Zephyr® Project
Developer Summit 2022

Zephyr Device Mysteries, Solved

Martí Bolívar
marti.bolivar@nordicsemi.no

Nordic Semiconductor

9 June 2022

Assumptions about you

You know devicetree

You know devicetree

If not:

`docs.zephyrproject.org/latest/build/dts/intro.html`

<zephyr/devicetree.h>

`<zephyr/devicetree.h>`
has betrayed you

```
src/main.c:14:14: error:
'__device_dts_ord_DT_N_S_buttons_S_button_0_P_gpos_IDX_0_PH_ORD'
undeclared here (not in a function)
   14 |  DT_FOREACH_CHILD(BUTTONS_NODE, GPIO_DT_SPEC_AND_COMMA)
      |              ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
src/main.c:14:86: error:
'DT_N_S_buttons_S_button_0_P_gpos_IDX_0_VAL_pin' undeclared here
(not in a function)
   14 |  DT_FOREACH_CHILD(BUTTONS_NODE, GPIO_DT_SPEC_AND_COMMA)

|
                     ^
src/main.c:14:164: error:
'__device_dts_ord_DT_N_S_buttons_S_button_1_P_gpos_IDX_0_PH_ORD'
undeclared here (not in a function)
   14 |  DT_FOREACH_CHILD(BUTTONS_NODE, GPIO_DT_SPEC_AND_COMMA)

|

                                  ^
src/main.c:14:236: error:
'DT_N_S_buttons_S_button_1_P_gpos_IDX_0_VAL_pin' undeclared here
(not in a function)
   14 |  DT_FOREACH_CHILD(BUTTONS_NODE, GPIO_DT_SPEC_AND_COMMA)
      |
```

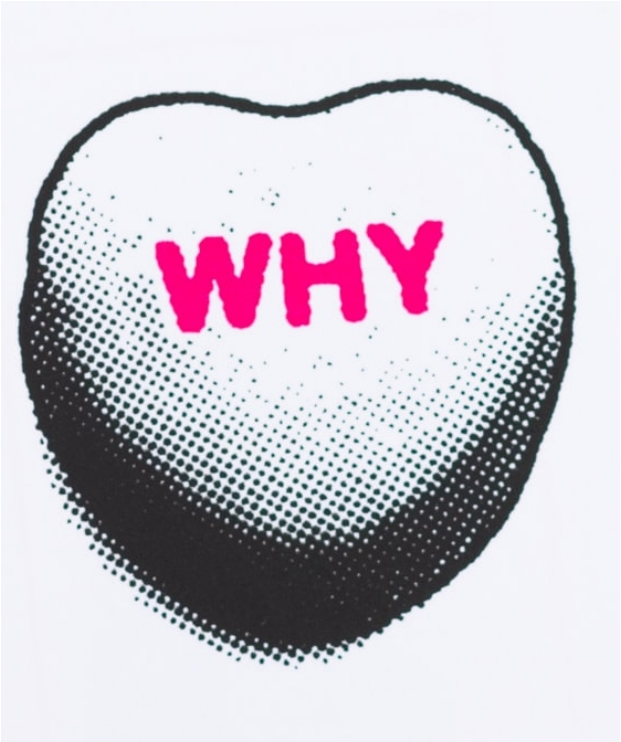


Photo by Patrick Perkins

```c
struct dt_property { ... };

struct dt_node {
        const char *name;
        struct dt_property *properties;
        struct dt_node *children;
};
```

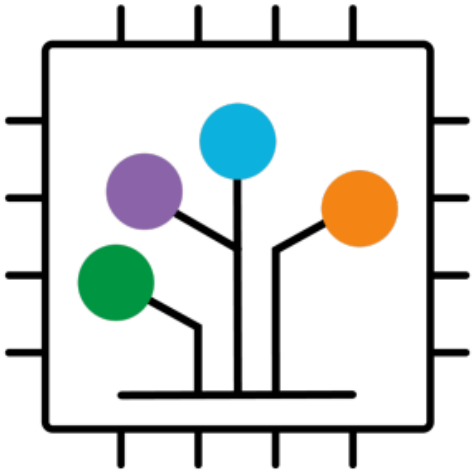
```
dt_property { ... };

struct dt_
        const
        _property
        uct dt_node *children;
```

- Wasted .rodata
- Wasted .text
- Runtime overhead

Instead:

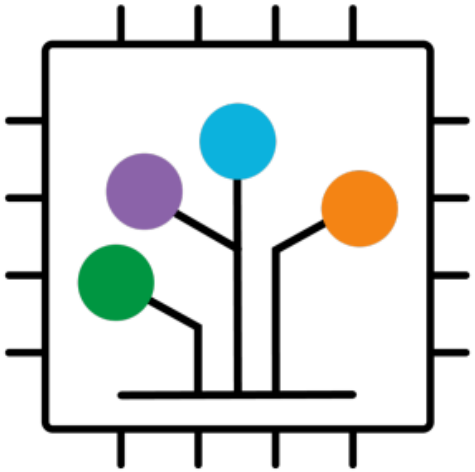
- DT specification
- Bindings
- Build system magic
- Macrobatics



Devicetree Specification

Release v0.3

devicetree.org



Devicetree Specification

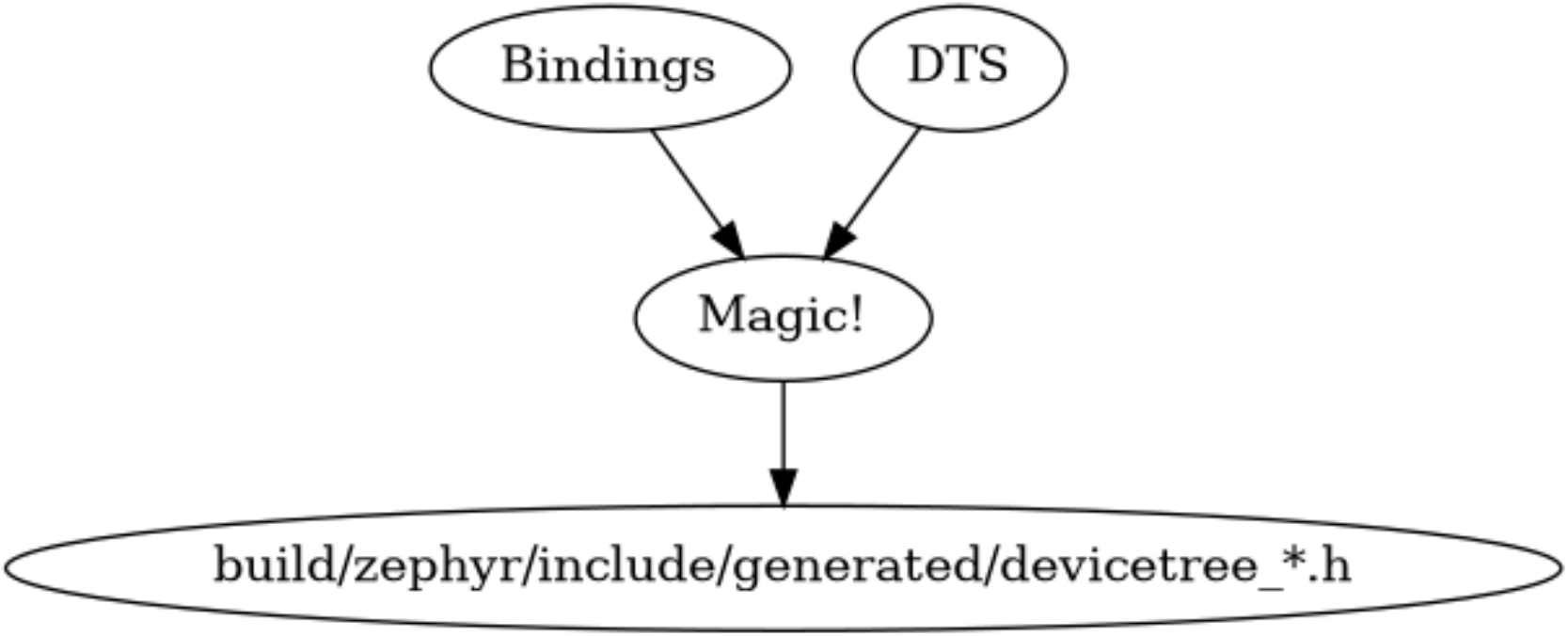
Release v0.3

devicetree.org

Bindings: schemas for nodes

vnd,device.yaml

docs.zephyrproject.org/latest/build/dts/bindings.html


Bindings
DTS
Magic!
build/zephyr/include/generated/devicetree_*.h

- `devicetree_fixups.h`: legacy
- `devicetree_unfixed.h`: your DTS, in C
- `devicetree_extern.h`: possible DTS devices

Macrobatics

Warm up

```
#define PASTE(a, b) a ## b
```

Warm up

```
#define PASTE(a, b) a ## b

PASTE(FOO, BAR)
```

Warm up

```
#define PASTE(a, b) a ## b

PASTE(FOO, BAR) → FOOBAR
```

Warm up

```
#define FOOBAR 42
#define PASTE(a, b) a ## b

PASTE(FOO, BAR) → FOOBAR
```

Warm up

```
#define FOOBAR 42
#define PASTE(a, b) a ## b

PASTE(FOO, BAR) → FOOBAR
                → 42
```

Stretch

```
#define PASTE(a, b)  PASTE2(a, b)
#define PASTE2(a, b) a ## b
```

Stretch

```
#define PASTE(a, b)  PASTE2(a, b)
#define PASTE2(a, b) a ## b

PASTE(FOO, BAR)
```

Stretch

```
#define PASTE(a, b)  PASTE2(a, b)
#define PASTE2(a, b) a ## b

PASTE(FOO, BAR) → PASTE2(FOO, BAR)
```

Stretch

```
#define BAR           BAZ
#define PASTE(a, b)   PASTE2(a, b)
#define PASTE2(a, b)  a ## b

PASTE(FOO, BAR) → PASTE2(FOO, BAR)
```

Stretch

```
#define BAR           BAZ
#define PASTE(a, b)   PASTE2(a, b)
#define PASTE2(a, b)  a ## b

PASTE(FOO, BAR) → PASTE2(FOO, BAR)
                → FOOBAZ
```

Stretch

```
#define FOOBAZ        42
#define BAR           BAZ
#define PASTE(a, b)   PASTE2(a, b)
#define PASTE2(a, b)  a ## b

PASTE(FOO, BAR) → PASTE2(FOO, BAR)
                → FOOBAZ
```

Stretch

```
#define FOOBAZ        42
#define BAR           BAZ
#define PASTE(a, b)   PASTE2(a, b)
#define PASTE2(a, b)  a ## b

PASTE(FOO, BAR) → PASTE2(FOO, BAR)
                → FOOBAZ
                → 42
```

Backflip

```
#define PASTE(a, b)  PASTE2(a, b)
#define PASTE2(a, b) a ## b

#define FOOIZE(arg)  PASTE(FOO_, arg)
```

Backflip

```
#define PASTE(a, b)  PASTE2(a, b)
#define PASTE2(a, b) a ## b

#define FOOIZE(arg)  PASTE(FOO_, arg)

#define BAR baz
FOOIZE(BAR) → ...
            → FOO_baz
```

Backflip

```
#define PASTE(a, b)  PASTE2(a, b)
#define PASTE2(a, b) a ## b

#define FOOIZE(arg)  PASTE(FOO_, arg)

#define BAR baz
FOOIZE(BAR)
```

```
FOO_baz
```

Backflip

```
#define PASTE(a, b)  PASTE2(a, b)
#define PASTE2(a, b) a ## b

#define FOOIZE(arg)  PASTE(FOO_, arg)

#define BAR baz
FOOIZE(BAR) → ...
            → FOO_baz



PASTE(FOOIZE(BAR), _1)
```

Backflip

```
#define PASTE(a, b)  PASTE2(a, b)
#define PASTE2(a, b) a ## b

#define FOOIZE(arg)  PASTE(FOO_, arg)

#define BAR baz
FOOIZE(BAR) → ...
            → FOO_baz

#define FOO_baz_1 42
#define FOO_baz_2 43
PASTE(FOOIZE(BAR), _1)
```

Backflip

```
#define PASTE(a, b)  PASTE2(a, b)
#define PASTE2(a, b) a ## b

#define FOOIZE(arg)  PASTE(FOO_, arg)

#define BAR baz
FOOIZE(BAR) → ...
            → FOO_baz

#define FOO_baz_1 42
#define FOO_baz_2 43
PASTE(FOOIZE(BAR), _1) → ...
                       → FOO_baz_1
                       → 42
```

Backflip

```
#define PASTE(a, b)  PASTE2(a, b)
#define PASTE2(a, b) a ## b

#define FOOIZE(arg)  PASTE(FOO_, arg)

#define BAR baz
FOOIZE(BAR) → ...
            → FOO_baz

#define FOO_baz_1 42
#define FOO_baz_2 43
PASTE(FOOIZE(BAR), _1) → ...
                       → FOO_baz_1
                       → 42
PASTE(FOOIZE(BAR), _2) → ...
                       → 43
```

$$\texttt{zephyr.dts} \rightarrow \texttt{devicetree\_unfixed.h}$$

```
/ {
        parent {
                size = <3>;
                child {
                        size = <4>;
                };
        };
};
```

```
/ {
        parent {
                size = <3>;
                child {
                        size = <4>;
                };
        };
};
```

```
/* Maybe maybe maybe */
DT_PROP("/parent", "size") → 3
DT_PROP("/parent/child", "size") → 4
```

```
DT_PROP("/parent", "size")
DT_PROP("/parent/child", "size")
```

Problem: quotes

```
DT_PROP("/parent", "size")
DT_PROP("/parent/child", "size")
```

Solution: macrobatics!

Nodes:

```
/ {
        parent {                /* path: /parent */
                child {         /* path: /parent/child */
                };
        };
};
```

Node identifiers:

```
DT_PATH(parent)        → DT_N_S_parent
DT_PATH(parent, child) → DT_N_S_parent_S_child

                         DT_N means "devicetree node"
                         / becomes _S_
```

Node identifiers

- `DT_N_S_foo_S_bar` means `/foo/bar`
- Lowercase-and-underscore everything from DTS

Node IDs are not values

```
DT_N_S_parent
DT_N_S_parent_S_child
```

Properties:

```
/ {
        parent {
                child {
                        size = <4>;
                };
        };
};
```

devicetree_unfixed.h macros:

```
#define DT_N_S_parent_S_child_P_size 4
```

Properties

- `<node_id>_P_baz`: property baz value
- Types from bindings
- Take only what you need

devicetree_unfixed.h macros:

```
/* devicetree_unfixed.h */
#define DT_N_S_parent_P_size 3
```

Property access with node identifiers:

```
#define PARENT_NODE DT_PATH(parent)

DT_PROP(PARENT_NODE, size) → PARENT_NODE ## _P_ ## size
                           → DT_PATH(parent) ## _P_ ## size
                           → DT_N_S_parent_P_size
                           → 3
```

Docs for v3.1

```
docs.zephyrproject.org/3.1.0/build/dts/api-usage.html#generated-macros
```

Docs example

```
; ---------------------------------------------------------------------
; property-macro: a macro related to a node property
;
; These combine a node identifier with a "lowercase-and-underscores form"
; property name. The value expands to something related to the property's
; value.
;
; The optional prop-suf suffix is when there's some specialized
; subvalue that deserves its own macro, like the macros for an array
; property's individual elements
;
; The "plain vanilla" macro for a property's value, with no prop-suf,
; looks like this:
;
;   DT_N_<node path>_P_<property name>
;
; Components:
;
; - path-id: node's devicetree path converted to a C token
; - prop-id: node's property name converted to a C token
; - prop-suf: an optional property-specific suffix
property-macro =  %s"DT_N" path-id %s"_P_" prop-id [prop-suf]
```

Vague docs?
Ask for clarifications!

$$\texttt{zephyr.dts} \rightarrow \texttt{devicetree\_extern.h}$$

devicetree_extern.h cartoon

```
extern const struct device
        DEVICE_DT_NAME_GET(DT_N),                /* dts_ord_0 */
        DEVICE_DT_NAME_GET(DT_N_S_aliases),      /* dts_ord_1 */
        /* ... */
        DEVICE_DT_NAME_GET(DT_N_S_leds_S_led_0), /* dts_ord_19 */
        DEVICE_DT_NAME_GET(DT_N_S_leds_S_led_1); /* dts_ord_20 */

extern const struct device
        __device_dts_ord_0,
        __device_dts_ord_1,
        /* ... */
        __device_dts_ord_19,
        __device_dts_ord_20;
```

Summary

- `__device_dts_ord_<N>`: device from the node with ordinal `<N>`
- `<zephyr/device.h>` includes this
- `DEVICE_DT_GET(...)` gets addresses from here

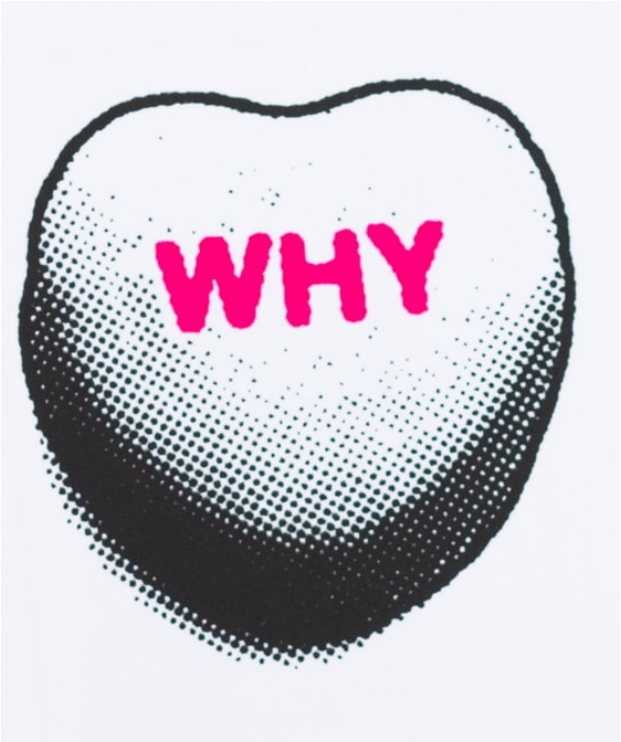


Photo by Patrick Perkins

This breaks user mode

```
extern const struct device DT_N_S_name_S_is_S_too_S_long;
```

Details

```
commit f91e9fba51e5da46ee5c6822f8656713d74a6ecf
Author: Peter Bigot <peter.bigot@nordicsemi.no>
Date:   Sat Jan 23 07:56:09 2021 -0600

    device: fix potential truncation of DT-derived device names
```

undefined reference to '__device_...

Usually, either:

- device was not allocated in a driver
- you have a typo

`__device_dts_ord_105`

- Look up node with ordinal 105 in `devicetree_unfixed.h`
- Find driver that should allocate the device node
- Is driver Kconfig `y`?
- Is the node enabled?

__device_dts_ord_DT_HOT_MESS

```
src/main.c:14:14: error:
'__device_dts_ord_DT_N_S_buttons_S_button_0_P_gpos_IDX_0_PH_ORD'
undeclared here (n
```

Look at the preprocessor output

docs.zephyrproject.org/latest/build/dts/troubleshooting.html

Look at the preprocessor output

To save preprocessor output when using GCC-based toolchains, add `-save-temps=obj` to the `EXTRA_CFLAGS` CMake variable. For example, to build Hello World with west with this option set, use:

```
west build -b BOARD samples/hello_world -- -DEXTRA_CFLAGS=-save-temps=obj
```

This will create a preprocessor output file named `foo.c.i` in the build directory for each source file `foo.c`.

You can then search for the file in the build directory to see what your devicetree macros expanded to. For example, on macOS and Linux, using `find` to find `main.c.i`:

```
$ find build -name main.c.i
build/CMakeFiles/app.dir/src/main.c.i
```

It's usually easiest to run a style formatter on the results before opening them. For example, to use `clang-format` to reformat the file in place:

```
clang-format -i build/CMakeFiles/app.dir/src/main.c.i
```

You can then open the file in your favorite editor to view the final C results after preprocessing.

Questions?