

hexdumptikz – Printing and annotating hexdumps with TikZ

Lukas Heindl*

v1.0.1 from 2026-06-20

Abstract

Showing hexdumps¹ is a common task in some fields. The most rudimentary form doing this in \LaTeX is using a simple listings package (like `listings`^{→CTAN}). But often just showing the data is not enough. Instead, it needs to be annotated and explained.

This package leverages `tikz`^{→CTAN} for printing the hexdumps. This way, it is very easy to highlight and annotate parts of the hexdump afterwards. But be careful, for large dumps, the *TikZ* drawing alone can take some time.

One key aspect of this package is that it tries to avoid calculating on the addresses as much as possible. In the cases where it is unavoidable, the package calculates just with the last digits of the addresses. This way the package tries to avoid issues where the address/offset gets too large² to represent it as an integer in \LaTeX .

Keep in mind the drawing happens with *TikZ* and every byte gets its own node. Thus, it may take a while to show larger hexdumps³.

1 General remarks on the documentation

Note this documentation makes use of *EBNF*⁴ for specifying valid input of some parsers. For typesetting these, the documentation uses `naive-ebnf`^{→CTAN} which is not 100 % in line with *ISO/IEC 14977*. So if in doubt, better check that package's documentation (1st page) for the notation.

2 Usage

```
\begin{hexdumptikz}[\textcolor{green}{\langle options \rangle}]{\langle filename/path \rangle}
\langle environment content \rangle
\end{hexdumptikz}
```

The environment which scopes working with a single hexdump. The commands `\hexdumptikzPrint`^{→P. 2} and `\hexdumptikzLabelBytes`^{→P. 2} are intended to be only used within this environment.

*E-Mail: oss.heindl+latex@protonmail.com

Issue tracker: codeberg.org/atticus-sullivan/hexdumptikz/issues

Codeberg (mirror): codeberg.org/atticus-sullivan/hexdumptikz

¹https://en.wikipedia.org/wiki/Hex_dump

²This happens quickly if hexdump is the dump of a processes the stack on a 64-bit machine

³According to my measurements it really is *TikZ* which is responsible for the slowdown and not the selection/styling logic or the parser

⁴https://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_form

Specify the path to the / the filename of the file with the hexdump in it as `{\filename/path}`. `[<options>]` allows to set keys from the `/hexdumptikz` space (see section 3, searches also `/tikz`) for the whole scope/group.

This still is inside a TikZ environment (it just opens a new TikZ-scope), so you may use arbitrary additional TikZ commands inside the environment.

`\hexdumptikzPrint[<options>]{<selector>}`

Read/Parse the hexdump as far as needed to fulfill the `{<selector>}` and generate the TikZ nodes to show the dump. See section 2.3 for how to specify the selector.

Note for this purpose, the selectors must *not overlap* and be sorted in the order of occurrence (thus usually it does not make sense to mix address and index selectors). Also here it does not make sense to specify a *style* (but it is not an error).

`[<options>]` allows to set keys from the `/hexdumptikz` space (see section 3, searches also `/tikz`). Common options here are `/hexdumptikz/tikz`^{P.5}, `/hexdumptikz/show*`, `/hexdumptikz/styled bytes`^{P.6} and redefining styles automatically used when drawing. You can also set options regarding the parsing here, but usually it makes more sense to specify these for the whole environment.

`\hexdumptikzLabelBytes[<options>]{<start-addr>}{<end-addr>}`

Label/Highlight a sequence of bytes in the hexdump. It will draw a path around the selected bytes (a rectangle by default).

The sequence of bytes is specified by `{<start-addr>}` and `{<end-addr>}`. It works across multiple lines and can jump over gaps and skipped offsets. Note you can only specify based on addresses, not based on indices.

`[<options>]` is evaluated in the `/hexdumptikz` space (see section 3, searches also `/tikz`) and allows customizing how the path is drawn.

Note this command relies on information gathered during `\hexdumptikzPrint` and thus needs to be executed after that command. Also `hexdumptikz`^{P.1} cleans up its state in the end, so `\hexdumptikzLabelBytes` must be executed within that environment.

`\hexdumptikzLabelBytesFallback[<options>]{<start>}{<end>}`

Basically the same as `\hexdumptikzLabelBytes` but does not rely on information gathered during `\hexdumptikzPrint`. Thus, it can also be used outside of `hexdumptikz`^{P.1} but the line does not perfectly fit the right row in the hexdump. Section 2.1 shows an example for the difference.

2.1 Simple Example

file: dat.hex

```
00      00 00 00 00 00 00 00 00
08      00 00 00 00 00 00 00 00
10      00 00 00 00 00 00 00 00
7fffffff00 58 ee ff ff ff 7f 00 00
7fffffff08 55 24 40 00 00 00 00 00
7fffffff10 55 24 40 00 00 00 00 00
7fffffff18 55 24 40 00 00 00 00 00
```

A simple example

```
\begin{tikzpicture}
\begin{hexdumptikz}{dat.hex}
\hexdumptikzPrint{0,0x08,0x7fffffff0000-0x7fffffff0000}
\hexdumptikzLabelBytes[rounded corners]{0x2}{0xa}
\end{hexdumptikz}
\hexdumptikzLabelBytesFallback[rounded corners,tikz=red]{0x2}{0xa}
\end{tikzpicture}
```

```
0x00 00 00 00 00 00 00 00 00
0x08 00 00 00 00 00 00 00 00
0x7fffffff0000 55 24 40 00 00 00 00 00
0x7fffffff0000 55 24 40 00 00 00 00 00
0x7fffffff0000 55 24 40 00 00 00 00 00
```

2.2 Hexdump-Parser

This package can implement multiple hexdump-parsers. But currently only one is implemented.

2.2.1 hd

This parser works with most formats. At some places it can be adjusted slightly to work with slightly different formats.

Note this parser stops parsing a line after `/hexdumptikz/bytes per rowP.6` have been parsed from the line (continues with the next line then).

This is the whole grammar which it can parse (for some of the optional stuff, specific keys need to be set):

```
<hexdump> → {<hexdumpline> [\n]}+
<hexdumpline> → <address> ["0x"] <values>
<address> → {SPACE} ["0x"] {<hexdigit>}+ {SPACE} [":" ] {SPACE}
<hexdigit> → /[0-9a-fA-F]/
<values> → {{SPACE} <hexdigit> <hexdigit> {SPACE}}
```

2.3 Selector

```
<selector> → {<selectorEle> ","} <selectorEle>
<selectorEle> → <spec> ["|"] <predicate> ["->"] <style>
<spec> → ((<range> | <atom> | <gap>))
<gap> → "x"
<range> → (<addr> "-" <addr> | <idx+x> "-" <idx+x>)
<atom> → (<addr> | <idx+x>)

<idx+x> → <idx> ["/"] /[0-9]+/
<idx> → /[0-9]+/
<addr> → "0x" /[0-9a-fA-F]+/
<style> → /'tikz-code or style'/
<predicate> → [("<x>" | "<y>" | "/" | ("odd" | "even" | <pred_mod>))]
<pred_mod> → "mod (" {SPACE} /[0-9]+/ {SPACE} "," {SPACE} /[0-9]+/ {SPACE} ")"
```

2.3.1 Predicates

Predicates allow to select a specific subset of rows/bytes within the given range. They usually operate on one of the axes *x* or *y*. If no axis is given, *x* is assumed as default.

Modulo The most common predicate is the modulo predicate. The predicate `mod(m,r)` checks the element *e* (*x* or *y* index) as follows: $e \bmod m \stackrel{!}{=} r$.

The following shortcuts are defined:

`odd` \rightarrow `mod(2,1)`

`even` \rightarrow `mod(2,0)`

Note that the selector is usually split around `.`. Thus, when using `mod` directly, you need to enclose it with braces, for example like this `{mod(2,1)}`.

2.4 Generated Nodes

Notation:

`in_idx` index of the row based on the *input* aka linenummer in the input file

`out_idx` index of the row based on the *output* aka line-/rownumber in the shown hexdump

`horizontal_idx` *x-index*

`end` refers to the last node in one direction

`end-end` special: the last byte-node in the last row.

Generated node-names: (Note: `*_idx` start at 0)

- Regular Address-Nodes:

- `hexdumptikz-in-<in_idx>`
- `hexdumptikz-out-<out_idx>`
- `hexdumptikz-<padded_offset>`

- End Address-Nodes:

- `hexdumptikz-end`

- Regular Byte-Nodes:

- `hexdumptikz-in-<in_idx>-<horizontal_idx>`
- `hexdumptikz-out-<out_idx>-<horizontal_idx>`
- `hexdumptikz-<padded_offset>-<horizontal_idx>`

- End Byte-Nodes:

- `hexdumptikz-in-<in_idx>-end`
- `hexdumptikz-out-<out_idx>-end`
- `hexdumptikz-<padded_offset>-end`
- `hexdumptikz-end-end`

- Gap-Nodes: (Note: `gap_num` starts at 1)
 - `hexdumptikz-in-<in_idx>-gap<gap_num>`
 - `hexdumptikz-out-<out_idx>-gap<gap_num>`
 - `hexdumptikz-<padded_offset>-gap<gap_num>`

2.4.1 Nodename-helpers

`\hexdumptikzAddrToNodename{<address>}`

After `\hexdumptikzPrint`^{→P.2} and within `hexdumptikz`^{→P.1}, this allows transforming the address of a byte to its node-name. This allows (moreover simplifies) drawing custom annotations into the hexdump.

Note this is the same functionality `\hexdumptikzLabelBytes`^{→P.2} uses internally.

`\hexdumptikzAddrToRow{<address>}`

After `\hexdumptikzPrint`^{→P.2} and within `hexdumptikz`^{→P.1}, this allows transforming the address of a byte to its row-index. This allows (moreover simplifies) drawing custom annotations into the hexdump.

Note this is the same functionality `\hexdumptikzLabelBytes`^{→P.2} uses internally.

`\hexdumptikzAddrToCol{<address>}`

After `\hexdumptikzPrint`^{→P.2} and within `hexdumptikz`^{→P.1}, this allows transforming the address of a byte to its column-index. This allows (moreover simplifies) drawing custom annotations into the hexdump.

Note this is the same functionality `\hexdumptikzLabelBytes`^{→P.2} uses internally.

3 Options

All keys reside inside the `/hexdumptikz` space. Note this space also searches the `/tikz` space. This way, you can intuitively set e.g. `rounded corners`. But on the other hand, unknown keys are always reported as unknown in `/tikz` and not in `/hexdumptikz`. Also, some keys do not work globally with a plain `\tikzset`. For such cases you may use `/hexdumptikz/tikz`.

`/hexdumptikz/tikz=<tikz-keys>`

Allows to set tikz-keys which do not work “globally” with `\tikzset`. An example is setting the color to `red`.

Internally, every drawing generated by this package is enclosed by a `TikZ-scope`. Thus, this key simply appends to the `/tikz/every scope` style.

`/hexdumptikz/name prefix=<prefix>`

Works similar like `/tikz/name prefix`.

This will also set `name prefix` on the following tikz-scope. Thus, when you draw manually in e.g. `hexdumptikz`^{→P.1} this key is effective.

See section 4.3 for why this is useful.

The following subsections describe options/keys and styles grouped by where they apply to. You may of course set them at other places too, thereby changing the scope of the setting (e.g. set `/hexdumptikz/show addr`^{→P.6} at the `hexdumptikz`^{→P.1} for all `\hexdumptikzPrint`^{→P.2} within the environment).

3.1 Print Hexdump

`/hexdumptikz/addr len=<len>` (initial: 12)

Specify the length of the addresses (the number of hex-digits without the leading 0x). This is the length to which addresses will get padded to.

Note if the real addresses have more digits, there might be issues with the selection and/or styling mechanism.

Further note, this affects the padding used for `/hexdumptikz/show addr padded` as well as the padding used internally for the selection and styling mechanism.

`/hexdumptikz/bytes per row=<BPR>` (initial: 8)

Specify the amount of bytes normally present on one row.

This primarily is used for calculating the length of the addresses suffix which is used when calculating on addresses. But parsers also may stop parsing a line after parsing this amount of bytes (each with two hex-digits).

3.1.1 Drawing

`/hexdumptikz/bytes sep after=<number>` (initial: 64)

Allows to insert a horizontal separating space in the hexdump. This key specifies after how many bytes (*<number>*) the separator should be inserted.

Note: The size of the separator is determined by `/hexdumptikz/bytes sep`

`/hexdumptikz/bytes sep=<dimension>` (initial: 2em)

Allows to insert a horizontal separating space in the hexdump. This key specifies the size of the separator (*<dimension>*).

Related: `/hexdumptikz/bytes sep after`

`/hexdumptikz/show addr=<bool>` (initial: true, default: true)

Whether the hexdump should display the nodes with the addresses/offsets.

Note for generality, the address nodes always are present. But when set to `false`, they are placed so that they are not visible and do not occupy additional space.

`/hexdumptikz/show addr base=<bool>` (initial: true, default: true)

Whether the addresses shown in the hexdump should be prefixed with 0x to indicate the hexadecimal base.

Note this is purely aesthetic and is irrelevant for how to specify addresses when selecting bytes/rows.

`/hexdumptikz/show addr padded=<bool>` (initial: false, default: true)

Whether the addresses shown in the hexdump should be padded to `/hexdumptikz/addr len`. Note this is purely aesthetic, internally addresses are always padded to `/hexdumptikz/addr len`.

`/hexdumptikz/styled bytes=<style-selector>`

This probably is by far the most complex setting in this package. It allows styling individual bytes based on a selection mechanism.

In principle, you specify a mapping `range → tikz-keys` (where you also can specify an additional predicate).

Note how the parser and thus the language used here is exactly the same as in the row-selection mechanism (see section 2.3 and `\hexdumptikzPrint→P.2`). But this time, you'll be using the `-> <style>` component as well.

For examples refer to section 4.

Styles

`/hexdumptikz/address node first` (style)

This style is applied to the first address node.

Note the entire hexdump is drawn based on this first node. Thus, you can leverage the `/tikz/at` key with this style to change the position of the entire hexdump.

`/hexdumptikz/address node` (style)

This style is applied to every address node.

Note this style has some initial code in it which is used for the node placement. Thus, normally you'll want to use the `.append style` handler with this style (and avoid overriding the entire style eventhough it is possible).

`/hexdumptikz/byte node` (style)

This style is applied to every byte node.

Note this style has some initial code in it which is used for the node placement. Thus, normally you'll want to use the `.append style` handler with this style (and avoid overriding the entire style eventhough it is possible).

3.1.2 Parsing

The package is designed to work with multiple parsers. Still so far only one parser exists: `hd`.

`/hexdumptikz/parser=<hd>` (initial: `hd`)

Allows setting the parser which is to be used.

`hd`

`/hexdumptikz/parser opts hd=<keys>`

Sets keys from the `/hexdumptikz/parser opts/hd` space.

Can be used to customize settings for the `hd` parser.

`/hexdumptikz/parser opts/hd/strict byte number per row=<bool>` (initial: false, default: true)

With this enabled, the parser will check that a row does not contain more than `/hexdumptikz/bytes per row`^{P. 6} bytes. Else it will raise an error.

With this disabled, the parsing just stops silently at `/hexdumptikz/bytes per row`^{P. 6} bytes.

`/hexdumptikz/parser opts/hd/strict hex digits=<bool>` (initial: false, default: true)

Strictly, the bytes do not get interpreted as of now. Thus, it does not matter whether valid hex digits are given (they just are read in pairs of two digits). This option allows to assert the byte values are valid hex digits (else an error will be raised).

`/hexdumptikz/parser opts/hd/leading value base=<bool>` (initial: false, default: true)

When enabled, the parser will remove once per row a leading `0x` from the field representing the byte-values. Thus, a format like in the following example can be parsed:

```
0x000 0x01020304
0x005 0x05060708
```

4 Extended Examples

4.1 Datafiles used

file: dat.hex

```
00          00 00 00 00 00 00 00 00
08          00 00 00 00 00 00 00 00
10          00 00 00 00 00 00 00 00
7fffffffef10 58 ee ff ff ff 7f 00 00
7fffffffef18 55 24 40 00 00 00 00 00
7fffffffefea0 55 24 40 00 00 00 00 00
7fffffffefea8 55 24 40 00 00 00 00 00
```

file: dat-small.hex

```
0          00 00
2          00 00
4          00 00
6          00 00
8          00 00
```

4.2 Example 1

Example 1

```
\begin{tikzpicture}
  \begin{hexdumptikz}[addr len=12,parser=hd,parser opts hd={strict hex digits=true,leading 2}
    {value base=false}]{dat.hex}
    \hexdumptikzPrint[
      styled bytes={
        0x00-0x07 | odd -> {blue},
        0x00-0x07 | even -> {green!50!black},
        0x7fffffffef18-0x7fffffffefea | {mod(4,2)} -> {draw=red},
      },
      ]{0,0x8,0x7fffffffef18-0x7fffffffefea8}
      %
      \hexdumptikzLabelBytes[rounded corners]{0x2}{0xa}
    \end{hexdumptikz}
  %
  \begin{hexdumptikz}[
    bytes per row=2,addr len=1,
    address node first/.style={
      at={(hexdumptikz-out-0-end)},right,xshift=4em
    },
  ]{dat-small.hex}
    \hexdumptikzPrint[
      styled bytes={},
      ]{0-4}
      %
      \hexdumptikzLabelBytes[rounded corners]{0x2}{0x6}
    \end{hexdumptikz}
  %
  \hexdumptikzLabelBytesFallback[
    bytes per row=2,addr len=1,rounded corners,tikz=red
```



```
] {0x2} {0x6}
\end{tikzpicture}
```

0x00	00 00	00 00 00 00 00 00	0x0	00 00
0x08	00 00 00	00 00 00 00 00 00	0x2	00 00
0x7fffffff18	55 24	40 00 00 00 00 00	0x4	00 00
0x7fffffff1a	55 24	40 00 00 00 00 00	0x6	00 00
0x7fffffff1c	55 24	40 00 00 00 00 00	0x8	00 00

4.3 Avoid nodename collisions

See how in this example the fallback labeling does not really know which hexdump to refer to:

Colliding nodenames

```
\begin{tikzpicture}
  \begin{hexdumptikz}[bytes per row=2,addr len=1]{dat-small.hex}
    \hexdumptikzPrint{0-4}
    \hexdumptikzLabelBytes[rounded corners]{0x2}{0x6}
  \end{hexdumptikz}
  %
  \begin{hexdumptikz}[
    bytes per row=2,addr len=1,
    address node first/.style={
      at={(hexdumptikz-out-0-end)},right,xshift=4em
    },
  ]{dat-small.hex}
    \hexdumptikzPrint{0-4}
    \hexdumptikzLabelBytes[rounded corners]{0x2}{0x6}
  \end{hexdumptikz}
  %
  \hexdumptikzLabelBytesFallback[
    bytes per row=2,addr len=1,rounded corners,tikz=red
  ]{0x2}{0x6}
\end{tikzpicture}
```

0x0	00 00	0x0	00 00
0x2	00 00	0x2	00 00
0x4	00 00	0x4	00 00
0x6	00 00	0x6	00 00
0x8	00 00	0x8	00 00

You can solve this issue by prefixing the nodenames with `/hexdumptikz/name prefix`^{P.5} so the nodenames are differentiate between the two dumps:

Avoid colliding nodenames

```
\begin{tikzpicture}
  \begin{hexdumptikz}[
    name prefix=hd1-,tikz={local bounding box=hd1},
    bytes per row=2,addr len=1
  ]{dat-small.hex}
```

```

\hexdumptikzPrint{0-4}
\hexdumptikzLabelBytes[rounded corners]{0x2}{0x6}
\end{hexdumptikz}
%
\begin{tikzpicture}
  \begin{tikzpicture}
    name prefix=hd2-,bytes per row=2,addr len=1,
    address node first/.style={
      at={(hd1-hd1.north east)},below right,xshift=4em
    },
  ]{dat-small.hex}
  \hexdumptikzPrint{0-4}
  \hexdumptikzLabelBytes[rounded corners]{0x2}{0x6}
\end{tikzpicture}
%
\hexdumptikzLabelBytesFallback[
  name prefix=hd1-,bytes per row=2,addr len=1,rounded corners,tikz=red
]{0x2}{0x6}
%
\node[above right] at (hd1-hexdumptikz-in-0-end.east) {x1};
\node[below right] at (hd1-hexdumptikz-out-0-end.east) {y1};
%
\node[above right] at (hd2-hexdumptikz-in-0-end.east) {x2};
\node[below right] at (hd2-hexdumptikz-out-0-end.east) {y2};
\end{tikzpicture}

```

0x0 00 00	^{x1}	0x0 00 00	^{x2}
0x2 00 00	^{y1}	0x2 00 00	^{y2}
0x4 00 00		0x4 00 00	
0x6 00 00		0x6 00 00	
0x8 00 00		0x8 00 00	

Also see how this allows you to specify for fully custom annotations to which hexdump the nodename refers to.

4.4 Gap nodes

Gap nodes

```

\begin{tikzpicture}[gap/.style={left,inner xsep=0pt,font=\ttfamily\small}]
\begin{tikzpicture}
\begin{tikzpicture}
\hexdumptikzPrint{0,0x08,x,0x10,x,x,0x7fffffff18-0x7fffffff18,x}
\node[gap,blue] at (hexdumptikz-out-2-gap1) {*};
\node[gap,green!50!black] at (hexdumptikz-out-3-gap1) {*};
\node[gap,red] at (hexdumptikz-0x7fffffff18-gap2) {*};
\end{tikzpicture}
\end{tikzpicture}
\end{tikzpicture}

```

<hr/>									
0x00	00	00	00	00	00	00	00	00	00
0x08	00	00	00	00	00	00	00	00	00
*									
0x10	00	00	00	00	00	00	00	00	00
*									
*									
0x7fffffffef18	55	24	40	00	00	00	00	00	00
0x7fffffffefa0	55	24	40	00	00	00	00	00	00
0x7fffffffefa8	55	24	40	00	00	00	00	00	00

See how the gap-nodenames (and also the generation of the gaps) always are coupled to the succeeding address(node) drawn. Thus, the last gap-node is not drawn at all.

5 Implementation

see `hexdumptikz-code.pdf`⁵

⁵eg on `hexdumptikz` ^{CTAN} (CTAN)

Index

`addr len` key, 6
`address node` key, 7
`address node first` key, 7

`byte node` key, 7
`bytes per row` key, 6
`bytes sep` key, 6
`bytes sep after` key, 6

Commands

`\hexdumptikzAddrToCol`, 5
`\hexdumptikzAddrToNodename`, 5
`\hexdumptikzAddrToRow`, 5
`\hexdumptikzLabelBytes`, 2
`\hexdumptikzLabelBytesFallback`, 2
`\hexdumptikzPrint`, 2

Environments

`hexdumptikz`, 1

`hexdumptikz` environment, 1
`\hexdumptikzAddrToCol`, 5
`\hexdumptikzAddrToNodename`, 5
`\hexdumptikzAddrToRow`, 5
`\hexdumptikzLabelBytes`, 2
`\hexdumptikzLabelBytesFallback`, 2
`\hexdumptikzPrint`, 2

Keys

`/hexdumptikz/`
 `addr len`, 6
 `address node`, 7
 `address node first`, 7
 `byte node`, 7
 `bytes per row`, 6
 `bytes sep`, 6
 `bytes sep after`, 6
 `name prefix`, 5
 `parser`, 7
 `parser opts hd`, 7
 `show addr`, 6
 `show addr base`, 6
 `show addr padded`, 6
 `styled bytes`, 6
 `tikz`, 5
`/hexdumptikz/parser opts/hd/`
 `leading value base`, 7
 `strict byte number per row`, 7
 `strict hex digits`, 7

`leading value base` key, 7

`name prefix` key, 5

`parser` key, 7
`parser opts hd` key, 7

`show addr` key, 6
`show addr base` key, 6
`show addr padded` key, 6
`strict byte number per row` key, 7
`strict hex digits` key, 7
`styled bytes` key, 6

`tikz` key, 5