the amount of space between the bit fields in a word group and the adjacent left or right curly brace. The \curlyspace option is a shortcut that puts the same space before both left and right curly braces.
\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}\%
\def\bf@leftcurlyspace{#1}%
\def\bf@rightcurlyspace{#1}%
\def\bf@leftcurlyspace{#1}}
\define@key{bytefield}{rightcurlyspace}{\def\bf@rightcurlyspace{#1}}
\define@key{bytefield}{curlyspace}{\def\bf@leftcurlyspace{#1}\%
\def\bf@rightcurlyspace{#1}\%
\def\bf@leftcurlyspace{#1}}
\define@key{bytefield}{rightcurlyspace}{\def\bf@rightcurlyspace{#1}}
\define@key{bytefield}{curlyspace}{\def\bf@leftcurlyspace{#1}\%
\def\bf@rightcurlyspace{#1}\%
\def\bf@leftcurlyspace{#1}}
\define@key{bytefield}{rightcurlyspace}{\def\bf@rightcurlyspace{#1}}

\KV@bytefield@leftlabelspace
\KV@bytefield@rightlabelspace
\bf@leftlabelspace
\bf@rightlabelspace
Specify the amount of space between a left or right word group's curly brace and the associated label text. The \labelspace option is a shortcut that puts the same space after both left and right curly braces.
\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}\%
\def\bf@leftlabelspace{#1}\%
\def\bf@rightlabelspace{#1}\%
Specify the number of points by which to reduce the height of a curly brace (left, right, or both) so its ends don’t overlap whatever’s above or below it.

\def\bf@leftcurlyshrinkage{#1}
\def\bf@rightcurlyshrinkage{#1}
\def\bf@curlyshrinkage{\def\bf@leftcurlyshrinkage{#1}\def\bf@rightcurlyshrinkage{#1}}

Specify a macro that takes either zero or one argument and that precedes the text that draws a left curly brace, right curly brace, or either curly brace.

\def\bf@leftcurlystyle{#1}
\def\bf@rightcurlystyle{#1}
\def\bf@curlystyle{\def\bf@leftcurlystyle{#1}\def\bf@rightcurlystyle{#1}}

Set the default endianness to either little endian or big endian.

\def\bf@endianness{#1}
\newcommand{\bf@parse@endianness}[1]{\def\bf@little{little}\def\bf@big{big}\def\bf@arg{#1}\ifx\bf@arg\bf@little\def\bf@bit@endianness{l}\else\ifx\bf@arg\bf@big\def\bf@bit@endianness{b}\else\PackageError{bytefield}{Invalid argument "#1" to the endianness option}{The endianness option must be set to either "little" or "big".\MessageBreak Please specify either endianness=little or endianness=big.}\fi\fi}

Specify a numerical value for the least significant bit of a word.

\def\bf@lsb{#1}

Specify a background color for a bit box or word box.

\def\bf@bgcolor{#1}

Specify a macro to invoke for each word of a word box. The macro must take two arguments: the word number (0-indexed) and the total number of words.
Reconfigure values for various `bytefield` parameters. Internally to the package we use the `\bf@bytefieldsetup` macro instead of `\bytefieldsetup`. This enables us to redefine `\bytefieldsetup` when entering version 1 compatibility mode without impacting the rest of `bytefield`.

We define only a single option that can be used only as a package option, not as an argument to `\bytefieldsetup`: `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.

We want to use `\bf@bytefieldsetup` to process `bytefield` package options. Unfortunately, `\DeclareOption` doesn’t handle `⟨key⟩=⟨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.

We define 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 `⟨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:
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
greatly reduce the likelihood of
“No room for a new \dimen”
errors (reported by Vitaly A.
Repin) ................. 29

\bf@parse@range@list: Bug fix:
Swapped order of arguments to
\ifx test (suggested by
Hans-Joachim Widmaier) .... 40

General: Restructured the .dtx file

v1.2

\curly@box: Bug fix: Defined
\curly@box globally (suggested by Stefan Ulrich) ........... 42

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. ............... 29

v1.3

\bf@newdimen: Added support for
\e-\TeX’s larger local (\dimen)
pool (code provided by Heiko
Oberdiek) ................. 30

v1.4

\bf@bitformatting: Introduced
this macro at Steven R. King’s
request to enable users to alter
the bit header’s font size ....... 31

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) ............. 1

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 ....... 52

General: Made a number of
non-backwards-compatible
changes, including replacing
\wordgroupr and
\endwordgroupr with a
rightwordgroup environment
and \wordgroupl and
\endwordgroupl with a
leftwordgroup environment
and also replacing a slew of
user-visible lengths and macros
with a single \bytefieldsetup
macro ................. 1

53
v2.1
\\: Augmented the definition of \\ to accept an optional argument, just like in a tabular environment . . . . . . \textbf{31}
\@parseorange: Added code due to Renaud Pacalet for shifting the bit header by a distance corresponding to \@firstbit, used for typesetting registers split across rows . . . . . . \textbf{40}
\bitheader: Changed the optional argument to accept \langle key \rangle = \langle value \rangle pairs instead of just “1” and “b” . . . . . . \textbf{40}
General: Included in the documentation a variable-height memory-map example suggested by Martin Demling . . . . . . \textbf{24}

v2.2
\bitboxes: Added this macro based on an idea proposed by Andrew Mertz . . . . . . \textbf{36}

v2.3
\@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 (see \texttt{http://tex.stackexchange.com/q/275042}) . . . . . . \textbf{30}

v2.4
bytefield: Make the code resilient to changes in \baselinestretch. Thanks to Karst Koymans for the bug report . . . . . . \textbf{31}

v2.5
General: Accept a new \texttt{bgcolor} option to set a bit’s background color . . . . . . \textbf{1}
Accept a new \texttt{perword} option to execute a macro for each word in a word box. This addresses a feature request by Victor Toni . \textbf{1}
Redefine the \texttt{bitbox}, \texttt{wordbox}, and \texttt{bitboxes} commands additionally to accept key/value options . . . . . . \textbf{1}

v2.6
General: Accept new \texttt{curlystyle}, \texttt{leftcurlystyle}, and \texttt{rightcurlystyle} options to control the styling (e.g., color) of curly braces. This addresses a feature request by Victor Toni . \textbf{1}
\leftwordgroup: Accept key/value options . . . . . . \textbf{46}
Suppress spaces following the \texttt{\leftwordgroup} . . . . . . \textbf{46}
\rightwordgroup: Accept key/value options . . . . . . \textbf{44}
Suppress spaces following the \texttt{\rightwordgroup} . . . . . . \textbf{44}

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>Code Line 1</th>
<th>Code Line 2</th>
<th>Code Line 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@currentvir</td>
<td>150</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>@ifstar</td>
<td>171</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@ifundefined</td>
<td>80</td>
<td>102</td>
<td>114</td>
</tr>
<tr>
<td>@tempcnta</td>
<td>143</td>
<td>150</td>
<td>153</td>
</tr>
</tbody>
</table>
L
\label@box@height \ldots 435, 437, 450, 493, 498, 507
\label@box@width \ldots 433, 450, 170, 176, 192, 198, 307
\labelspace \ldots 28, 511, 520
leftcurly (option) \ldots 10, 14
leftcurlyshrinkage (option) \ldots 14
leftcurlyspace (option) \ldots 13, 14
leftcurlystyle (option) \ldots 7, 14
leftwordgroup (environment) \ldots 7, 13
\line \ldots 127, 130, 133, 283, 284, 285, 286
\locdimen \ldots 13
lsb (option) \ldots 15, 18, 20

M
\make@1space \ldots 44, 208, 478
\makebox \ldots 120, 353, 450, 498
\max@header@0val \ldots 318

N
\newbytefield \ldots 522
\newdimen \ldots 16
\next \ldots 192, 268, 634
\num@high \ldots 281, 285, 286
\num@tall \ldots 280, 282, 285, 286, 389
\num@wide \ldots 270, 282, 283, 284, 286

O
\old@axis \ldots 379, 407, 415
\old@nl \ldots 27
\openup \ldots 35
options:
bgcolor \ldots 4, 12, 20, 33, 54
bitformatting \ldots 10, 73, 19
bitheight \ldots 10
bitwidth \ldots 10, 11, 22, 28
boxformatting \ldots 10, 73, 20
compat1 \ldots 10, 73, 41
curlyshrinkage \ldots 10, 73, 41
curlyspace \ldots 10, 73, 41
curlystyle \ldots 7, 14, 74
dianianness \ldots 6, 10
leftcurly \ldots 10, 14, 11
leftcurlyshrinkage \ldots 10, 14, 11
leftcurlyspace \ldots 10, 14, 11
leftcurlystyle \ldots 7, 14, 74
lsb \ldots 15, 18, 46
perword \ldots 15, 18, 46
rightcurly \ldots 10, 18, 14
rightcurlyshrinkage \ldots 10, 18, 14
rightcurlyspace \ldots 10, 18, 14
rightcurlystyle \ldots 7, 14, 74

P
\PackageError \ldots 539, 552, 618
\PackageInfo \ldots 511
packages:
bytefield \ldots 29, 29, 30, 34, 41, 44, 44, 52, 52, 59
calc \ldots 30, 53
color \ldots 12
etex \ldots 30
graphix \ldots 19
register \ldots 29
rotating \ldots 29
xcolor \ldots 75
\PackageWarning \ldots 262, 347, 357
\parbox \ldots 120
perword (option) \ldots 16, 54
\ProcessOptions \ldots 611
\put \ldots 120, 124, 127, 130, 133, 149
\setkeys \ldots 283, 285, 288, 286, 322, 446, 497
\textfont \ldots 386, 388, 407, 409, 415

R
register (package) \ldots 21
\RequirePackage \ldots 12
rightcurly (option) \ldots 10, 18, 14
rightcurlyshrinkage (option) \ldots 14
rightcurlyspace (option) \ldots 13, 14
rightcurlystyle (option) \ldots 7, 14, 74
rightwordgroup (environment) \ldots 7, 124
\rlap \ldots 82, 104
\rule \ldots 82, 104

S
\selectfont \ldots 30
\setkeys \ldots 343, 631
\show@wordlabelr \ldots 43, 48, 115, 330
\skippedwords \ldots 8, 276
\starting@point \ldots 487, 488, 497
\store@lcurly \ldots 307, 471, 495
\store@rcurly \ldots 376, 437
\strip@pt \ldots 83, 89, 105, 111, 145
\textfont \ldots 386, 388, 394, 407, 409, 415

T
<table>
<thead>
<tr>
<th>\text{theheader@val}</th>
<th>\tiny</th>
<th>\text{total@box@width}</th>
<th>\text{total@box@width}</th>
<th>\text{totalheight}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{\theheader@val}</td>
<td>\text{\tiny}</td>
<td>\text{438}</td>
<td>\text{439}</td>
<td>\text{445}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>\text{U}</th>
<th>\text{\unitlength}</th>
<th>\text{\units@high}</th>
<th>\text{\units@tain}</th>
<th>\text{\units@wide}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{120}</td>
<td>\text{137}</td>
<td>\text{163}</td>
<td>\text{164}</td>
<td>\text{166}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>\text{V}</th>
<th>\text{\vrule}</th>
<th>\text{\width}</th>
<th>\text{\widthunits}</th>
<th>\text{\wordlabel\box}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{390}</td>
<td>\text{411}</td>
<td>\text{85}</td>
<td>\text{107}</td>
<td>\text{158}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>\text{X}</th>
<th>\text{xcolor} (package)</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{12}</td>
<td></td>
</tr>
</tbody>
</table>