Working with binary descriptors: `west bindesc`

The `bindesc` command allows users to read *binary descriptors* of executable files. It currently supports `.bin`, `.hex`, `.elf` and `.uf2` files as input.

You can search for a specific descriptor in an image, for example:

```
west bindesc search KERNEL_VERSION_STRING build/zephyr/zephyr.bin
```

You can search for a custom descriptor by type and ID, for example:

```
west bindesc custom_search STR 0x200 build/zephyr/zephyr.bin
```

You can dump all of the descriptors in an image using:

```
west bindesc dump build/zephyr/zephyr.bin
```

You can list all known standard descriptor names using:

```
west bindesc list
```

The Devicetree Shell: `west dtsh`

The `dtsh` command opens a DTS file in a shell-like interactive command line interface:

```
west dtsh [DTS]
```

If the command line does not explicitly specify a DTS file path, `dtsh` will try to open the devicetree at `build/zephy/zephyr.dts`, which allows the simple work-flow bellow:

```
$ west build
$ west dtsh
dtsh (0.0.99): Shell-like interface with Devicetree
How to exit: q, or quit, or exit, or press Ctrl-D

> ls -l
Name              Labels          Binding
-----------------------------------------------------
chosen
aliases
soc
pin-controller    pinctrl         nordic,nrf-pinctrl
entropy_bt_hci    rng_hci         zephyr,bt-hci-entropy
sw-pwm            sw_pwm          nordic,nrf-sw-pwm
cpus
leds                              gpio-leds
pwmleds                           pwm-leds
buttons                           gpio-keys
connector         arduino_header  arduino-header-r3
analog-connector  arduino_adc     arduino,uno-adc
```

By default, `dtsh` will try to retrieve, or work out, the bindings Zephyr has used at build-time to generate the DTS file, or would use if it were to generate it *now*.

The `-b` option overrides this behavior by explicitly enumerating the directories to search for binding files:

```
west dtsh -b dir1 -b dir2
```

For the full command synopsis, run `west dtsh -h`.

Please refer to the DTSh *User Guide* for detailed documentation.

DTSh User Guide **DTSh** is an interactive *Devicetree* source file (DTS) viewer with a shell-like command line interface:

- browse a devicetree through a hierarchical file system metaphor
- search for devices, bindings, buses or interrupts with flexible criteria
- filter, sort and format commands output
- generate simple documentation artifacts (text, HTML, SVG) by redirecting the output of commands to files
- *rich* Textual User Interface, command line auto-completion, command history, user themes

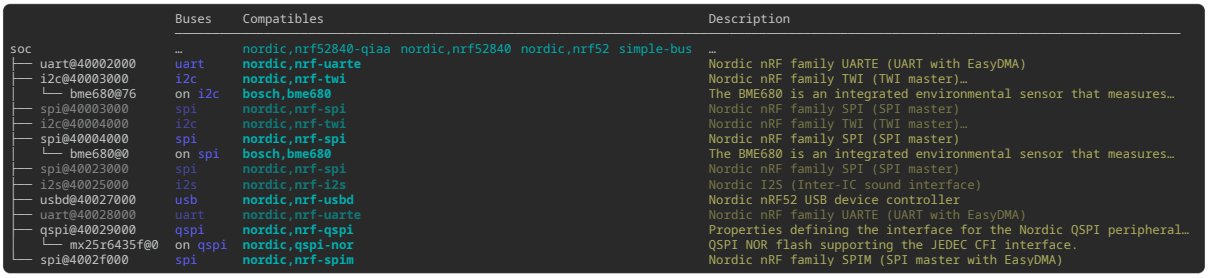

Fig. 4: `find --with-bus * --OR --on-bus * -T --format NYcd > buses.svg`

Getting Started Getting started with the Devicetree shell:

- verify a few requirements
- open DTS files with `west dtsh`
- the bindings search path
- user files

Requirements For a better use experience, please check:

- the GNU Readline library integration with Python
- the Terminal application and its configuration

Warning: On **Windows**, the GNU Readline support will likely be disabled, resulting in a **degraded user experience**: no auto-completion nor command history.

GNU Readline For auto-completion, command history and key bindings, DTSh relies on the GNU Readline library variant of the standard Python readline API.

GNU Linux

On GNU/Linux systems, the GNU Readline library is most likely already installed, e.g. as a dependency of the Bash shell or of the Python package for your distribution.

The `readline` module of the Python standard library should load GNU Readline (*libreadline*).

macOS

On macOS, the Readline functionality is provided by the Editline library (aka *libedit*), which is not fully compatible with the GNU Readline API.

Here, the Python `readline` module can either wrap *libedit* or a bundled GNU Readline library (*libreadline*), depending on how Python was installed.

To streamline the installation, DTSh requirements on macOS include a Stand-alone GNU readline module, which will be substituted for the `readline` module of the Python standard library: this is the supported setup.

If it is known that the Python run-time already wraps the GNU Readline library, it should be safe to uninstall the stand-alone module: `pip uninstall gnureadline`.

Windows

Although the reference documentation might still lack in clarity, Python does no longer provide the `readline` module of the standard library on non POSIX platforms. And DTSh does not currently implement any work-around.

On Windows, it is therefore very likely that the GNU Readline integration will be **disabled**, resulting in a **degraded user experience**, especially the lack of command history and auto-completion.

The Terminal The terminal is expected to support the standard ANSI escape sequences (plus the OSC 8 extension for *hyperlinks*): most Terminal applications used *today* on GNU Linux, macOS (e.g. Terminal.app and iTerm2) and Windows (since OSC 8 support for conhost and terminal) should be fine.

DTSh commands output may involve a few Unicode glyphs for common symbols like ellipsis, arrows and box-drawing: if any appear to be missing, either configure *alternative symbols* in a preferences file, or install a *no tofu* monospace font (e.g. Source Code Pro) to use in the terminal.

Tip: To address accessibility issues, or improve an unpleasant user experience:

- first, try configuring the terminal itself: font, text and background colors, colors palette, options like *Show bold text in bright colors* (these are often grouped as *profiles*)
- once it's mostly legible and pleasant, adjust DTSh styles and colors with a *user theme*
- it's recommended to run DTSh full screen or in a wide/maximized window

If trees in the Terminal look like the examples in this documentation, you're missing the Unicode range `2500–257F`.

Usage DTSh runs as a Zephyr extension to *West*:

```
usage: west dtsh [-h] [-b DIR] [-u] [--preferences FILE] [--theme FILE] [DTS]

options:
  -h, --help            show this help message and exit

open a DTS file:
  -b DIR, --bindings DIR
                        directory to search for binding files
  DTS                   path to the DTS file

user files:
  -u, --user-files      initialize per-user configuration files and exit
  --preferences FILE    load additional preferences file
  --theme FILE          load additional theme file
```

Open DTS files To open a DTS file in the Devicetree shell, simply pass its path as the West command parameter, e.g.:

```
$ west dtsh build/zephyr/zephyr.dts
dtsh (0.0.99): Shell-like interface with Devicetree
How to exit: q, or quit, or exit, or press Ctrl-D

/
> find --also-known-as partition --with-reg-size >48k --format NKr
Name             Also Known As             Registers
═══════════════════════════════════════════════════════
partition@c000   image-0, slot0_partition  0xc000 (472 kB)
partition@82000  image-1, slot1_partition  0x82000 (472 kB)
```

Tip: If the command line does not specify a DTS file path, `dtsh` will try to open the devicetree at `build/zephy/zephyr.dts`, e.g.:

```
$ west build
$ west dtsh
```

The Bindings Search Path A DTS file alone is actually an incomplete Devicetree source: interpreting its contents requires finding the defining bindings.

By default, DTSh will fist try to retrieve the bindings Zephyr has used at build-time, when the DTS file was generated. For this, it will rely on the CMake cache file contents, assuming a typical build layout:

```
build
├── CMakeCache.txt
└── zephyr
    └── zephyr.dts
```

This is the most straight forward way to get a complete and legit bindings search path.

When no suitable CMake cache is available, DTSh will instead try to work out the search path Zephyr would use if it were to generate the DTS *now* (*Where bindings are located*): bindings found in `$ZEPHYR_BASE/dts/bindings` and other *default* directories should still cover the most simple use cases (e.g. Zephyr samples).

When this default behavior fails to retrieve the appropriate binding files, the `-b --bindings` option permits to explicitly enumerate all the directories to search in.

```
$ west dtsh --bindings dir1 --bindings dir2
```

User Files Users can tweak DTSh appearance and behavior by overriding its defaults in configuration files:

- `dtsh.ini`: to override global preferences (*User Preferences*)
- `theme.ini`: to override styles and colors (*User Themes*)

These files should be located in a platform-dependent directory, e.g. `~/.config/dtsh` on GNU/Linux systems.

Running `dtsh` with the `--user-files` option will initialize configuration templates at the proper location:

```
$ west dtsh --user-files
User preferences: ~/.config/dtsh/dtsh.ini
User theme: ~/.config/dtsh/theme.ini
```

Tip: DTSh won't override a user file that already exists: manually remove the file(s), and run the command again.

Eventually:

- the `--preferences FILE` option permits to specify an additional preferences file to load
- the `--theme FILE` option permits to specify an additional theme file to load

The Shell DTSh is an interactive interface with a devicetree and its bindings that resembles a POSIX shell.

Hierarchical File System Metaphor In POSIX systems, all directories are part of a global file system tree, the root of which is denoted `/`.

DTSh shows a devicetree as such a *hierarchical file system*, where a Devicetree *path name* (DTSpec 2.2.3) may represent:

- a node that appears as a *file* whose name would be the *node name* (DTSpec 2.2.1), and whose contents would be the node's properties
- a branch that appears as a *directory* of nodes

Devicetree path names are then *absolute paths*.

A *current working branch* is defined, allowing support for *relative paths*. The usual *cd* command navigates through the devicetree.

Paths may contain the habitual *path references*:

- `.` represents the current path, typically the current working branch
- `..` represents the path to the parent of the current path

Paths may start with a DTS *label* (DTSpec 6.2): `&i2c0` represents the absolute path to the devicetree node with label "i2c0".

By convention, the devicetree root *directory* is its own parent.

The expression bellow is then a valid (convoluted) path to the `flash-controller@4001e000` device:

```
&i2c0/bme680@76/../../../soc/./flash-controller@4001e000
```

The Command Line A `dtsh` *command line* is parsed into a *command string* followed by an optional *output redirection*:

```
COMMAND_LINE := COMMAND_STRING [REDIRECTION]
```

Command lines are entered and edited at *the prompt*.

Command Strings `dtsh` command strings conform to GNU getopt syntax:

```
COMMAND_STRING := COMMAND [OPTION...] [PARAM...]
```

where:

- `COMMAND`: the command name, e.g. `ls`
- `OPTION`: the options the command is invoked with, e.g. `--enabled-only`
- `PARAM`: the parameters the command is invoked with, e.g. a devicetree path

`OPTION` and `PARAM` are not positional: `ls -l /soc` is equivalent to `ls /soc -l`.

An option may accept:

- a short name, starting with a single `-` (e.g. `-h`)
- and/or a long name, starting with `--` (e.g. `--help`)

Short option names can combine: `-lR` is equivalent to `-l -R`.

An option may require an argument value, e.g. `find --with-reg-size >0x1000`.

Note: An argument value can contain spaces if quoted: `find --with-reg-size ">0x1000"` will as expected match nodes with a register whose size is greater than 4 kB, but `find --with-reg-size >0x1000` would complain that > alone is an invalid expression.

Options semantic should be consistent across commands, e.g. `-l` always means *use a long listing format*.

`dtsh` also tries to re-use *well-known* option names, e.g. `-r` for *reverse* or `-R` for *recursive*.

Output Redirection Command output redirection follows a POSIX-like syntax:

- starts with either > (create), or >> (append to)
- followed by a file path, the extension of which will determine the file format (HTML, SVG, text)

Appending to structured contents (HTML, SVG) is supported.

For example, the command line bellow will save the tree of the flash partitions to the file `flash.svg` in the current working directory, in SVG format:

```
/
> tree &flash_controller > flash.svg
```

Then, the command line bellow will *append* details about individual partitions:

```
/
> ls --format naLrC &flash0/partitions >> flash.svg
```

When a command output is redirected, the terminal width is virtually re-sized to match the output contents, up to a configurable limit (default is 255 characters): this avoids unnecessarily limiting the command output to the actual terminal width.

See *Output Redirection* for all available options.

Tip: By default, DTSh won't override any existing file, to prevent unintentional operations, e.g.:

```
/
> ls > /path/to/very/important/file
```

This is obviously orthogonal to the append mode (>>):

```
/
> tree /soc > soc.svg

/
```

(continues on next page)

(continued from previous page)

```
> ls /soc >> soc.svg
dtsh: file exists: 'soc.svg'
```

To enable the append mode, unset the `pref.fs.no_overwrite` preference.

Format Commands Output We here consider the DTSh commands that will eventually enumerate devicetree nodes, like *shell* commands may list files or directories.

Nodes may be represented:

- by a path: this is the default, and the DTSh command's output should then follow the layout of its Unix-like homonym (e.g. `ls`, `find` or `tree`)
- using a *long listing format*: the command's output is then formatted in columns, like the Unix command `ls` that will also show owner, permissions and file size when the `-l` option is set

DTSh generalizes and extends this approach to all commands that enumerate nodes:

- the `-l` option enables long listing formats
- the `--format FMT` option explicitly sets what information is shown (the columns) through a *format string*

Setting a format string with `--format FMT` implies `-l`. What the `-l` option alone will show depends on configurable default format strings.

Formatted outputs include *list* and *tree* views.

Tip: DTSh is a bit biased toward POSIX shells' users, for who it should make sense to default to POSIX-like output:

- by default commands output will sound familiar, and nothing is printed that the user has not explicitly asked for
- then, add DTSh format strings (`--format FMT`) to select the relevant columns to show on a per-command basis

This behavior can however be overridden by setting the `pref.always_longfmt` preference: the `-l` flag will then be implied wherever supported, and outputs will be formatted according to configurable defaults. Although `--format` will still permit to select columns on a per-command basis, there's no syntax to get the POSIX-like output when this preference is set.

Format Strings A *format string* is simply a list of *specifier* characters, each of which appending a *column* to the formatted output.

For example, the format string "Nd" specifies the *Name* and *Description* columns:

```
/soc
> ls --format Nd
 Name                          Description
 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 interrupt-controller@e000e100  ARMv7-M NVIC (Nested Vectored Interrupt Controller)
 timer@e000e010                 ARMv7-M System Tick
 ficr@10000000                  Nordic FICR (Factory Information Configuration Registers)
```

Table 6: Format string specifiers

a	unit address
A	node aliases
b	bus of appearance
B	supported bus protocols
c	compatible strings
C	binding's compatible or headline
d	binding's description
D	node dependencies
i	generated interrupts
K	all labels and aliases
l	device label
L	DTS labels
N	node name
n	unit name
o	dependency ordinal
p	path name
r	registers (base address and size)
R	registers (address range)
s	status string
T	dependent nodes
v	vendor name
X	child-binding depth
Y	bus information

Formatted Lists A *formatted list* is basically a table:

- the format string describes the ordered columns
- each node in the list appends a table row

This is the view most commands will produce by default when using long listing formats.

```
/
> ls leds --format NLC
 Name   Labels  Binding
 ═══════════════════════════════
 led_0  led0    GPIO LED child node
 led_1  led1    GPIO LED child node
 led_2  led2    GPIO LED child node
 led_3  led3    GPIO LED child node
```

When no format string is explicitly set with `--format FMT`, the default format configured by the `pref.list.fmt` preference is used.

Preferences for formatted lists also include e.g. whether to show the table header row or how to represent missing values (*placeholders*).

Formatted Trees A *formatted tree* is actually a 2-sided view:

- left-side: the actual node tree
- right-side: a detailed list-view
- the first column specified by the format string tells how to represent the tree anchors (left-side), while the remaining specifiers describe the detailed view columns (right-side)

Formatted trees are produced:

- by commands whose natural semantic is to output trees, e.g. `tree -l`

- when the user explicitly asks for a tree-like representation, e.g. `find -l -T`

```
/
> find --on-bus * --format NYC -T
                          Bus      Binding
                          ═══════════════════════
/                         …        …
└── soc                   …        …
    ├── i2c@40003000      i2c      nordic,nrf-twi
    │   └── bme680@76     on i2c   bosch,bme680
    ├── spi@40004000      spi      nordic,nrf-spi
    │   └── bme680@0      on spi   bosch,bme680
    └── qspi@40029000     qspi     nordic,nrf-qspi
        └── mx25r6435f@0  on qspi  nordic,qspi-nor
```

When no format string is explicitly set with `--format FMT`, the default format configured by the `pref.tree.fmt` preference is used.

Preferences for formatted trees also include e.g. whether to show the table header row or how to represent missing values (*placeholders*).

Sort Commands Output When a command eventually enumerate nodes, ordering its output means ordering the nodes.

When the command outputs a list (e.g. `ls` or `find`), the nodes are eventually sorted at once as a whole.

When the command outputs a tree (e.g. `tree` or `find -T`), nodes are sorted branch by branch while walking through the devicetree (*children ordering*).

How nodes are sorted is specified with the `--order-by KEY` command option, where `KEY` is simply a single-character identifier for the order relationship.

The `-r` command option will reverse the command's output.

Note: When no order relationship is explicitly set, nodes are expected to appear in the order they appear when walking through the DTS file.

Sort Keys A sort *key* identifies an order relationship.

Table 7: Sort keys

a	sort by unit address
A	sort by aliases
B	sort by supported bus protocols
b	sort by bus of appearance
c	sort by compatible strings
C	sort by binding
i	sort by IRQ numbers
I	sort by IRQ priorities
l	sort by device label
L	sort by node labels
N	sort by node name
n	sort by unit name
o	sort by dependency ordinal
p	sort by node path
r	sort by register addresses
s	sort by register sizes
v	sort by vendor name
X	sort by child-binding depth

Tip:

- When applicable, sort keys and format specifiers will represent the same *aspect* of a node, e.g. `a` represents unit addresses both in format strings and as a sort key.
- Any ordering relationship can be applied to any set of nodes, e.g. sorting by IRQ numbers even though not all nodes generate interrupts (see *Sort Directions* bellow).

Sort Directions In the default direction (*ascending*), nodes for which the sort key has no value will appear last (after all nodes for which the sort key has a value): e.g. devices that do not generate interrupts will appear last.

In reversed order (*descending* direction), nodes for which the sort key has no value will appear first (before the nodes for which the sort key has a value): e.g. devices that do not generate interrupts will appear first.

If the sort key admits multiple values (e.g. register sizes), for each node:

- the lowest value (e.g. the smallest register size) is used in the ascending direction
- the highest value (e.g. the largest register size) is used in the descending direction

To improve legibility, key values may also be sorted *horizontally*.

For example, when sorting nodes by register sizes in the ascending direction (`--order-by s`):

```
timer@e000e010                0xe000e010 16 bytes
gpio@50000000                 0x50000000 512 bytes, 0x50000500 768 bytes
gpio@50000300                 0x50000300 512 bytes, 0x50000800 768 bytes
...
qspi@40029000                 0x40029000 4 kB, 0x12000000 128 MB
pwm@4002d000                  0x4002d000 4 kB
spi@4002f000                  0x4002f000 4 kB
crypto@5002a000               0x5002a000 4 kB, 0x5002b000 4 kB
memory@20000000               0x20000000 256 kB
```

And, when sorting nodes by register sizes in the descending direction (`--order-by s -r`):

```
qspi@40029000                   0x12000000 128 MB, 0x40029000 4 kB
memory@20000000                 0x20000000 256 kB
crypto@5002a000                 0x5002a000 4 kB, 0x5002b000 4 kB
spi@4002f000                    0x4002f000 4 kB
pwm@4002d000                    0x4002d000 4 kB
...
gpio@50000300                   0x50000800 768 bytes, 0x50000300 512 bytes
gpio@50000000                   0x50000500 768 bytes, 0x50000000 512 bytes
timer@e000e010                  0xe000e010 16 bytes
```

Search the Devicetree Searching the devicetree for devices, bindings, buses or interrupts is simply matching nodes with criteria, each of which represents a predicate applied to an aspect of the node.

A predicate is specified either as a *text pattern* or an *integer expression*, depending on the node aspect it applies to: e.g. matching compatible strings involves strings, while matching addresses or sizes involves integers.

Criteria may be chained.

See also the *find* command for detailed examples.

Text Patterns A criterion that applies to a node aspect that has a natural textual representation is specified by a text *pattern*, and may behave as a Regular Expression match or a plain text search.

When the criterion is a RE match (`-E` command flag), any character in the pattern may be interpreted as special character:

- in particular, `*` will represent a repetition qualifier, not a wild-card for any character: e.g. the pattern "*" would be invalid because *there's nothing to repeat*
- parenthesis will group sub-expressions, as in `(image|storage).*`
- brackets will mark the beginning and end of a character set, as in `i[\d]c`

When the criterion is not a RE match (default), but the pattern contains at least one `*`:

- `*` is actually interpreted as a wild-card and not a repetition qualifier: here "*" is a valid expression that actually means *anything*, and will match any node for which the aspect has a value, e.g. `--on-bus *` will match all nodes that appear on a bus
- the criterion behaves as a string-match: `*pattern` means ends with "pattern", and `pattern*` means starts with "pattern" (`*pattern*` would be a convoluted syntax for a plain text search)

Eventually, when the criterion is not a RE match, and the pattern does not contain any `*`, a *plain text search* happens.

Tip: When the command option `-i` is set, text patterns are assumed case-insensitive.

Table 8: Text-based criteria

`--also-known-as`	match labels or aliases
`--chosen-for`	match chosen nodes
`--on-bus`	match buse of appearance
`--with-alias`	match aliases
`--with-binding`	match binding's compatible or headline
`--with-bus`	match supported bus protocols
`--with-compatible`	match compatible strings
`--with-description`	grep binding's description
`--with-device-label`	match device label
`--with-label`	match node labels
`--with-name`	match node name
`--with-status`	match status string
`--with-unit-name`	match unit name
`--with-vendor`	match vendor prefix or name

Integer Expressions A criterion that applies to a node aspect that has a natural integer representation is specified by an *integer expression*:

- a wild-card character alone "*" will match any node for which the aspect has a value, e.g. `--with-irq-number *` would match any node that generates interrupts
- an integer value alone (decimal or hexadecimal) will match nodes for which the aspect has this exact value, e.g. `--with-irq-number 1` would match nodes that generate IRQ 1
- a comparison operator followed by an integer value will match nodes for which the expression evaluates to true, e.g. `--with-reg-size >0x1000` would match nodes that have a register whose size is greater than 4 kB

Simple comparison operators are supported: <, >, <=, >=, !=, = (where the later is equivalent to no operator).

Eventually, an integer expression may end with a SI unit (`kB`, `MB` and `GB`), e.g.:

```
--with-reg-size "256 kB"
--with-reg-size >=4kB
--with-reg-size "> 1 MB"
```

Tip: SI units are case-insensitive and can be truncated to the first letter, e.g.:

```
--with-reg-size 256k
--with-reg-size >4K
--with-reg-size ">= 1M"
```

Table 9: Integer-based criteria

`--with-irq-number`	match IRQ numbers
`--with-irq-priority`	match IRQ priorities
`--with-reg-addr`	match register addresses
`--with-reg-size`	match register sizes
`--with-unit-addr`	match unit addresse
`--with-binding-depth`	match child-binding depth

Criterion Chains By default, chained criteria evaluate as a logical conjunction:

```
--also-known-as partition --with-reg-size >64kB
```

would match nodes that have a register larger than 64 kB, and a label or an alias that contains the string “partition”.

When the `--OR` option is set, the criterion chain will instead evaluate as a logical disjunction of all criteria. For example, the chain bellow would match nodes that are either a bus device or a device connected to a bus:

```
--with-bus * --OR --on-bus *
```

A logical negation may eventually be applied to the criterion chain with the option `--NOT`:

```
/
> find --NOT --with-description *
.
./chosen
./aliases
./soc
./cpus
```

`NOT` and `OR` can combine: they then both apply to the whole criterion chain.

Note: Since, like any option, `NOT` and `OR` are not positional, they always apply to the whole chain, such that the expressions bellow all are valid syntax for the same semantic, not different predicates:

```
--NOT --with-compatible * --OR --with-binding *

--with-compatible * --OR --NOT --with-binding *

--with-compatible * --with-binding * --NOT --OR
```

User Preferences User *preferences* permit to override DTSh defaults for:

- the prompt
- commands output redirection
- the contents of formatted lists and trees
- common symbols and other miscellaneous configuration options

Preferences are loaded in that order:

1. DTSh loads default values for all defined configuration options
2. if a user preferences file exists, e.g. `~/.config/dtsh.ini` on GNU Linux, it’s loaded, overriding the default values with the options it contains
3. if an additional preferences file is specified with the West command option `--preferences`, it’s eventually loaded, overriding previous values with the options it contains

See *User Files* to initialize a preferences file template at the proper location.

Configuration Format DTSh is configured with simple INI files that contain key-value pairs.

Values support *interpolation* with the `${key}` syntax:

```
# Define a key.
wchar.ellipsis = \u2026

# Reference it with interpolation.
et_caetera = Et caetera${wchar.ellipsis}

# Use $$ to escape the dollar sign.
dollar = This is the dollar sign: $$
```

Values are typed:

String
: Strings may contain Unicode characters as literals or 4-digit hexadecimal code points.

  It's necessary to double-quote strings only when:

  - the string value actually ends with spaces
  - the string value contains the double quote character

  Leading and trailing double quotes are always striped.

Boolean
: Valid values (case-insensitive):

  - True: `1`, `yes`, `true`, and `on`
  - False: `0`, `no`, `false`, and `off`

Integer
: Integers in base-10, base-2 (prefix `0b`), base-8 (prefix `0o`), and base-16 (prefix `0x`) are supported.

  Prefixes are case insensitive.

Float
: Decimal (e.g. `0.1`) and scientific (e.g. `1e-1`) notations are supported.

The Prompt These configuration options permit to customize the DTSh *prompt*.

To properly integrate with GNU Readline, we must provide an ANSI prompt where Select Graphic Rendition (SGR) sequences for colors and styles are *protected* by specific markers. Refer to the comments in the configuration template for details.

`prompt.wchar`
: Default prompt.

  This is the character (or string) from which are derived the ANSI prompts bellow.

  Default: Medium Right-Pointing Angle Bracket Ornament (`\u276D`)

`prompt.default`
: ANSI prompt.

  Default: Medium Right-Pointing Angle Bracket Ornament, slate blue

`prompt.alt`
: ANSI prompt for the error state.

  This prompt is used after a command has failed, and until a command succeeds.

  Default: Medium Right-Pointing Angle Bracket Ornament, dark red

`prompt.sparse`
: Whether to insert a newline between a command output and the next prompt.

  Default: Yes

Tip: To simply change the prompt character (Unicode U+276D), no need to actually mess with the ANSI strings, just set `prompt.wchar` to your liking.

Alternative Symbols Preferences whose keys start with `wchar.` permit to override the characters DTSh will use for some common symbols:

- to work-around missing Unicode glyphs: e.g. substitute `->` for the right-arrow U+2192
- to suit personal taste

Output Redirection These preferences configure *commmand output redirection*.

pref.redir2.maxwidth
Maximum width in number of characters for command output redirection.

VT-like terminals do not scroll horizontally, and will crop or wrap what's printed to match the current terminal width.

When redirecting commands output to files, this may unnecessarily limit the width of the produced documents (SVG, HTML and plain text): these will be used in viewers (e.g. a text editor or Web brower) that can handle outputs significantly wider than the current width of the console (e.g. with horizontal scrolling).

This preference configures the maximum width of the virtual console DTSh will actually redirect the command's output to.

Default: 255

pref.html.theme
Preferred default text and background colors to use when redirecting commands output to HTML files.

Possible values:

- `svg`: default theme for SVG documents (dark background, light text)
- `html`: default theme for HTML documents (light background, dark text)
- `dark`: darker
- `light`: lighter
- `night`: darkest, highest contrasts

Default: `html`

pref.html.font_family
Preferred font to use when redirecting commands output to HTML files.

This the family name, e.g. "Courier New".

Multiple coma separated values allowed, e.g. "Source Code Pro, Courier New".

The generic "monospace" family is automatically appended as fallback.

Default: `Courier New` (Web Safe Fonts)

pref.svg.theme
Preferred default text and background colors to use when redirecting commands output to SVG files.

Possible values:

- `svg`: default theme for SVG documents (dark background, light text)
- `html`: default theme for HTML documents (light background, dark text)

- `dark`: darker
- `light`: lighter
- `night`: darkest, highest contrasts

Default: `svg`

`pref.svg.font_family`
Preferred font to use when redirecting commands output to SVG files.

This the family name, e.g. “Courier New”.

Multiple coma separated values allowed, e.g. “Source Code Pro, Courier New”.

The generic “monospace” family is automatically appended as fallback.

Default: `Courier New` (Web Safe Fonts)

`pref.svg.font_ratio`
Font aspect ratio to use when redirecting commands output to SVG files.

This is the width to height ratio, which varies with fonts.

Most monospace fonts will render fine with a 3:5 ratio.

Default: 0.6

Formatted Lists These preferences configure *formatted lists*.

`pref.list.headers`
Whether to show the header row.

Default: Yes

`pref.list.place_holder`
Place holder text for missing values.

Default: None (*blank*)

`pref.list.fmt`
Default *format string* (*columns*) to use in formatted lists.

Default: `NLC` (node name, labels and binding)

`pref.list.actionable_type`
How to render *actionable text* in formatted lists.

Possible values:

- `none`: don’t make text actionable
- `link`: let the terminal alone handle the rendering, typically with a dashed line bellow the text to link
- `alt`: append an alternative actionable view next to the text

Default: `link`

`pref.list.multi`
Whether to allow multiple-line cells in formatted lists.

May improve legibility when focusing on properties that have multiple values like registers or dependencies (*required-by* and *depends-on*).

Default: No

Formatted Trees These preferences configure *formatted trees*.

pref.tree.headers
Whether to show the header row in 2-sided views.

Default: Yes

pref.tree.place_holder
Place holder text for missing values.

Default: `${wchar.ellipsis}`

pref.tree.fmt
Default *format string* (*columns*) to use in formatted trees.

Default: `Nd` (node name and description)

pref.tree.actionable_type
How to render *actionable text* in formatted trees.

Possible values:

- `none`: don't make text actionable
- `link`: let the terminal alone handle the rendering, typically with a dashed line bellow the text to link
- `alt`: append an alternative actionable view next to the text

Default: `none`

pref.tree.cb_anchor
Symbol used to anchor child-bindings to their parents in formatted trees.

Unset (left blank) to disable, or set e.g. to "+" to avoid Unicode.

Default: `${wchar.arrow_right_hook}`

File System Access

pref.fs.hide_dotted
Whether to hide files and directories whose name starts with ".".

These are commonly hidden file system entries on POSIX-like systems.

Default: Yes

pref.fs.no_spaces
Whether to forbid spaces in paths when redirecting commands output to files.

Default: Yes

pref.fs.no_overwrite
Whether to forbid command output redirection to overwrite existing files.

Default: Yes

Other Preferences

pref.always_longfmt
Whether to assume the flag *use a long listing format* (`-l`) is always set (see *Format Commands Output*).

Default: No

pref.sizes_si
Whether to print sizes in SI units (bytes, kB, MB, GB).

Sizes are otherwise printed in hexadecimal format.

Default: Yes

pref.hex_upper
Whether to print hexadecimal digits upper case (e.g. `0XFF` rather than `0xff`).

May improve legibility or accessibility for some users.

Default: No

Builtin Commands This is the reference manual for DTSh builtin commands.

cd **cd** Change the current working branch.

Synopsis:

```
cd [PATH]
```

PATH
Devicetree path to the destination branch (see *Hierarchical File System Metaphor*).

If unset, defaults to the devicetree root.

```
/
> cd &i2c0

/soc/i2c@40003000
/
> cd

/
>
```

pwd **pwd** Print path of the current working branch.

Tip: Use `ls -ld` instead to quickly get detailed information about the current working branch:

```
/soc/flash-controller@4001e000/flash@0/partitions/partition@0
> pwd
/soc/flash-controller@4001e000/flash@0/partitions/partition@0

/soc/flash-controller@4001e000/flash@0/partitions/partition@0
> ls -ld
 Name         Labels          Binding
 ═══════════════════════════════════════════════════════════════════════════
 partition@0  boot_partition  Each child node of the fixed-partitions node represents…
```

---

ls **ls** List branch contents.

Synopsis:

```
ls [OPTIONS] [PATH ...]
```

PATH
Paths of the devicetree branches to list the contents of.

```
/
> ls leds pwmleds
leds:
led_0
led_1
led_2
led_3

pwmleds:
pwm_led_0
```

If the `-d` option is set, paths to the nodes to list.

Paths support wild-cards (*globbing*) in the node name:

```
/
> ls soc/timer* -d
soc/timer@e000e010
soc/timer@40008000
soc/timer@40009000
soc/timer@4000a000
soc/timer@4001a000
soc/timer@4001b000
```

Options

-d

List nodes, not branch contents.

```
/pin-controller
> ls --format NC
 Name           Binding
 ═══════════════════════════════════════════════════════════
 uart0_default  nRF pin controller pin configuration state nodes.
 uart0_sleep    nRF pin controller pin configuration state nodes.
 uart1_default  nRF pin controller pin configuration state nodes.

/ pin-controller
> ls --format NC -d
 Name            Binding
 ═══════════════════════════════
 pin-controller  nordic,nrf-pinctrl
```

-l

Use a *long listing format*.

The default *format string* is configured by the `pref.list.fmt` preference.

See also *Formatted Lists*.

-r

Reverse command output, usually combined with `--order-by`.

See *Sort Directions*.

-R

List recursively.

```
/
> ls /soc/flash-controller@4001e000/flash@0/partitions -R
/soc/flash-controller@4001e000/flash@0/partitions:
partition@0
partition@c000
```

(continues on next page)

(continued from previous page)

```
partition@82000
partition@f8000

/soc/flash-controller@4001e000/flash@0/partitions/partition@0:

/soc/flash-controller@4001e000/flash@0/partitions/partition@c000:

/soc/flash-controller@4001e000/flash@0/partitions/partition@82000:

/soc/flash-controller@4001e000/flash@0/partitions/partition@f8000:
```

In the output above, nodes `partition@0` to `partition@f8000` appear as *empty directories*.

`--enabled-only`
Filter out disabled nodes or branches.

`--fixed-depth DEPTH`
Limit devicetree depth when listing recursively.

`--format FMT`
Node output format (see *Formatted Lists*).

`--order-by KEY`
Sort nodes or branches.

This sets the *order relationship*.

`--pager`
Page command output (see *The Pager*).

Examples Open a quick view of the whole devicetree in the pager (press q to quit the pager):

```
/
> ls -Rl --pager
```

Basic SoC memory layout, highest to lowest addresses:

```
/
> ls soc --format nr --order-by r -r
 Name                  Registers
 ═══════════════════════════════════════════════════════════════
 interrupt-controller  0xe000e100 (3 kB)
 timer                 0xe000e010 (16 bytes)
 crypto                0x5002b000 (4 kB), 0x5002a000 (4 kB)
 gpio                  0x50000800 (768 bytes), 0x50000300 (512 bytes)
 gpio                  0x50000500 (768 bytes), 0x50000000 (512 bytes)
 spi                   0x4002f000 (4 kB)
 pwm                   0x4002d000 (4 kB)
 qspi                  0x40029000 (4 kB), 0x12000000 (128 MB)
```

Tip: If distinguishing the leading lowercase “e” in `0xe000e100` proves difficult, try setting the `pref_hex_upper` preference to get `0XE000E100` instead (see *Other Preferences*).

tree **`tree`** List branch contents in tree-like format.

Synopsis:

```
tree [OPTIONS] [PATH ...]
```

PATH
: *Paths* of the devicetree branches to walk through.

```
/
> tree &i2c0 &i2c1
&i2c0
└── bme680@76

&i2c1
```

Paths support wild-cards (*globbing*) in the node name:

```
/
> tree soc/i2c*
soc/i2c@40003000
└── bme680@76

soc/i2c@40004000
```

Options

-l
: Use a *long listing format*.

The default *format string* is configured by the `pref.tree.fmt` preference.

See also *Formatted Trees*.

-r
: Reverse command output, usually combined with `--order-by`.

See *Sort Directions*.

--enabled-only
: Filter out disabled nodes and branches.

--fixed-depth DEPTH
: Limit the devicetree depth.

```
/
> tree soc --fixed-depth 2
soc
├── interrupt-controller@e000e100
├── timer@e000e010
├── ficr@10000000
├── uicr@10001000
├── memory@20000000
├── clock@40000000
├── power@40000000
│   ├── gpregret1@4000051c
│   └── gpregret2@40000520
```

--format FMT
: Node output format (see *Formatted Trees*).

--order-by KEY
: Sort nodes or branches.

This sets the *order relationship*.

--pager
: Page command output (see *The Pager*).

Examples Open a tree view of the whole devicetree in the pager (press q to quit the pager):

```
/
> tree --format NKYC --pager
```

Tree-view of Flash memory:

```
/
> tree &flash_controller --format Nrc
                            Registers          Compatible
                            ═══════════════════════════════════════════
flash-controller@4001e000   0x4001e000 (4 kB)  nordic,nrf52-flash-controller
└── flash@0                 0x0 (1 MB)         soc-nv-flash
    └── partitions                             fixed-partitions
        ├── partition@0     0x0 (48 kB)
        ├── partition@c000  0xc000 (472 kB)
        ├── partition@82000 0x82000 (472 kB)
        └── partition@f8000 0xf8000 (32 kB)
```

Highlight child-bindings:

```
/
> tree pin-controller --format nXC
                Binding Depth  Binding
                ═══════════════════════════════════════════════════════
pin-controller          0      nordic,nrf-pinctrl
├── uart0_default       1      + nRF pin controller pin configuration state nodes.
│   ├── group1          2        + nRF pin controller pin configuration group.
│   └── group2          2        + nRF pin controller pin configuration group.
```

Tip: In the command output above, "+" has been substituted for the default Unicode *Rightwards arrow with hook* (U+21B3) by setting the user preference `pref.tree.cb_anchor`.

find **find** Search branches for nodes.

Synopsis:

```
find [OPTIONS] [PATH ...]
```

PATH
: *Paths* of the devicetree branches to search.

```
/
> find &i2c0 &i2c1 -l
 Name          Labels             Binding
 ═══════════════════════════════════════════
 i2c@40003000  i2c0, arduino_i2c  nordic,nrf-twi
 bme680@76     bme680_i2c         bosch,bme680
 i2c@40004000  i2c1               nordic,nrf-twi
```

Paths support wild-cards (*globbing*) in the node name:

```
/
> find soc/i2c*
soc/i2c@40003000
soc/i2c@40003000/bme680@76
soc/i2c@40004000
```

Options

-E
: *Text patterns* are interpreted as regular expressions.

-i
: Ignore case in *Text patterns*.

```
/
> find -E -i --also-known-as "green.*led" --format pK
 Path         Also Known As
 ═══════════════════════════════════════════════════════════
 /leds/led_0  Green LED 0, led0, led0, bootloader-led0, mcuboot-led0
 /leds/led_1  Green LED 1, led1, led1
 /leds/led_2  Green LED 2, led2, led2
 /leds/led_3  Green LED 3, led3, led3
```

-l
: Use a *long listing format*.

  The default *format string* is configured by the `pref.list.fmt` preference, or `pref.tree.fmt` when `-T` is set.

-r
: Reverse command output, usually combined with `--order-by`.

  See *Sort Directions*.

-T
: List results in tree-like format.

```
/soc
> find -T --also-known-as partition --format NK
                               Also Known As
                               ═══════════════════════
soc
└═ flash-controller@4001e000  flash_controller
  └═ flash@0                  flash0
    └═ partitions
       ├═ partition@0         mcuboot, boot_partition
       ├═ partition@c000      image-0, slot0_partition
       ├═ partition@82000     image-1, slot1_partition
       └═ partition@f8000     storage, storage_partition
```

--count
: Print matches count.

```
/ soc
> find --also-known-as partition --count
./flash-controller@4001e000/flash@0/partitions/partition@0
./flash-controller@4001e000/flash@0/partitions/partition@c000
./flash-controller@4001e000/flash@0/partitions/partition@82000
./flash-controller@4001e000/flash@0/partitions/partition@f8000
```

(continues on next page)

(continued from previous page)

```
Found: 4
```

--enabled-only
Filter out disabled nodes and branches.

--format FMT
Node output format (see *Formatted Lists*, or *Formatted Trees* if `-T` is set).

--order-by KEY
Sort nodes or branches.

This sets the *order relationship*.

--pager
Page command output (see *The Pager*).

--on-bus PATTERN
Match `PATTERN` with the bus a nodes appears on (is connected to).

See *Text Patterns*.

--also-known-as PATTERN
Match `PATTERN` with device labels, DTS labels and node aliases.

See *Text Patterns*.

--chosen-for PATTERN
Match `PATTERN` with chosen nodes.

See *Text Patterns*.

See also the *chosen* command.

--with-alias PATTERN
Match `PATTERN` with node aliases.

See *Text Patterns*.

See also the *alias* command.

--with-binding PATTERN
Match `PATTERN` with the node's binding.

Both the binding's compatible string and description headline are possible matches:

```
/
> find --with-binding DMA --format Kd
Also Known As          Description
═══════════════════════════════════════════════════════════════
uart0                  Nordic nRF family UARTE (UART with EasyDMA)
uart1, arduino_serial  Nordic nRF family UARTE (UART with EasyDMA)
spi3, arduino_spi      Nordic nRF family SPIM (SPI master with EasyDMA)
```

See *Text Patterns*.

--with-binding-depth EXPR
Match `EXPR` with the child-binding depths.

The child-binding depth associated to a node represents *how far* we should walk the devicetree backward until the specifying binding is not a child-binding:

- 0: top level binding
- 1: child-binding
- greater than 1: nested child-binding

Nodes whose binding is a child-binding:

```
/
> find --with-binding-depth >0 --format NXd
Name             Binding Depth  Description
═══════════════════════════════════════════════════════════════════════════════
partition@0            1        Each child node of the fixed-partitions node represents…
partition@c000         1        Each child node of the fixed-partitions node represents…
partition@82000        1        Each child node of the fixed-partitions node represents…
partition@f8000        1        Each child node of the fixed-partitions node represents…
uart0_default          1        nRF pin controller pin configuration state nodes.
group1                 2        nRF pin controller pin configuration group.
group2                 2        nRF pin controller pin configuration group.
```

See *Integer Expressions*.

--with-bus PATTERN
Match PATTERN with the bus protocols a node supports (provides).

See *Text Patterns*.

--with-compatible PATTERN
Match PATTERN with the node's compatible strings.

See *Text Patterns*.

--with-description PATTERN
Grep the binding's description (not only the headline) for PATTERN.

```
/
> find --with-description gpio -i --format Nd
Name             Description
═══════════════════════════════════════════════════════════════════════════════
radio@40001000   Nordic nRF family RADIO peripheral…
gpiote@40006000  NRF5 GPIOTE node
gpio@50000000    NRF5 GPIO node
gpio@50000300    NRF5 GPIO node
leds             This allows you to define a group of LEDs. Each LED in the group is…
led_0            GPIO LED child node
led_1            GPIO LED child node
buttons          Zephyr Input GPIO KEYS parent node…
button_0         GPIO KEYS child node
button_1         GPIO KEYS child node
connector        GPIO pins exposed on Arduino Uno (R3) headers
```

See *Text Patterns*.

--with-device-label PATTERN
Match PATTERN with device labels.

```
/
>
find --with-device-label "Push butt" --format Nld
Name      Label                Description
═══════════════════════════════════════════════════
button_0  Push button switch 0  GPIO KEYS child node
button_1  Push button switch 1  GPIO KEYS child node
```

See *Text Patterns*.

--with-irq-number EXPR
Match EXPR with the generated interrupts' numbers.

See *Integer Expressions*.

--with-irq-priority EXPR
Match EXPR with the generated interrupts' priorities.

See *Integer Expressions*.

--with-label PATTERN
Match PATTERN with the node's DTS labels.

See *Text Patterns*.

--with-name PATTERN
Match PATTERN with the node's name.

See *Text Patterns*.

--with-reg-addr EXPR
Match EXPR with the register addresses.

See *Integer Expressions*.

--with-reg-size EXPR
Match EXPR with the register sizes.

See *Integer Expressions*.

--with-status PATTERN
Match PATTERN with the node's status string.

```
/
> find --NOT --with-status okay --format Ns
Name                 Status
━━━━━━━━━━━━━━━━━━━━━━━━━━━
timer@e000e010       disabled
spi@40003000         disabled
i2c@40004000         disabled
timer@40008000       disabled
```

See *Text Patterns*.

--with-unit-addr EXPR
Match PATTERN with the node's unit address.

See *Integer Expressions*.

--with-unit-name PATTERN
Match PATTERN with the node's unit name.

See *Text Patterns*.

--with-vendor PATTERN
Match PATTERN with the device vendors.

Both vendor's name and prefix are possible matches.

See *Text Patterns*.

--OR
Evaluate the criterion chain a logical disjunction.

If unset, a logical conjunction is assumed.

See see *Criterion Chains*.

--NOT
Negate the criterion chain.

See *Criterion Chains*.

Tip: Criteria start with `--with-` unless another term really seems more natural, e.g. `--also-known-as` or `--on-bus`.

Examples Search the devicetree for supported bus protocols and connected devices:

```
/soc
> find --with-bus * --OR --on-bus * --enabled-only --format NYC -T
                      Bus      Binding
                      ━━━━━━━━━━━━━━━━━━━━━━━
soc
├── uart@40002000     uart     nordic,nrf-uarte
├── i2c@40003000      i2c      nordic,nrf-twi
│   └── bme680@76     on i2c   bosch,bme680
├── spi@40004000      spi      nordic,nrf-spi
│   └── bme680@0      on spi   bosch,bme680
├── usbd@40027000     usb      nordic,nrf-usbd
├── qspi@40029000     qspi     nordic,nrf-qspi
│   └── mx25r6435f@0  on qspi  nordic,qspi-nor
└── spi@4002f000      spi      nordic,nrf-spim
```

Find devices compatible with BME sensors:

```
/
> find --with-compatible bme --format NCY
Name       Binding       Buses
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
bme680@76  bosch,bme680  on i2c
bme680@0   bosch,bme680  on spi
```

Find devices that may generate the interrupt with IRQ number 0:

```
/
> find --with-irq-number 0 --format Nid
Name            IRQs  Description
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
clock@40000000  0:1   Nordic nRF clock control node
power@40000000  0:1   Nordic nRF power control node
```

Search for *large* memory resources:

```
/
> find --with-reg-size >512k --format NrC --order-by s -r
Name           Registers                                  Binding
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
qspi@40029000  0x12000000 (128 MB), 0x40029000 (4 kB)  nordic,nrf-qspi
flash@0        0x0 (1 MB)                                 soc-nv-flash
```

alias **alias** List aliased nodes.

Synopsis:

```
alias [OPTIONS] [NAME]
```

NAME

The alias name to search for.

Partial matches are also answered: a `led` parameter value would match aliases “led0”, “led1”, etc.

If not set, **alias** will enumerate all aliased nodes.

See also DTSpec 3.3 /aliases node.

Options

-l
Use a *long listing format*.

Default to `pC` when `--format FMT` is not set.

--enabled-only
Filter out disabled nodes.

--format FMT
Node output format (see *Formatted Lists*).

Examples Led devices are usually aliased:

```
/
>
alias -l led
                    Path                Binding
                    ═══════════════════════════════════
led0            ->  /leds/led_0         GPIO LED child node
led1            ->  /leds/led_1         GPIO LED child node
led2            ->  /leds/led_2         GPIO LED child node
led3            ->  /leds/led_3         GPIO LED child node
pwm-led0        ->  /pwmleds/pwm_led_0  PWM LED child node
bootloader-led0 ->  /leds/led_0         GPIO LED child node
mcuboot-led0    ->  /leds/led_0         GPIO LED child node
```

Tip: In the command output above, "->" has been substituted for the default Unicode *Rightwards arrow* (`U+2192`) by setting the user preference `wchar.arrow_right`.

chosen **chosen** List chosen nodes.

Synopsis:

```
chosen [OPTIONS] [NAME]
```

NAME
The chosen name to search for.

Partial matches are also answered: a `sram` parameter value would match the chosen name "zephyr,sram".

If not set, **chosen** will enumerate all chosen nodes.

See also DTSpec 3.6 /chosen node.

Options

-l
Use a *long listing format*.

Default to `NC` when `--format FMT` is not set.

--enabled-only
Filter out disabled nodes.

--format FMT
Node output format (see *Formatted Lists*).

Examples Get the source of entropy:

```
/
> find --chosen-for entropy -l
Name             Labels  Binding
═══════════════════════════════════
random@4000d000  rng     nordic,nrf-rng
```

Find RAM size:

```
/
> chosen ram --format nrd
               Name    Registers           Description
               ═══════════════════════════════════════════════════
zephyr,sram -> memory  0x20000000 (256 kB)  Generic on-chip SRAM description
```

Textual User Interface This Devicetree shell should come as no surprise to those familiar with CLI applications:

- a prompt with some state information
- command line auto-completion and history
- *standard* keybindings

Additionally, DTSh also supports:

- consistent styles and colors, configurable with user themes
- integration with the system pager

Tip: Command line history and auto-completion helps:

- typing only few keystrokes
- discovering and memorizing the commands and their options

The Prompt Once its configuration is loaded, and the devicetree model is initialized, DTSh will clear the Terminal and start an interactive session with the root node as current working branch:

```
dtsh (0.0.99): Shell-like interface with Devicetree
How to exit: q, or quit, or exit, or press Ctrl-D

/
> cd &i2c0

/soc/i2c@40003000
>
```

The prompt contains some basic state information:

- the current working branch is reminded above the actual prompt line
- the prompt changes appearance (e.g. turns red) when a command fails, and returns to its default appearance when a command succeeds

Convenient key bindings are available when entering commands at the prompt:

- *Moving and Editing*
- *Command Line History*

Tip:

The default prompt character is actually *Medium Right-Pointing Angle Bracket Ornament* (U+276D): the rationale for choosing an *exotic* character is to better distinguish the Devicetree shell prompt from the OS shell.

If that does not work for you, just change it in your preferences, e.g.:

```
# Prompt used for the examples in this documentation.
prompt.wchar = >

# Note: the dollar sign is used for interpolation, and must be escaped.
prompt.wchar = $$
```

Refer to the *prompt preferences* for details.

Note: On POSIX-like systems, Ctrl-D (at the beginning of an empty line) closes the DTSh session by *signaling* the end of the input *file* (EOF).

On Windows systems, it seems that you instead have to *send* Ctrl-Z to the Terminal input to achieve the same signaling.

Auto-completion Auto-completion is triggered by first pressing the TAB key twice ([TAB][TAB]) at the end of a word or after a space: this builds and shows the list of completion candidates (also termed *completion states*).

Auto-completion is contextual, and may propose:

- commands
- command options
- valid values for command arguments (e.g. compatible strings or order relationships)
- valid values for command parameters (e.g. devicetree paths or alias names)
- files or directories (when redirecting commands output)

If there's is no candidate, or if the command line is otherwise invalid (e.g. undefined command), [TAB][TAB] does nothing. If there's a single match, it may be substituted for the word to complete (e.g. DTS labels).

Each subsequent [TAB] will navigate the completion states, until the command line changes, which terminates the auto-completion context.

Auto-complete Commands Triggering auto-completion anywhere from an empty command line will list all DTSh built-in commands:

```
/
> [TAB][TAB]
alias     list aliased nodes
cd        change the current working branch
chosen    list chosen nodes
find      search branches for nodes
ls        list branch contents
pwd       print path of current working branch
tree      list branch contents in tree-like format
```

If auto-completion is triggered at the end of the first word in the command line, only matching commands are proposed:

```
/
> c[TAB][TAB]
cd        change the current working branch
chosen    list chosen nodes
```

Auto-complete Options Auto-completion for command options is available when the command line starts with a valid command name.

Then, triggering auto-completion for a word that starts with `-` will list:

- all the command options if the word to complete is `-`, all options with a long name if the word to complete is `--`
- matching options otherwise

```
/
> alias -[TAB][TAB]
-h --help         print command help
-l                use a long listing format
--enabled-only    filter out disabled nodes or branches
--format FMT      node output format

/
> find --with-reg-[TAB][TAB]
--with-reg-addr EXPR    match register addresses
--with-reg-size EXPR    match register sizes
```

Auto-complete Arguments Command arguments are options that expect a value.

Auto-completion for argument values is available when:

- the command line starts with a valid command name
- the word to complete follows an argument name

What auto-completion will propose then depends on the argument's semantic, e.g.:

```
/
> find --with-compatible gpio-[TAB][TAB]
gpio-keys    Zephyr Input GPIO KEYS parent node…
gpio-leds    This allows you to define a group of LEDs. Each LED in the group is…

/
❯ find --with-bus i2[TAB][TAB]
i2c
i2s

/
> find --order-by [TAB][TAB]
a    sort by unit address
A    sort by aliases
B    sort by supported bus protocols
...
s    sort by register sizes
v    sort by vendor name
X    sort by child-binding depth

/
> ls --format [TAB][TAB]
a    unit address
A    node aliases
```

(continues on next page)

(continued from previous page)

```
b    bus of appearance
...
v    vendor name
X    child-binding depth
Y    bus information
```

Auto-complete Parameters Auto-completion for parameter values is available when:

- the command line starts with a valid command name
- the cursor is at a position where a parameter value is expected (e.g. not where an argument's value is expected)

What auto-completion will propose then depends on the parameter's semantic, e.g.:

```
/
> alias led[TAB][TAB]
led0
led1
led2
led3

/soc/flash-controller@4001e000/flash@0
> ls partitions/partition@[TAB][TAB]
partition@0
partition@82000
partition@c000
partition@f8000
```

Auto-completion for devicetree paths also supports DTS labels:

```
/
> ls &i2c[TAB][TAB]
i2c0            Nordic nRF family TWI (TWI master)…
i2c0_default    nRF pin controller pin configuration state nodes.
i2c0_sleep      nRF pin controller pin configuration state nodes.
i2c1            Nordic nRF family TWI (TWI master)…
i2c1_default    nRF pin controller pin configuration state nodes.
i2c1_sleep      nRF pin controller pin configuration state nodes.
```

Auto-complete Redirection Paths Auto-completion for file system paths is available when:

- the command line starts with a valid command name
- the cursor is at a position where DTSh expects the path of the file the command's output should be redirected to (see *Output Redirection*)

Assuming the current working directory is the root of a Zephyr application project:

```
/
> ls soc -lR > [TAB][TAB]
boards/
build/
src/
CMakeLists.txt
README.rst
prj.conf
sample.yaml
```

Tip:

- A leading "~" is interpreted as the user directory: `~/Doc` would auto-complete to e.g. `/home/myself/Documents`
- By default, files and directories whose name starts with "." are commonly hidden on POSIX-like systems, and DTSh follows this convention.

  To include these files and directories in the auto-completion candidates, unset `pref.fs.hide_dotted` (see *File System Access*).

Command History The command line history is accessible with intuitive keybindings:

- `Up` to navigate the history backward
- `Down` to navigate the history forward
- `Ctrl-R` to *reverse* search the history

Refer to the *keybindings for command line history* for more keyboard shortcuts.

The history persists across DTSh sessions.

The command history honors the `HISTCONTROL` and `HISTIGNORE` environment (see Bash History Facilities for details).

Tip: The `history` file is located in the DTSh application data directory for the current user (*User Files*).

It's a simple text file you may edit or remove.

The Pager Paging basically means viewing the command's output *page by page*, as if it were the content of a file.

This is almost necessary when a command would output far more lines than what the Terminal could show at once: for example, the output of `ls -R --pager`, that recursively (`-R`) list all devicetree nodes, would be painful to *understand* without the `--pager` option.

System pagers have their own textual user interface, and offer other (basic) features besides paging.

POSIX

On POSIX-like systems (GNU Linux, macOS) the pager is typically `less`, which has simple key bindings, and will preserve formatting, colors and styles.

Useful key bindings include:

Table 10: Pager key bindings (`less`)

Down, Page Down	Forward one line, one window
Up, Page Up	Backward one line, one window
`g`	Go to first line
`G`	Go to last line
`/ pattern`	Search forward for `pattern`
`h`	Display the pager help
`q`	Exit the pager and returns to DTSh prompt

Windows

On Windows, the system pager is typically `more`, which is usually less convenient than `less` (sic):

- doesn't support backward movements, and will exit once the end of the command output is reached

- styles and colors may mess with formatting

`h` should display the pager help, and `q` return to the DTSh prompt.

If unhappy with the default pager, try installing GNU less compiled for Windows.

Tip: Pagers work vertically: the command output is paged only vertically, and movements are vertical.

To get *something* that you can also scroll horizontally, consider redirecting the command output to an HTML file, e.g.:

```
ls -R --format NcrYd > board.html
```

Actionable Text An *actionable text* is a TUI component that associates an action to open a resource (a link) with the default system *viewer* for the resource's type, e.g.:

- a text editor for DTS or YAML files
- a Web browser for external links

Although all modern Terminal applications should support these hyperlinks, their appearance and how to *action* them will vary. Typically:

- linked text (ANSI *tags*) will appear with a (possibly dashed) underline bellow, somewhat imitating usual hyperlinks in Web browsers; hoovering the linked text will reveal the link destination, e.g. with a tooltip
- clicking the text while holding the `Ctrl` key will open the link destination

---

Tip:

- compatible strings and binding descriptions are actionable: this permits to easily locate and open related binding files
- if the default rendering eventually feels too invasive, try configuring alternative views for hyperlinks when they appear in *lists* or *trees*
- links may open in inappropriate applications, or not open at all (e.g. files that end with `.dts` may be associated with DTS-encoded audio content): in this case, configure MIME types and associated applications at the operating system level

---

Key Bindings DTSh relies on the standard keybindings provided by the GNU Readline library, described in the Bindable Readline Commands documentation.

For consistency with this reference documentation (and other material like the GNU Emacs Reference Card), we'll use the GNU convention:

- *modifiers* are represented by single uppercase letters, e.g. `C` for *Control* or `M` for *Meta*
- keys are represented by lowercase letters

`C-a` is then the GNU writing for `Ctrl-A`.

PC

- the *Control* modifier is usually bound to `Ctrl Left`, i.e. the `Ctrl` key located at left of the keyboard
- the *Meta* modifier is usually bound to `Alt Left`, i.e. the `Alt` key located at left of the keyboard

macOS

The default configuration for modifier keys may not work well with the GNU Readline keybindings: e.g. the *Meta* modifier is likely bound to `ESC`.

Terminal.app and iTerm2 users can easily remap the modifiers with a more convenient configuration: it seems common to e.g. remap *Meta* to `Option` with some settings like *Use Option as Meta key*.

Moving and Editing By default, moving and editing the command line happens in Emacs mode, and the keybindings mentioned here are those of GNU Emacs (and Bash, GDB, etc).

The tables below show the most commonly used ones.

Table 11: Moving

`Home`	`C-a`	Move cursor to beginning of command line.
`End`	`C-e`	Move cursor to end of command line.
`Right`	`C-f`	Move forward a character.
`Left`	`C-b`	Move back a character.
`Ctrl-Right`	`M-f`	Move forward to the end of the next word (letters and digits).
`Ctrl-Left`	`M-b`	Move back to the start of the current or previous word.
	`C-l`	Clear the screen, then redraw the current line.

Table 12: Changing text

`C-t`	Drag the character before the cursor forward over the character at the cursor, moving the cursor forward as well (*transpose chars*).
`M-t`	Drag the word before point past the word after point, moving point past that word as well (*transpose words*).
`C-d`	Delete the following character.
`M-d`	Kill (*cut*) from point to the end of the current word.
`C-k`	Kill the text from point to the end of the line.
`C-u`	Kill backward from the cursor to the beginning of the current line.
`C-w`	Kill the word behind point.
`C-y`	Yank (*paste*) the top of the kill ring into the buffer at point.
`M-y`	Rotate the kill-ring, and yank the new top (must follow `C-y`).
`C-_`	Undo some previous changes. Repeat this command to undo more changes.

Note: Although the Readline library comes with a set of Emacs-like keybindings installed by default, it is possible to use different keybindings: see the Readline Init File documentation.

Command Line History Bellow are the most common keybindings for *invoking* the command history (see Commands For Manipulating The History for a complete list).

Table 13: Command History

`Return`		Accept the line regardless of where the cursor is.
`Up`	`C-p`	Move back through the history list, fetching the previous command.
`Down`	`C-n`	Move forward through the history list, fetching the next command.
	`M-<`	Move to the first line in the history.
	`M->`	Move to the end of the input history, i.e., the line currently being entered.
	`C-r`	Search backward starting at the current line and moving up through the history as necessary. This is an *incremental search*.

Tip: An *incremental search* updates its results while typing:

- initiate incremental search, e.g. backward with `C-r`: a sub-prompt like `(reverse-i-search):` should replace the DTSh prompt
- the string the prompt expects is the pattern to search the command history for: while typing the piece of command line to search for, the sub-prompt will update to show the most recent matches
- repeat `C-r` to navigate the history backward as needed
- eventually, press the `Left` or `Right` key to select the command line at the sub-prompt, and return to the DTSh prompt for further edits
- or press `Return` to re-execute the command line selected at the sub-prompt, or `C-g` to abort the search and return to the DTSh prompt

User Themes A *theme* is a collection of *styles*, each of which is consistently used to represent a type of information: e.g. by default compatible strings are always green, and things that behave like *symbolic links* (e.g. aliases) are all italics.

A style has the form `[STYLE] [FG] [on BG]` where:

- `STYLE` is an optional style applied to the text, e.g. "bold" or "italic"
- `FG` is the text foreground color, e.g. "dark_green" or "#005f00"
- `BG` is the text background color

Colors and such are subjective, and a theme that pleases one person may cause accessibility issues (or headaches) to another. To address this legit concern, DTSh supports user themes that will override the default styles.

Styles are loaded in that order:

1. DTSh loads default values for all defined styles
2. if a user theme file exists, e.g. `~/.config/theme.ini` on GNU Linux, it's loaded, overriding the default styles
3. if an additional theme file is specified with the West command option `--theme`, it's eventually loaded, overriding previous styles

See *User Files* to initialize a theme file template at the proper location.

Comments in the template file should be sufficient to understand how to write styles, and identify which style applies to which type of information. The INI syntax itself is reminded in *Configuration Format*.

2.11.13 History and Motivation

West was added to the Zephyr project to fulfill two fundamental requirements:

- The ability to work with multiple Git repositories
- The ability to provide an extensible and user-friendly command-line interface for basic Zephyr workflows

During the development of west, a set of *Design Constraints* were identified to avoid the common pitfalls of tools of this kind.