cogeno Documentation

Release 0.2.2

Bobby Noelte

Sep 09, 2020

CONTENTS

CHAPTER

ONE

WELCOME TO COGENO'S DOCUMENTATION!

This documentation is continuously written. It is edited via text files in the reStructuredText markup language and then compiled into a static website/ offline document using the open source Sphinx and Read the Docs tools.

You can contribute to cogeno's dcoumentation by opening GitLab issues or sending patches via merge requests on its GitLab repository.

1.1 Sections

1.2 Indices and tables

1.2.1 Generated Content Index

adc-controller

```
include: adc-controller.yaml
```

```
# Copyright (c) 2017, NXP
# SPDX-License-Identifier: Apache-2.0

# Common fields for ADC controllers

include: base.yaml

properties:
    label:
      required: true

    "#io-channel-cells":
      type: int
      required: true
```

base

```
include: base.yaml
```

```
# Common fields for all devices

properties:
    status:
        type: string
        required: false
        description: indicates the operational status of a device
        enum:
           - "okay"
           - "disabled"
           - "reserved"
           - "fail"
           - "fail-sss"

    compatible:
        type: string-array
        required: true
        description: compatible strings

    reg:
        type: array
        description: register space
        required: false

    reg-names:
        type: string-array
        description: name of each register space
        required: false
```

(continues on next page)

(continued from previous page)

```
    interrupts:
        type: array
        required: false
        description: interrupts for device

    # Does not follow the 'type: phandle-array' scheme, but gets type-checked
    # by the code. Declare it here just so that other bindings can make it
    # 'required: true' easily if they want to.
    interrupts-extended:
        type: compound
        required: false
        description: extended interrupt specifier for device

    interrupt-names:
        type: string-array
        required: false
        description: name of each interrupt

    interrupt-parent:
        type: phandle
        required: false
        description: phandle to interrupt controller node

    label:
        type: string
        required: false
        description: Human readable string describing the device (used by Zephyr for␣
↪API name)

    clocks:
        type: phandle-array
        required: false
        description: Clock gate information

    clock-frequency:
        type: int
        required: false
        description: >
          Frequency of clock in Hz. Encoded as an arbitrary number
          of frequency values.

    clock-accuracy:
        type: int
        required: false
        description: >
          Accuracy of clock in ppb (parts per billion). Encoded as an
          arbitrary number of ppb values.

    "#address-cells":
        type: int
        required: false
        description: number of address cells in reg property

    "#size-cells":
        type: int
        required: false
        description: number of size cells in reg property
```

(continues on next page)

(continued from previous page)

```
    pinctrl-\d+:
        type: phandle
        required: false
        description: pin control for device

    pinctrl-names:
        type: string-array
        required: false
        description: name of each pin control
```

can-controller

```
include: can-controller.yaml
```

```
# Common fields for CAN controllers

include: base.yaml

bus: can

properties:
    "#address-cells":
      required: true
      const: 1
    "#size-cells":
      required: true
      const: 0
    label:
      required: true
    bus-speed:
      type: int
      required: true
      description: bus speed in Baud/s
    sjw:
      type: int
      required: true
      description: Resynchronization jump width (ISO 11898-1)
    prop-seg:
      type: int
      required: true
      description: Time quantums of propagation segment (ISO 11898-1)
    phase-seg1:
      type: int
      required: true
      description: Time quantums of phase buffer 1 segment (ISO 11898-1)
    phase-seg2:
      type: int
      required: true
      description: Time quantums of phase buffer 2 segment (ISO 11898-1)
```

can-device

```
include: can-device.yaml
```

```
# Copyright (c) 2018 Alexander Wachter
# SPDX-License-Identifier: Apache-2.0

# Common fields for CAN devices

include: base.yaml

on-bus: can

properties:
    reg:
      required: true
    label:
      required: true
```

chosen

```
include: chosen.yaml
```

```
# Copyright (c) 2019..2020 Bobby Noelte
# SPDX-License-Identifier: Apache-2.0

description: Chosen properties

compatible: "chosen"

properties:
    stdout-path:
        type: string
        required: false
        description: >
          Device to be used for boot console output
          If the character ":" is present in the value, this terminates the path.
          The meaning of any characters following the ":" is device-specific, and
          must be specified in the relevant binding documentation.
          For UART devices, the preferred binding is a string in the form:
            path:<baud>{<parity>{<bits>{<flow>}}}
          where
            path    - path of uart device
            baud        - baud rate in decimal
            parity      - 'n' (none), 'o', (odd) or 'e' (even)
            bits        - number of data bits
            flow        - 'r' (rts)
          For example: 115200n8r

    zephyr,entropy:
        type: path
        required: false
        description: A device which can be used as a system-wide entropy source.

    zephyr,flash:
```

(continues on next page)

(continued from previous page)

```
        type: path
        required: false
        description: >
          A node whose reg is sometimes used to set the defaults for CONFIG_FLASH_
↪BASE_ADDRESS
          and CONFIG_FLASH_SIZE.

    zephyr,sram:
        type: path
        required: false
        description: >
          A node whose reg sets the base address and size of SRAM memory available to␣
↪the Zephyr
          image, used during linking.

    zephyr,ccm:
        type: path
        required: false
        description: TBD

    zephyr,console:
        type: path
        required: false
        description: TBD

    zephyr,shell-uart:
        type: path
        required: false
        description: TBD

    zephyr,bt-uart:
        type: path
        required: false
        description: TBD

    zephyr,uart-pipe:
        type: path
        required: false
        description: TBD

    zephyr,bt-mon-uart:
        type: path
        required: false
        description: TBD

    zephyr,uart-mcumgr:
        type: path
        required: false
        description: TBD
```

clock-consumer

```
include: clock-consumer.yaml
```

```
# Copyright (c) 2018..2020 Bobby Noelte
# SPDX-License-Identifier: Apache-2.0

# -- Assigned clock parents and rates --
# Some platforms may require initial configuration of default parent clocks
# and clock frequencies. Such a configuration can be specified in a device tree
# node through assigned-clocks, assigned-clock-parents and assigned-clock-rates
# properties.

description: Clock consumer

include: base.yaml

properties:
    clocks:
      required: true
      description: >
        List of phandle and clock specifier pairs, one pair for each clock
        input to the device. Note - if the clock provider specifies '0' for
        clock-cells, then only the phandle portion of the pair will appear.

    clock-names:
      required: false
      description: >
        List of clock input name strings sorted in the same order as the clocks
        property.

    clock-ranges:
      type: boolean
      required: false
      description: >
        Empty property indicating that child nodes can inherit named clocks from
        this node. Useful for bus nodes to provide a clock to their children.

    assigned-clocks:
      type: phandle-array
      required: false
      description: >
        List of phandle and clock specifier pairs, one pair for each assigned
        clock input. Note - if the clock provider specifies '0' for
        clock-cells, then only the phandle portion of the pair will appear.

    assigned-clock-parents:
      type: phandle-array
      required: false
      description: >
        List of parent clocks in the form of a phandle and clock
        specifier pair. The list shall correspond to the clocks listed in the
        assigned-clocks directive.

    assigned-clock-rates:
      type: array
      required: false
```

(continues on next page)

(continued from previous page)

```
      description: >
        List of frequencies in Hz. The list shall correspond to the clocks
        listed in the assigned-clocks directive.
```

clock-controller

```
include: clock-controller.yaml
```

```
# Copyright (c) 2019, Linaro Limited
# Copyright (c) 2019..2020 Bobby Noelte
# SPDX-License-Identifier: Apache-2.0

# Common fields for clock controllers

properties:
    "#clock-cells":
      type: int
      required: true
      description: Number of items to expect in a Clock specifier

    clock-output-names:
        type: string-array
        required: false
        description: >
          A list of strings of clock output signal names indexed by the first
          cell in the clock specifier.

    clock-indices:
        type: array
        required: false
        description: >
          The identifying number for the clocks in the node. If it is not linear
          from zero, then this allows the mapping of identifiers into the
          clock-output-names array.

    protected-clocks:
        type: phandles
        required: false
        description: >
          Clocks that are not fully exposed, such as in situations where those
          clocks are used by drivers running in ARM secure execution levels.
```

cpu

```
include: cpu.yaml
```

```
# Copyright (c) 2019 Nordic Semiconductor ASA
# SPDX-License-Identifier: Apache-2.0

# Common fields for CPUs

include: base.yaml
```

(continues on next page)

(continued from previous page)

```
properties:
    clock-frequency:
      type: int
      required: false
      description: Clock frequency in Hz
```

dma-client

```
include: dma-client.yaml
```

```
# Copyright (c) 2018..2020 Bobby Noelte
# SPDX-License-Identifier: Apache-2.0

description: DMA client

properties:
    dmas:
      type: phandle-array
      required: true
      description: >
        List of one or more DMA specifiers, each consisting of
        - A phandle pointing to DMA controller node
        - A number of integer cells, as determined by the
        dma-cells property in the node referenced by phandle
        containing DMA controller specific information. This
        typically contains a DMA request line number or a
        channel number, but can contain any data that is
        required for configuring a channel.

    dma-names:
      type: string-array
      required: false
      description: >
        Contains one identifier string for each DMA specifier in
        the dmas property. The specific strings that can be used
        are defined in the binding of the DMA client device.
        Multiple DMA specifiers can be used to represent
        alternatives and in this case the dma-names for those
        DMA specifiers must be identical (see examples).
```

dma-controller

```
include: dma-controller.yaml
```

```
# Copyright (c) 2019, Song Qiang <songqiang1304521@gmail.com>
# SPDX-License-Identifier: Apache-2.0

# Common fields for DMA controllers

include: base.yaml
```

(continues on next page)

(continued from previous page)

```
bus: dma

properties:
    label:
      required: true

    "#dma-cells":
      type: int
      required: true
      description: Number of items to expect in a DMA specifier

    dma-channel-mask:
      type: int
      required: false
      description: >
        Bitmask of available DMA channels in ascending order that are
        not reserved by firmware and are available to the
        kernel. i.e. first channel corresponds to LSB.

    dma-channels:
      type: int
      required: false
      description: Number of DMA channels supported by the controller

    dma-requests:
      type: int
      required: false
      description: Number of DMA request signals supported by the controller.
```

eeprom-base

```
include: eeprom-base.yaml
```

```
# Copyright (c) 2019 Vestas Wind Systems A/S
# SPDX-License-Identifier: Apache-2.0

# Common fields for EEPROM devices

include: base.yaml

properties:
    size:
      type: int
      required: false
      description: Total EEPROM size in bytes
    read-only:
      type: boolean
      required: false
      description: Disable writes to the EEPROM
    label:
      required: true
```

eeprom-spi-i2c

```
include: eeprom-spi-i2c.yaml
```

```
# Copyright (c) 2019 Vestas Wind Systems A/S
# SPDX-License-Identifier: Apache-2.0

# Common fields for I2C and SPI EEPROM devices

include: eeprom-base.yaml

properties:
    size:
      required: true
    pagesize:
      type: int
      required: true
      description: EEPROM page size in bytes
    address-width:
      type: int
      required: true
      description: EEPROM address width in bits
    timeout:
      type: int
      required: true
      description: EEPROM write cycle timeout in milliseconds
    wp-gpios:
      type: phandle-array
      required: false
      description: GPIO to which the write-protect pin of the chip is connected
```

espi-controller

```
include: espi-controller.yaml
```

```
# Copyright (c) 2019 Intel Corporation
# SPDX-License-Identifier: Apache-2.0

# Common fields for ESPI devices

include: base.yaml

bus: espi

properties:
    label:
      required: true
```

ethernet

```
include: ethernet.yaml
```

```
# Copyright (c) 2018, Linaro Limited
# SPDX-License-Identifier: Apache-2.0

# Common fields for Ethernet devices

include: base.yaml

properties:
    local-mac-address:
      type: uint8-array
      required: false
      description: mac address
    label:
      required: true
```

fixed-clock

```
include: fixed-clock.yaml
```

```
#
# Copyright (c) 2017..2019 b0661n0e17e@gmail.com
#
# SPDX-License-Identifier: Apache-2.0
#

title: Simple fixed-rate clock sources.

description: >
    Binding for simple fixed-rate clock sources.

inherits:
    !include clock-provider.yaml

properties:
    compatible:
      constraint: "fixed-clock"

    "#clock-cells":
      constraint: 0

    clock-frequency:
      type: int
      category: required
      description: Frequency of clock in Hz. Should be a single cell.

    clock-accuracy:
      type: int
      category: optional
      description: Accuracy of clock in ppb (parts per billion). Should be a single␣
↪cell.
```

(continues on next page)

(continued from previous page)

```
    oscillator:
      type: int
      category: optional
      description: clock is an oszillator (a quartz)
```

fixed-partition

```
include: fixed-partition.yaml
```

```
# Copyright (c) 2018..2020 Bobby Noelte
# SPDX-License-Identifier: Apache-2.0

description: Fixed partition (one of the partions of a 'fixed-partitions' node).

compatible: "fixed-partition"

properties:
    label:
        type: string
        required: false
        description: The label / name for this partition.  If omitted, the label is␣
↪taken
                     from the node name (excluding the unit address).

    read-only:
        type: boolean
        required: false
        description: This parameter, if present, is a hint that this
                     partition should/ can only be used read-only.

    reg:
        type: array
        required: false
        description: partition offset (address) and size within flash
```

flash

```
include: flash.yaml
```

```
# Copyright (c) 2018..2020 Bobby Noelte
# SPDX-License-Identifier: Apache-2.0

description: Flash Base Structure

include: base.yaml

properties:
    label:
      required: true

    reg:
      required: true
```

(continues on next page)

(continued from previous page)

```
    "#address-cells":
      type: int
      required: true
      description: >
        <1>: for flash devices that require a single 32-bit cell to represent their
            address (aka the value is below 4 GiB)
        <2>: for flash devices that require two 32-bit cells to represent their
            address (aka the value is 4 GiB or greater).

    "#size-cells":
      type: int
      required: true
      description: >
        <1>: for flash devices that require a single 32-bit cell to represent their
            size (aka the value is below 4 GiB)
        <2>: for flash devices that require two 32-bit cells to represent their
            size (aka the value is 4 GiB or greater).

    write-block-size:
      type: int
      required: false
      description: Size of flash blocks  for write operations

    erase-block-size:
      type: int
      required: false
      description: Size of flash blocks for erase operations
```

flash-controller

```
include: flash-controller.yaml
```

```
# Common fields for flash controllers

include: base.yaml

properties:
    label:
      required: true

    reg:
      required: true
```

gpio-controller

```
include: gpio-controller.yaml
```

```
# Copyright (c) 2019, Linaro Limited
# Copyright (c) 2019..2020 Bobby Noelte
# SPDX-License-Identifier: Apache-2.0

# Common fields for GPIO controllers

include: base.yaml

properties:
    "gpio-controller":
      type: boolean
      required: true
      description: Convey's this node is a GPIO controller

    "#gpio-cells":
      type: int
      required: true
      description: Number of items to expect in a GPIO specifier

    ngpios:
      type: int
      required: false
      description: Number of gpios supported

    gpio-ranges:
      type: phandle-array
      required: false
      description: gpio range in pin controller

    gpio-ranges-group-names:
      type: string-array
      required: false
      description: gpio range names

gpio-cells:
  - pin
  - flags
```

gpio-keys

```
include: gpio-keys.yaml
```

```
# Copyright (c) 2018, Linaro Limited
# SPDX-License-Identifier: Apache-2.0

description: GPIO KEYS parent node

compatible: "gpio-keys"

child-binding:
    description: GPIO KEYS child node
```

(continues on next page)

(continued from previous page)

```
    properties:
       gpios:
          type: phandle-array
          required: true
       label:
          required: true
          type: string
          description: Human readable string describing the device (used by Zephyr␣
↪for API name)
```

gpio-led

```
include: gpio-led.yaml
```

```
# Copyright (c) 2020 Bobby Noelte
# SPDX-License-Identifier: Apache-2.0

description: GPIO LED

compatible: "gpio-led"

properties:
    label:
        type: string
        required: false
        description: Human readable string describing the device (used by Zephyr for␣
↪API name)
    gpios:
      type: phandle-array
      required: true
    function:
      required: false
      type: int
      description: Numerical LED functon identifier.
    color:
      required: false
      type: int
      description: Numerical LED color identifier
    default-state:
      required: false
      type: int
      default: 0
      description: >
        The initial state of the LED. Valid values are LED_STATE_OFF (0),
        LED_STATE_ON (1), and LED_STATE_KEEP (2). If the LED is already on or
        off and the default-state property is set the to same value, then no
        glitch should be produced where the LED momentarily turns off (or on).
        The LED_STATE_KEEP setting will keep the LED at whatever its current
        state is, without producing a glitch. The default is LED_STATE_OFF (0)
        if this property is not present.
    led-pattern:
      required: false
      type: array
      description: Array of integers with default pattern for certain triggers.
```

(continues on next page)

(continued from previous page)

```
    retain-state-suspended:
      required: false
      type: boolean
      description: Retain the state of the LED in suspend state.
    retain-state-shutdown:
      required: false
      type: boolean
      description: Retain the state of the LED on shutdown.
    panic-indicator:
      required: false
      type: boolean
      description: The LED should be used as a panic indicator.
```

gpio-leds

```
include: gpio-leds.yaml
```

```
# Copyright (c) 2018, Linaro Limited
# Copyright (c) 2020 Bobby Noelte
# SPDX-License-Identifier: Apache-2.0

description: GPIO LEDs controller

compatible: "gpio-leds"

include: base.yaml

properties:
       label:
          default: "GPIO_LEDS"
```

gpio-nexus

```
include: gpio-nexus.yaml
```

```
# Copyright (c) 2019, Linaro Limited
# SPDX-License-Identifier: Apache-2.0

# Common fields for GPIO nexus nodes

properties:
    gpio-map:
      type: compound
      required: true

    gpio-map-mask:
      type: compound
      required: false

    gpio-map-pass-thru:
      type: compound
      required: false
```

(continues on next page)

(continued from previous page)

```
    "#gpio-cells":
      type: int
      required: true
      description: Number of items to expect in a GPIO specifier
```

i2c-controller

```
include: i2c-controller.yaml
```

```
# Copyright (c) 2017 I-SENSE group of ICCS
# SPDX-License-Identifier: Apache-2.0

# Common fields for I2C controllers

include: base.yaml

bus: i2c

properties:
    "#address-cells":
      required: true
      const: 1
    "#size-cells":
      required: true
      const: 0
    clock-frequency :
      type: int
      required: false
      description: Initial clock frequency in Hz
    label:
      required: true
```

i2c-device

```
include: i2c-device.yaml
```

```
# Copyright (c) 2017, Linaro Limited
# SPDX-License-Identifier: Apache-2.0

# Common fields for I2C devices

include: base.yaml

on-bus: i2c

properties:
    reg:
      required: true
    label:
      required: true
```

i2s-controller

```
include: i2s-controller.yaml
```

```
# Copyright (c) 2018, STMicroelectronics
# SPDX-License-Identifier: Apache-2.0

# Common fields for I2S controllers

include: base.yaml

bus: i2s

properties:
    "#address-cells":
      required: true
      const: 1
    "#size-cells":
      required: true
      const: 0
    label:
      required: true
```

i2s-device

```
include: i2s-device.yaml
```

```
# Copyright (c) 2018, STMicroelectronics
# SPDX-License-Identifier: Apache-2.0

# Common fields for I2S devices

include: base.yaml

on-bus: i2s

properties:
    reg:
      required: true
    label:
      required: true
```

interrupt-controller

```
include: interrupt-controller.yaml
```

```
# Copyright (c) 2019, Linaro Limited
# SPDX-License-Identifier: Apache-2.0

# Common fields for interrupt controllers

properties:
```

(continues on next page)

(continued from previous page)

```
    "interrupt-controller":
      type: boolean
      required: true
      description: Convey's this node is an interrupt controller
    "#interrupt-cells":
      type: int
      required: true
      description: Number of items to expect in an interrupt specifier
```

kscan

```
include: kscan.yaml
```

```
# Copyright (c) 2019, Intel Corporation
# SPDX-License-Identifier: Apache-2.0

# Common properties for keyboard matrix devices

include: base.yaml

bus: kscan

properties:
    "#address-cells":
      required: true
      const: 1
    "#size-cells":
      type: int
      const: 0
    label:
      required: true
```

memory

```
include: memory.yaml
```

```
# Copyright (c) 2020 Bobby Noeltw
# SPDX-License-Identifier: Apache-2.0

# Common fields for memory

description: Memory properties

compatible: "memory"

include: base.yaml

properties:
    reg:
      required: true
```

memory-region

```
include: memory-region.yaml
```

```
# Copyright (c) 2020 Bobby Noeltw
# SPDX-License-Identifier: Apache-2.0

# Common fields for memory regions

description: Memory region properties

compatible: "memory-region"

include: base.yaml

properties:
    reg:
      required: true

    label:
      required: false
```

mmc

```
include: mmc.yaml
```

```
# Copyright (c) 2019, NXP
# SPDX-License-Identifier: Apache-2.0

# Specifies the MMC/SDHC module

include: base.yaml
```

mmc-spi-slot

```
include: mmc-spi-slot.yaml
```

```
# Copyright (c) 2018 Google LLC.
# SPDX-License-Identifier: Apache-2.0

description: MMC/SD/SDIO slot connected via SPI

compatible: "zephyr,mmc-spi-slot"

include: spi-device.yaml
```

mmio-sram

```
include: mmio-sram.yaml
```

```
# Copyright (c) 2018, Linaro Limited
# SPDX-License-Identifier: Apache-2.0

description: Generic on-chip SRAM description

compatible: "mmio-sram"

include: base.yaml

properties:
    reg:
      required: true

    label:
      required: false
```

partition

```
include: partition.yaml
```

```
# Copyright (c) 2018,2019 Bobby Noelte
# SPDX-License-Identifier: Apache-2.0

description: Flash partition definition for fixed partitions

compatible: "fixed-partitions"

properties:
    "#address-cells":
      type: int
      required: true
      description: >
        <1>: for partitions that require a single 32-bit cell to represent their
            size/address (aka the value is below 4 GiB)
        <2>: for partitions that require two 32-bit cells to represent their
            size/address (aka the value is 4 GiB or greater).

    "#size-cells":
      type: int
      required: false
      description: >
        <1>: for partitions that require a single 32-bit cell to represent their
            size/address (aka the value is below 4 GiB)
        <2>: for partitions that require two 32-bit cells to represent their
            size/address (aka the value is 4 GiB or greater).
```

phy-controller

```
include: phy-controller.yaml
```

```
# Copyright (c) 2018, Yannis Damigos
# SPDX-License-Identifier: Apache-2.0

# Common fields for PHY providers

include: base.yaml

properties:
    "#phy-cells":
        type: int
        required: true
        description: Number of cells in a PHY provider. The meaning those
                     cells is defined by the binding for the phy node.
```

pin-controller

```
include: pin-controller.yaml
```

```
# Copyright (c) 2018..2020 Bobby Noelte
# SPDX-License-Identifier: Apache-2.0

description: Pin controller

include: base.yaml

properties:
    pin-controller:
      type: boolean
      required: true
      description: device controller identification

    "#pinctrl-cells":
      type: int
      required: false
      description: >
        Number of pin control cells in addition to the index within the
        pin controller device instance
```

pincfg

```
include: pincfg.yaml
```

```
# Copyright (c) 2018..2020 Bobby Noelte
# SPDX-License-Identifier: Apache-2.0

description: Pin configuration

compatible: "pincfg"
```

(continues on next page)

(continued from previous page)

```
properties:

    groups:
        description: >
            The list of names of the groups that properties in the node apply to
            (either this, "pins", "pinmux" or "pinctrl-pin-array" have to be
            specified).
        required: false
        type: string-array

    pins:
        description: >
            The list of numeric pin ids that properties in the node apply to
            (either this, "groups", "pinmux" or "pinctrl-pin-array" have to be
            specified).
        required: false
        type: array

    function:
        required: false
        type: string

    pinmux:
        description: >
            The list of numeric pin ids and their mux settings that properties
            in the node apply to (either this, "pins", "groups" or
            "pinctrl-pin-array" have to be specified).
        required: false
        type: array

    pinctrl-pin-array:
        description: >
            The list of pin controller register index and values (either this,
            "pins", "pinmux" or "groups" have to be specified). pinctrl-cells
            specifies the number of value cells in addition to the index of the
            registers. No other properties shall be in the node.
        required: false
        type: array

    bias-disable:
        description: Disable any pin bias.
        required: false
        type: boolean

    bias-high-impedance:
        required: false
        type: boolean

    bias-bus-hold:
        required: false
        type: boolean

    bias-pull-up:
        description: Pull up strength in Ohm. 0 to disable pull up.
        required: false
        type: int
```

(continues on next page)

(continued from previous page)

```
    bias-pull-down:
        description: Pull down strength in Ohm. 0 to disable pull down.
        required: false
        type: int

    bias-pull-pin-default:
        required: false
        type: int

    drive-push-pull:
        required: false
        type: boolean

    drive-open-drain:
        required: false
        type: boolean

    drive-open-source:
        required: false
        type: boolean

    drive-strength:
        description: >
            Sink or source at most X mA. 0 to disable drive strength control.
        required: false
        type: int

    input-enable:
        required: false
        type: boolean

    input-debounce:
        description: Debounce time in usec. 0 to disable debouncing.
        required: false
        type: int

    input-disable:
        required: false
        type: boolean

    input-schmitt-enable:
        required: false
        type: boolean

    input-schmitt-disable:
        required: false
        type: boolean

    low-power-enable:
        description: Enable low power mode.
        required: false
        type: boolean

    low-power-disable:
        description: Disable low power mode.
        required: false
```

(continues on next page)

(continued from previous page)

```
        type: boolean

    output-disable:
        required: false
        type: boolean

    output-enable:
        required: false
        type: boolean

    output-low:
        required: false
        type: boolean

    output-high:
        required: false
        type: boolean

    power-source:
        description: Numerical ID of power source to select.
        required: false
        type: int
        default: 0

    slew-rate:
        description: Slew rate
        required: false
        type: int
        default: 0

    skew-delay:
        description: >
            Expected clock skew on input pins and the delay before latching
            a value to an output pin.
            Typically indicated by the number of double-inverters used to
            delay the signal.
        required: false
        type: int
        default: 0
```

pinctrl-state

```
include: pinctrl-state.yaml
```

```
# Copyright (c) 2018..2020 Bobby Noelte
# SPDX-License-Identifier: Apache-2.0

description: >
    Pinctrl state. A node that bundles a set of pin configuration
    nodes that shall be applied in a single pin control operation.

compatible: "pinctrl-state"
```

ps2

```
include: ps2.yaml
```

```
# Copyright (c) 2019, Intel Corporation
# SPDX-License-Identifier: Apache-2.0

# Common properties for PS/2 devices

include: base.yaml

bus: ps2

properties:
    "#address-cells":
      type: int
      required: true
      description: should be 1.
    "#size-cells":
      type: int
      required: true
      description: should be 0.
    label:
      required: true
```

pwm-client

```
include: pwm-client.yaml
```

```
#
# Copyright (c) 2018..2020 Bobby Noelte
#
# SPDX-License-Identifier: Apache-2.0
#

description: PWM Client

properties:
    pwms:
      type: phandle-array
      required: true
      description: >
        List of phandle and pwm specifiers, one set for each pwm
        input to the device. Note - The number of pwm specifiers is
        determined by the dma-cells property in the node referenced by phandle

    pwm-names:
      type: string-array
      required: false
      description: >
        List of strings to label each of the PWM devices sorted in the same
        order as the pwms property.
```

pwm-controller

```
include: pwm-controller.yaml
```

```
# Copyright (c) 2017, NXP
# Copyright (c) 2019 Bobby Noelte
#
# SPDX-License-Identifier: Apache-2.0

# Common fields for PWM controllers

include: base.yaml

properties:
    label:
      required: true

    "#pwm-cells":
      type: int
      required: true
      description: Number of items to expect in a pwm specifier

    clocks:
      required: false
```

pwm-leds

```
include: pwm-leds.yaml
```

```
# Copyright (c) 2018, Linaro Limited
# SPDX-License-Identifier: Apache-2.0

description: PWM LEDs parent node

compatible: "pwm-leds"

child-binding:
    description: PWM LED child node
    properties:
        pwms:
          type: phandle-array
          required: true

        label:
          required: false
          type: string
          description: Human readable string describing the device (used by Zephyr␣
↪for API name)
```

reserved-memory

```
include: reserved-memory.yaml
```

```
# Copyright (c) 2020 Bobby Noeltw
# SPDX-License-Identifier: Apache-2.0

# Common fields for reserved memory

description: Reserved memory properties

compatible: "reserved-memory"

include: base.yaml

properties:
    ranges:
      type: boolean
      required: true
      description: standard definition
```

reserved-memory-region

```
include: reserved-memory-region.yaml
```

```
# Copyright (c) 2020 Bobby Noeltw
# SPDX-License-Identifier: Apache-2.0

# Common fields for reserved memory regions

description: Reserved memory region properties

compatible: "reserved-memory-region"

include: base.yaml

properties:
    reg:
      required: false

    size:
      type: int
      required: false
      description: >
        length based on parent's #size-cells
        Size in bytes of memory to reserve.

    alignment:
      type: int
      required: false
      description: >
        length based on parent's #size-cells
        Address boundary for alignment of allocation.

    alloc-ranges:
```

(continues on next page)

(continued from previous page)

```
      type: array
      required: false
      description: >
        prop-encoded-array (address, length pairs).
        Specifies regions of memory that are
        acceptable to allocate from.
```

rtc

```
include: rtc.yaml
```

```
# Copyright (c) 2018, blik GmbH
# SPDX-License-Identifier: Apache-2.0

# Common fields for RTC devices

include: base.yaml

properties:
    clock-frequency:
      type: int
      required: false
      description: Clock frequency information for RTC operation
    label:
      required: true
    interrupts:
      required: true

    prescaler:
      type: int
      required: false
      description: RTC frequency equals clock-frequency divided by the prescaler value
```

shared-irq

```
include: shared-irq.yaml
```

```
description: Shared IRQ interrupt dispatcher

compatible: "shared-irq"

include: [interrupt-controller.yaml, base.yaml]

properties:
  interrupts:
      required: true

  label:
      required: true
```

soc-nv-flash

```
include: soc-nv-flash.yaml
```

```
description: Flash node

compatible: "soc-nv-flash"

include: base.yaml

properties:
    label:
      required: false

    erase-block-size:
     type: int
     description: address alignment required by flash erase operations
     required: false

    write-block-size:
     type: int
     description: address alignment required by flash write operations
     required: false
```

spi-controller

```
include: spi-controller.yaml
```

```
# Copyright (c) 2018, I-SENSE group of ICCS
# SPDX-License-Identifier: Apache-2.0

# Common fields for SPI controllers

include: base.yaml

bus: spi

properties:
    clock-frequency:
      type: int
      required: false
      description: Clock frequency the SPI peripheral is being driven at
    "#address-cells":
      required: true
      const: 1
    "#size-cells":
      required: true
      const: 0
    label:
      required: true
    cs-gpios:
      type: phandle-array
      required: false
```

spi-device

```
include: spi-device.yaml
```

```
# Copyright (c) 2018, I-SENSE group of ICCS
# SPDX-License-Identifier: Apache-2.0

# Common fields for SPI devices

include: base.yaml

on-bus: spi

properties:
    reg:
      required: true
    spi-max-frequency:
      type: int
      required: true
      description: Maximum clock frequency of device's SPI interface in Hz
    label:
      required: true
```

uart-controller

```
include: uart-controller.yaml
```

```
# Common fields for UART controllers

include: base.yaml

bus: uart

properties:
    clock-frequency:
      type: int
      required: false
      description: Clock frequency information for UART operation
    current-speed:
      type: int
      required: false
      description: Initial baud rate setting for UART
    label:
      required: true
    hw-flow-control:
      type: boolean
      required: false
      description: Set to enable RTS/CTS flow control at boot time
```

uart-device

```
include: uart-device.yaml
```

```
# Copyright (c) 2018, Foundries.io
# SPDX-License-Identifier: Apache-2.0

# Common fields for UART devices

include: base.yaml

on-bus: uart

properties:
    label:
      required: true
```

usb-controller

```
include: usb-controller.yaml
```

```
# Copyright (c) 2018, I-SENSE group of ICCS
# SPDX-License-Identifier: Apache-2.0

# Common fields for USB controllers

include: base.yaml

properties:
    maximum-speed:
      type: string
      required: false
      description: Configures USB controllers to work up to a specific
                   speed. Valid arguments are "super-speed", "high-speed",
                   "full-speed" and "low-speed". If this is not passed
                   via DT, USB controllers should use their maximum
                   hardware capability.
      enum:
         - "low-speed"
         - "full-speed"
         - "high-speed"
         - "super-speed"

    label:
      required: true
```

usb-ep

```
include: usb-ep.yaml
```

```
# Copyright (c) 2018, I-SENSE group of ICCS
# SPDX-License-Identifier: Apache-2.0

# Common fields that give the number of endpoints supported by the USB hardware

include: usb-controller.yaml

properties:
    num-bidir-endpoints:
      type: int
      required: true
      description: Number of bi-directional endpoints supported by hardware
                   (including EP0)

    num-in-endpoints:
      type: int
      required: false
      description: Number of IN endpoints supported by hardware
                   (including EP0 IN)

    num-out-endpoints:
      type: int
      required: false
      description: Number of OUT endpoints supported by hardware
                   (including EP0 OUT)
```

EDTS Bindings Index

- *fixed-partition*
- *flash-controller*
- *flash*
- *gpio-controller*
- *gpio-keys*
- *gpio-led*
- *gpio-leds*
- *gpio-nexus*
- *i2c-controller*
- *i2c-device*
- *i2s-controller*
- *i2s-device*
- *interrupt-controller*
- *kscan*
- *memory-region*
- *memory*
- *mmc-spi-slot*
- *mmc*
- *mmio-sram*
- *partition*
- *phy-controller*
- *pin-controller*
- *pincfg*
- *pinctrl-state*
- *ps2*
- *pwm-client*
- *pwm-controller*
- *pwm-leds*
- *reserved-memory-region*
- *reserved-memory*
- *rtc*
- *shared-irq*
- *soc-nv-flash*
- *spi-controller*
- *spi-device*
- *uart-controller*

- *uart-device*
- *usb-controller*
- *usb-ep*

CHAPTER

TWO

ABOUT COGENO

For some repetitive or parameterized coding tasks, it's convenient to use a code generating tool to build code fragments, instead of writing (or editing) that source code by hand.

Cogeno, the inline code generation tool, processes Python 3 or Jinja2 script "snippets" inlined in your source files. It can also access CMake build parameters and device tree information to generate source code automatically tailored and tuned to a specific project configuration.

Cogeno can be used, for example, to generate source code that creates and fills data structures, adapts programming logic, creates configuration-specific code fragments, and more.

2.1 About Cogeno scripts

Script snippets that are inlined in a source file are used as code generators. The tool to scan the source file for the script snippets and process them is Cogeno. Cogeno and part of this documentation is based on Cog from Ned Batchelder.

The inlined script snippets can contain any Python 3 or Jinja2 code, they are regular scripts.

All Python snippets in a source file and all Python snippets of included template files are treated as a python script with a common set of global Python variables. Global data created in one snippet can be used in another snippet that is processed later on. This feature could be used, for example, to customize included template files.

Jinja2 snippets provide a - compared to Python - simplified script language.

An inlined script snippet can always access the cogeno module. The cogeno module encapsulates and provides all the functions to retrieve information (options, devicetree properties, CMake variables, config properties) and to output the generated code.

Cogeno transforms files in a very simple way: it finds snippets of script code embedded in them, executes the script code, and places its output combined with the original file into the generated file. The original file can contain whatever text you like around the script code. It will usually be source code.

For example, if you run this file through cogeno:

```
/* This is my C file. */
...
/**
 * @code{.cogeno.py}
 * fnames = ['DoSomething', 'DoAnotherThing', 'DoLastThing']
 * for fn in fnames:
 *     cogeno.outl(f'void {fn}();')
 * @endcode{.cogeno.py}
 */
/** @code{.cogeno.ins}@endcode */
...
```

it will come out like this:

```
/* This is my C file. */
...
/**
 * @code{.cogeno.py}
 * fnames = ['DoSomething', 'DoAnotherThing', 'DoLastThing']
 * for fn in fnames:
 *     cogeno.outl(f'void {fn}();')
 * @endcode{.cogeno.py}
 */
void DoSomething();
void DoAnotherThing();
void DoLastThing();
/** @code{.cogeno.ins}@endcode */
...
```

Lines with `@code{.cogeno.py}` or `@code{.cogeno.ins}@endcode` are marker lines. The lines between `@code{.cogeno.py}` and `@endcode{.cogeno.py}` are the generator Python code. The lines between `@endcode{.cogeno.py}` and `@code{.cogeno.ins}@endcode` are the output from the generator.

When Cogeno runs, it discards the last generated Python output, executes the generator Python code, and writes its generated output into the file. All text lines outside of the special markers are passed through unchanged.

The Cogeno marker lines can contain any text in addition to the marker tokens. This makes it possible to hide the generator Python code from the source file.

In the sample above, the entire chunk of Python code is a C comment, so the Python code can be left in place while the file is treated as C code.

2.2 About Cogeno extensions

On invocation Cogeno scans a list of given directories for extension modules and imports the modules. An extension module may add additional options to the Cogeno invocation or may provide other information specific to the extension directory. The *Cogeno extensions* are available to the script snippets.

2.3 About Cogeno modules

Cogeno includes several *Cogeno modules* to support specific code generation tasks.

- *C code generation (ccode)*
- *CMake support (cmake)*
- *Config/ Kconfig support (config)*
- *Extended device tree support (edts)*
- *Protocol buffer code generation (protobuf)*
- *ReST/ RST code generation (rstcode)*
- *Zephyr support (zephyr)*

CHAPTER

THREE

GETTING STARTED WITH COGENO

3.1 Set up an environment for Cogeno

To start off, be sure that you have installed a fairly recent Python 3 and the Python package manager pip3. Git is needed in case you want to get the source.

3.2 Get Cogeno

3.2.1 Get the source

Cogeno's latest version is available from https://gitlab.com/b0661/cogeno:

```
git clone https://gitlab.com/b0661/cogeno.git
```

3.2.2 Install Cogeno

Cogeno can be installed from PyPi:

```
pip3 install cogeno
```

Cogeno's latest version can be installed by:

```
git clone https://gitlab.com/b0661/cogeno.git
pip3 install -e cogeno
```

3.3 Run Cogeno

To run from source one time assure all dependencies are installed:

```
pip3 install -r <cogeno repo clone>/requirements.txt
```

Then call:

```
python3 <cogeno repo clone>/cogeno/cogeno.py <any option>
```

To run the installed variant you only have to call:

```
cogeno <any option>
```

For all the possible options see *Invoking Cogeno*.

3.4 Integrate Cogeno into your project´s build system

Cogeno usually is invoked during the build process by your favourite build system.

Cogeno provides example solutions for some of them (see *Build with Cogeno*), especially for the CMake build system generator.

3.5 Get Cogeno´s documentation

Cogeno´s documentation is available online at Read the Docs.

The source of the documentation is in the `docs` folder.

To build it first assure all dependencies are fulfilled:

```
pip3 install -r <cogeno repo clone>/cogeno/docs/requirements.txt
```

You can then create it by:

```
cd <cogeno repo clone>/cogeno/docs
make html
```

CHAPTER

FOUR

INVOKING COGENO

4.1 Synopsis

cogeno [OPTIONS]

4.2 Description

Cogeno transforms files in a very simple way: it finds chunks of script code embedded in them, executes the script code, and places its output combined with the original file content into the generated file. It supports Python and Jinja2 scripts.

4.3 Options

The following options are understood:

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

-x, --delete-code Delete the generator code from the output file.

-w, --warn-empty Warn if a file has no generator code in it.

-n ENCODING, --encoding ENCODING Use ENCODING when reading and writing files.

-U, --unix-newlines Write the output with Unix newlines (only LF line-endings).

-D DEFINE, --define DEFINE Define a global string available to your generator code.

-m DIR [DIR ...], --modules DIR [DIR ...] Use modules from modules DIR. We allow multiple

-t DIR [DIR ...], --templates DIR [DIR ...] Use templates from templates DIR. We allow multiple

-e DIR [DIR ...], --extensions DIR [DIR ...] Import extensions from extensions DIR. We allow multiple

-i FILE, --input FILE Get the input from FILE.

-o FILE, --output FILE Write the output to FILE. "-" indicates stdout.

--output-sanitize-suffix Sanitize the suffix of the output file. Remove typical template suffixes.

-l FILE, --log FILE Log to FILE.

-k FILE, --lock FILE Use lock FILE for concurrent runs of cogeno.

--base Return the base directory of cogeno. Other options are ignored.

Additional options are related to *Cogeno modules*.

--cmake:cache FILE Use CMake variables from CMake cache FILE.

--cmake:define [defxxx=valyyy ...] Define CMake variables to your generator code.

--config:db FILE Write or read config database to/ from FILE.

--config:file FILE Read configuration variables from this FILE.

--config:kconfig-file FILE Top-level Kconfig FILE (default: Kconfig).

--config:kconfig-srctree DIR Kconfig files are looked up relative to the srctree DIR (unless absolute paths are used), and .config files are looked up relative to the srctree DIR if they are not found in the current directory.

--config:kconfig-defines DEFINE [DEFINE ...] Define variable to Kconfig. We allow multiple.

--config:inputs FILE [FILE ...] Read configuration file fragment from FILE. We allow multiple.

--edts:bindings-dirs DIR [DIR ...] Use bindings from bindings DIR for device tree extraction. We allow multiple.

--edts:bindings-exclude [DIR ...] [FILE ...] Exclude bindings DIR or FILE from usage for device tree extraction. We allow multiple.

--edts:bindings-no-default Do not add EDTS database's generic bindings to bindings by default.

--edts:db FILE Write or read EDTS database to/ from FILE.

--edts:dts FILE Write (see dts-pp-sources) or read device tree specification to/ from this FILE.

--edts:dts-pp-sources FILE [FILE ...] Generate the DTS file by pre-processing the DTS source FILE(s).

--edts:dts-pp-include-dirs DIR [DIR ...] Define include DIR(s) to pre-processor.

--edts:dts-pp-defines DEFINE [DEFINE ...] DEFINE variable to pre-processor.

--protobuf:db-dir DIR Write or read protocol buffer code databases to/ from DIR.

--protobuf:sources FILE [FILE ...] The *.proto source FILE(s) to generate the protocol buffer code for.

--protobuf:include-dirs DIR [DIR ...] Search for *.proto import files in the protocol buffer include DIR(s).

CHAPTER

FIVE

SCRIPTING WITH COGENO

5.1 Code generation functions

The `cogeno` module provides the core functions for inline code generation. It encapsulates all the functions to retrieve information (options, device tree properties, CMake variables, config properties) and to output the generated code.

The `cogeno` module is automatically imported by all code snippets. No explicit import is necessary.

Note: The `cogeno` module provides the public functions of the code generator `Mixin` classes as cogeno functions. You can simply write:

cogeno.func(...)

The mixin class function cogeno.xxx.XxxMixin.func(self, ...) is not directly available to code snippets.

5.1.1 Output

cogeno.out(*args)

cogeno::output::OutputMixin.out(self, args)

Write text to the output.

The string arguments are concenated. The filters are then applied to the lines of the concenated string. The resulting string is written to the output.

`OutputFilterDedent` and `OutputFilterTrimBlankLines` make it easier to use multi-line strings, and they are only are useful for multi-line strings:

```
cogeno.out("""
   These are lines I
   want to write into my source file.
""", cogeno.OutputFilterDedent(), cogeno.OutputFilterTrimBlankLines())
```

Return output string

Parameters

- `*args`: Variable length argument list of strings and output filters.

cogeno.outl(*args)

cogeno::output::OutputMixin.outl(self, args)
Write text to the output with newline appended.

See OutputMixin::out(self, *args)

Return output string

Parameters

- `*args`: Variable length argument list of strings and output filters.

cogeno.out_insert(insert_file, *args)

cogeno::output::OutputMixin.out_insert(self, insert_file, args)
Insert the text from the file into the output.

See OutputMixin::out(self, *args)

Return output string

Parameters

- `insert_file`: Path of file, either absolute path or relative to current directory or relative to templates directory.
- `*args`: Variable length argument list of strings and output filters.

Output filters

cogeno.OutputFilterTrimBlankLines()

cogeno.output.OutputMixin.OutputFilterTrimBlankLines : public cogeno.output.OutputMixin.Out
Remove initial and trailing blank lines from the block of lines.

cogeno.OutputFilterTrimDedent()

cogeno.output.OutputMixin.OutputFilterDedent : public cogeno.output.OutputMixin.OutputFilte
Remove common initial white space from the lines.

Parameters

- `new_indent`: Optional new indentation (after dedent)

cogeno.OutputFilterLineNumbers()

cogeno.output.OutputMixin.OutputFilterLineNumbers : public cogeno.output.OutputMixin.Output
Filter lines by line numbers.

Filter lines that are given by line sumber specifications (such as "1,2,4-6").

Parameters

- `args`: list of arguments denoting line number specifications

cogeno.OutputFilterStartAt()

cogeno.output.OutputMixin.OutputFilterStartAt : public cogeno.output.OutputMixin.OutputFilt
Start output at pattern.

Parameters

- `start_at`: Start pattern

cogeno.OutputFilterStopAt()

cogeno.output.OutputMixin.OutputFilterStopAt : public cogeno.output.OutputMixin.OutputFilte
Stop output at pattern.

Parameters

- `stop_at`: Stop pattern

cogeno.OutputFilterReplace()

cogeno.output.OutputMixin.OutputFilterReplace : public cogeno.output.OutputMixin.OutputFilt
Replace substring.

Parameters

- `old`: old substring to replace
- `new`: new substring which will replace the old substring. if new is None and the resulting line is empty it is deleted.
- `count`: (optional) the number of times to replace the old substring with the new substring

cogeno.OutputFilterReSub()

cogeno.output.OutputMixin.OutputFilterReSub : public cogeno.output.OutputMixin.OutputFilte

Substitude regular expression pattern.

Replace the leftmost non-overlapping occurrences of pattern in each line by the replacement repl.

Parameters

- `pattern`: Pattern for replacement. Pattern is a string that will be compiled to a pattern object.
- `repl`: Replacement. Repl can be a string or a function. If repl is a function, it is called for every non-overlapping occurrence of pattern.
- `count`: (optional) maximum number of pattern occurrences to be replaced

cogeno.OutputFilterTemplateSubstitude()

cogeno.output.OutputMixin.OutputFilterTemplateSubstitude : public cogeno.output.OutputMixi

Substitude template placeholders.

Template placeholder substitution supports $-based substitutions, using the following rules:

- $$ is an escape; it is replaced with a single $.
- $identifier names a substitution placeholder matching a mapping key of "identifier". By default, "identifier" is restricted to any case-insensitive ASCII alphanumeric string (including underscores) that starts with an underscore or ASCII letter. The first non-identifier character after the $ character terminates this placeholder specification.
- ${identifier} is equivalent to $identifier. It is required when valid identifier characters follow the placeholder but are not part of the placeholder, such as "${noun}ification".

At least up to 4 nesting levels of placeholders are substituded, e.g.:

- ${placeholder_level1} : mapping = 'placeholder_level1' : 'holder'
- ${place${placeholder_level1}_level2} : mapping = 'placeholder_level2' : 'placeholder'
- ${${placeholder_level2}_level3} : mapping = 'placeholder_level3' : 'placeholder_level'
- ${${placeholder_level3}4} : mapping = 'placeholder_level4' : 'success'

If more than one placeholder patterns are provided the substitution is done for each pattern sequencing through the patterns list.

Parameters

- `mapping`: Mapping is any dictionary-like object with keys that match the template placeholders.
- `patterns`: (optional) Patterns is a list of regular expressions describing the pattern for non-braced placeholders.

The cogeno module also provides a set of convenience functions:

5.1.2 Code generator

cogeno.cogeno_state()

cogeno::generator::CodeGenerator.cogeno_state(self)
numeric cogeno state id

5.1.3 Code generation module import

cogeno.import_module(name)

cogeno::importmodule::ImportMixin.import_module(self, name)
Import a Cogeno module.

Import a module.

Module file is searched in current directory or modules directories.

Parameters

- `name`: Module to import. Specified without any path.

See *Cogeno modules* for the available modules.

5.1.4 Template file inclusion

cogeno.out_include(include_file)

cogeno::include::IncludeMixin.out_include(self, include_file)
Write the text from include_file to the output.

The `include_file` is processed by cogeno. Inline code generation in `include_file` can access the globals defined in the `including source file` before inclusion. The `including source file` can access the globals defined in the `include_file` (after inclusion).

Parameters

- `include_file`: Path of include file, either absolute path or relative to current directory or relative to templates directory (e.g. 'templates/drivers/simple_tmpl.c')

cogeno.guard_include()

cogeno::include::IncludeMixin.guard_include(self)
Prevent the current file to be included by `cogeno.out_include()` when called the next time.

5.1.5 Error handling

cogeno.error(msg [, frame_index=0] [, snippet_lineno=0])

cogeno::error::ErrorMixin.error(self, msg='Error raised by cogeno generator.', frame_index=
Raise Error exception.

Extra information is added that maps the python snippet line seen by the Python interpreter to the line of the file that inlines the python snippet.

Parameters

- `msg`: [optional] exception message
- `frame_index`: [optional] Call frame index. The call frame offset of the function calling *error()*. Zero if directly called in a snippet. Add one for every level of function call.
- `lineno`: [optional] line number within template

5.1.6 Log output

cogeno.log(message, message_type=None, end="n", logonly=True)

cogeno::log::LogMixin.log(self, message, message_type=None, end="\, logonly=True)
Print message and write to log file.

Parameters

- `message`: Message
- `message_type`: If given will be prepended to the message
- `end`: Character to put at the end of the message. '\n' by default.
- `logonly`: Only write to logfile. True by default.

cogeno.msg(message)

cogeno::log::LogMixin.msg(self, message)
Print message to stdout and log with a "message: " prefix.

See *LogMixin::log()*

Parameters

- `message`: Message

cogeno.warning(message)

cogeno::log::LogMixin.warning(self, message)
Print message to stdout and log with a "warning: " prefix.

See *LogMixin::log()*

Parameters

- `message`: Message

cogeno.prout(message, end="n")

cogeno::log::LogMixin.prout(self, message, end="\)
Print message to stdout and log.

See *LogMixin::log()*

Parameters

- `message`: Message
- `end`: Character to put at the end of the message. '\n' by default.

cogeno.prerr(message, end="n")

cogeno::log::LogMixin.prerr(self, message, end="\)
Print message to stderr and log with a "error: " prefix.

See *LogMixin::log()*

Parameters

- `message`: Message
- `end`: Character to put at the end of the message. '\n' by default.

See *Invoking Cogeno* for how to provide the path to the file used for logging.

5.1.7 Lock access

cogeno.lock()

cogeno::lock::LockMixin.lock(self)
Get the global cogeno lock.

```
try:
     with cogeno.lock().acquire(timeout = 10):
          ...
except cogeno.lock_timeout():
    cogeno.error(...)
except:
    raise
```

Return Lock object

cogeno.lock_timeout()

cogeno::lock::LockMixin.lock_timeout(self)
Lock timeout.

Return Lock timeout object

See *Invoking Cogeno* for how to provide the path to the file used for locking.

Lock object

cogeno::filelock::BaseFileLock.acquire(self, timeout=None, poll_intervall=0.05)

```
.. code-block:: python

    # You can use this method in the context manager (recommended)
    with lock.acquire():
pass

    # Or use an equivalent try-finally construct:
    lock.acquire()
    try:
pass
    finally:
lock.release()

:arg float timeout:
    The maximum time waited for the file lock.
    If ``timeout <= 0``, there is no timeout and this method will
    block until the lock could be acquired.
    If ``timeout`` is None, the default :attr:`~timeout` is used.

:arg float poll_intervall:
    We check once in *poll_intervall* seconds if we can acquire the
    file lock.

:raises Timeout:
    if the lock could not be acquired in *timeout* seconds.

.. versionchanged:: 2.0.0

    This method returns now a *proxy* object instead of *self*,
    so that it can be used in a with statement without side effects.
```

cogeno::filelock::BaseFileLock.release(self, force=False)

```
Please note, that the lock is only completely released, if the lock
counter is 0.

Also note, that the lock file itself is not automatically deleted.

:arg bool force:
    If true, the lock counter is ignored and the lock is released in
    every case.
```

cogeno.filelock.BaseFileLock.is_locked(self)

```
True if the object is currently acquired, False otherwise.

.. versionchanged:: 2.0.0

    This was previously a method and is now a property.
```

5.1.8 Options

cogeno.option()

cogeno::options::OptionsMixin.option(self, name)

Get option of actual context.

Return option value

Parameters

- `name`: Name of option

cogeno.options_add_argument(*args, **kwargs)

cogeno::options::OptionsMixin.options_add_argument(self, args, kwargs)

Add option arguments to option parser of actual context.

Cogeno modules may add arguments to the cogeno option parser. The argument variables given to cogeno are rescanned after new option arguments are provided.

```
def mymodule(cogeno):
    if not hasattr(cogeno, '_mymodule'):
        cogeno._mymodule = None

        cogeno.options_add_argument('-m', '--mymodule', metavar='FILE',
            dest='mymodule_file', action='store',
            type=lambda x: cogeno.option_is_valid_file(x),
            help='Load mymodule data from FILE.')

   if getattr(cogeno, '_mymodule') is not None:
       return cogeno._mymodule

   if cogeno.option('mymodule_file'):
       mymodule_file = cogeno.option('mymodule_file')
   else:
       cogeno.error(..., 2)

   ...
   cogeno._mymodule = ...
```

5.1.9 Path functions

cogeno.path_walk(top, topdown = False, followlinks = False)

cogeno::paths::PathsMixin.path_walk(top, topdown=False, followlinks=False)
Walk directory tree.

For each directory in the tree rooted at directory top (including top itself), it yields a 3-tuple (dirpath, dirnames, filenames):

- dirpath: the path to the directory.
- dirnames: list of the paths of the subdirectories in dirpath
- filenames: list of the paths of the non-directory files in dirpath

See Python docs for os.walk, exact same behavior but it yields Path() instances instead. From: http://ominian.com/2016/03/29/os-walk-for-pathlib-path/

Return yields a 3-tuple (dirpath, dirpathes, filepathes)

Parameters

- `top`: root of directory tree
- `topdown`: if topdown is True, the triple for a directory is generated before the triples for any of its subdirectories (directories are generated top-down). If topdown is False, the triple for a directory is generated after the triples for all of its subdirectories (directories are generated bottom-up).
- `followlinks`:

cogeno.rmtree(top)

cogeno::paths::PathsMixin.rmtree(top)
Delete an entire directory tree.

Parameters

- `top`: root of directory tree

cogeno.find_template_files(top, marker, suffix='.c')

cogeno::paths::PathsMixin.find_template_files(top, marker, suffix='.c')
Find template files.

Return List of template file pathes

Parameters

- `marker`: Marker as b'my-marker'
- `suffix`:

5.1.10 Standard streams

cogeno.set_standard_streams(self, stdout=None, stderr=None)

cogeno::redirectable::RedirectableMixin.set_standard_streams(self, stdout=None, stderr=None
Redirect status and error reporting.

Assign new files for standard out and/or standard error.

Parameters

- `stdout`:
- `stderr`:

5.1.11 Standard Modules - CMake

cogeno.cmake_variable(variable_name [, default="<unset>"])

cogeno::stdmodules::StdModulesMixin.cmake_variable(self, variable_name, default="<unset>")
Get the value of a CMake variable.

If variable_name is not provided to cogeno by CMake the default value is returned.

A typical set of CMake variables that are not available in the `CMakeCache.txt` file and have to be provided as defines to cogeno if needed:

- "PROJECT_NAME"
- "PROJECT_SOURCE_DIR"
- "PROJECT_BINARY_DIR"
- "CMAKE_SOURCE_DIR"
- "CMAKE_BINARY_DIR"
- "CMAKE_CURRENT_SOURCE_DIR"
- "CMAKE_CURRENT_BINARY_DIR"
- "CMAKE_CURRENT_LIST_DIR"
- "CMAKE_FILES_DIRECTORY"
- "CMAKE_PROJECT_NAME"
- "CMAKE_SYSTEM"
- "CMAKE_SYSTEM_NAME"
- "CMAKE_SYSTEM_VERSION"
- "CMAKE_SYSTEM_PROCESSOR"
- "CMAKE_C_COMPILER"
- "CMAKE_CXX_COMPILER"
- "CMAKE_COMPILER_IS_GNUCC"
- "CMAKE_COMPILER_IS_GNUCXX"

Return value

Parameters

- `variable_name`: Name of the CMake variable
- `default`: Default value

cogeno.cmake_cache_variable(variable_name [, default="<unset>"])

cogeno::stdmodules::StdModulesMixin.cmake_cache_variable(self, variable_name, default="<uns
Get the value of a CMake variable from CMakeCache.txt.

If variable_name is not given in `CMakeCache.txt` the default value is returned.

Return value

Parameters

- `variable_name`: Name of the CMake variable
- `default`: Default value

See *Invoking Cogeno* and *Build with Cogeno* for how to provide CMake variables to cogeno.

5.1.12 Standard Modules - config

cogeno.config_properties()

cogeno::stdmodules::StdModulesMixin.config_properties(self)
Get all config properties.

The property names are the ones config file.

Return A dictionary of config properties.

cogeno.config_property(property_name [, default="<unset>"])

cogeno::stdmodules::StdModulesMixin.config_property(self, property_name, default="<unset>"
Get the value of a configuration property fromthe config file.

If property_name is not given in .config the default value is returned.

Return property value

Parameters

- `property_name`: Name of the property
- `default`: Property value to return per default.

See *Invoking Cogeno* and *Build with Cogeno* for how to provide config variables to cogeno.

5.1.13 Standard Modules - Extended Device Tree Database

cogeno.edts()

cogeno::stdmodules::StdModulesMixin.edts(self, force_extract=False)
Get the extended device tree database.

Return Extended device tree database.

See *Invoking Cogeno* and *Build with Cogeno* for how to provide all files to enable cogeno to build the extended device tree database.

5.2 Cogeno modules

Cogeno modules provide supporting functions for code generation.

Some modules have to be imported to gain access to the module's functions and variables. The standard modules are accessible by convenience functions.

```
/* This file uses modules. */
...
/**
 * @code{.cogeno.py}
 * cogeno.import_module('my_special_module')
 * my_special_module.do_everything():
 * @endcode{.cogeno.py}
 */
/** @code{.cogeno.ins}@endcode */
...
```

5.2.1 C code generation (ccode)

Description

The ccode module supports code generation for the C language.

To use the module in inline code generation import it by:

```
cogeno.import_module('ccode')
```

In case you want to use the ccode module in another Python project import it by:

```
import cogeno.modules.ccode
```

Cogeno invocation options

There are **NO** Cogeno invocation options.

Code generation functions

cogeno.modules.ccode.**outl_config_guard**(*property_name*)
Write #if guard for config property to output.

If there is a configuration property of the given name the property value is used as guard value, otherwise it is set to 0.

Parameters

- property_name: Property name

cogeno.modules.ccode.**outl_config_unguard**(*property_name*)
Write #endif guard for config property to output.

This is the closing command for outl_guard_config().

Parameters

- property_name: Property name

cogeno.modules.ccode.**out_comment**(*s*, *blank_before=True*)
Write 's' as a comment.

Parameters

- s: string, is allowed to have multiple lines.
- blank_before: True adds a blank line before the comment.

cogeno.modules.ccode.**out_edts_defines**(*prefix='EDT_'*)
Write EDTS database properties as C defines.

Parameters

- prefix: Define label prefix. Default is 'EDT_'.

5.2.2 CMake support (cmake)

Description

The CMake module provides access to CMake variables and the CMake cache.

You may get access to the database by *cogeno.cmake()*.

Cogeno invocation options

--cmake:cache FILE Use CMake variables from CMake cache FILE.

--cmake:define [defxxx=valyyy ...] Define CMake variables to your generator code.

Code generation functions

There are convenience functions to access the CMake variables:

cogeno::stdmodules::StdModulesMixin.cmake(self)
Get the cmake variables database.

Return CMake variables database.

cogeno::stdmodules::StdModulesMixin.cmake_variable(self, variable_name, default="<unset>")
Get the value of a CMake variable.

If variable_name is not provided to cogeno by CMake the default value is returned.

A typical set of CMake variables that are not available in the `CMakeCache.txt` file and have to be provided as defines to cogeno if needed:

- "PROJECT_NAME"
- "PROJECT_SOURCE_DIR"
- "PROJECT_BINARY_DIR"
- "CMAKE_SOURCE_DIR"
- "CMAKE_BINARY_DIR"
- "CMAKE_CURRENT_SOURCE_DIR"
- "CMAKE_CURRENT_BINARY_DIR"
- "CMAKE_CURRENT_LIST_DIR"
- "CMAKE_FILES_DIRECTORY"
- "CMAKE_PROJECT_NAME"
- "CMAKE_SYSTEM"
- "CMAKE_SYSTEM_NAME"
- "CMAKE_SYSTEM_VERSION"
- "CMAKE_SYSTEM_PROCESSOR"
- "CMAKE_C_COMPILER"
- "CMAKE_CXX_COMPILER"
- "CMAKE_COMPILER_IS_GNUCC"
- "CMAKE_COMPILER_IS_GNUCXX"

Return value

Parameters

- `variable_name`: Name of the CMake variable
- `default`: Default value

cogeno::stdmodules::StdModulesMixin.cmake_cache_variable(self, variable_name, default="<uns
Get the value of a CMake variable from CMakeCache.txt.

If variable_name is not given in `CMakeCache.txt` the default value is returned.

Return value

Parameters

- `variable_name`: Name of the CMake variable
- `default`: Default value

The CMake database may be used directly to get access to the CMake variables:

cogeno.modules.cmake.CMakeDB : public object

5.2.3 Config/ Kconfig support (config)

Description

The config module provides access to config properties and the config database.

You may get access to the database by *`cogeno.configs()`*.

Cogeno invocation options

--config:db FILE Write or read config database to/ from FILE.

--config:file FILE Read configuration variables from this FILE.

--config:kconfig-file FILE Top-level Kconfig FILE (default: Kconfig).

--config:kconfig-srctree DIR Kconfig files are looked up relative to the srctree DIR (unless absolute paths are used), and .config files are looked up relative to the srctree DIR if they are not found in the current directory.

--config:kconfig-defines DEFINE [DEFINE ...] Define variable to Kconfig. We allow multiple.

--config:inputs FILE [FILE ...] Read configuration file fragment from FILE. We allow multiple.

Code generation functions

There are convenience functions to access the config properties:

cogeno::stdmodules::StdModulesMixin.configs(self, force_extract=False)
Get the configuration variables database.

Return Configuration variables database.

cogeno::stdmodules::StdModulesMixin.config_properties(self)
Get all config properties.

The property names are the ones config file.

Return A dictionary of config properties.

cogeno::stdmodules::StdModulesMixin.config_property(self, property_name, default="<unset>"
Get the value of a configuration property fromthe config file.

If property_name is not given in .config the default value is returned.

Return property value

Parameters

- `property_name`: Name of the property
- `default`: Property value to return per default.

The config database itself provides a more rich set of methods for configuration access:

cogeno.modules.config.ConfigDB : public object

Public Functions

generate(*self*, *config_kconfig_file*, *config_kconfig_srctree*, *config_kconfig_defines=None*, *config_inputs=None*, *config_file=None*)
Generate config database.

Parameters

- `config_kconfig_file`: path of top-level Kconfig file
- `config_kconfig_srctree`: path for relative lookup of files
- `config_kconfig_defines`: optional, dictionary of variables to be defined to Kconfig
- `config_inputs`: optional, list of configuration fragment file pathes
- `config_file`: optional, config file path

load(*self*, *file_path*)
Load config database from JSON database file.

Parameters

- `file_path`: Path of JSON file

save(*self*, *file_path*)
Save config database to JSON database file.

Parameters

- `file_path`: Path of JSON file

extract (*self*, *config_file*)
Extract config database from config file.

The config file does not provide symbol information. *symbols()*, *symbol_info()* and *kconfig_files()* will not work on a database that was extracted from a config file.

Parameters

- `config_file`: Path of config file

property (*self*, *property_name*, *default='<unset>'*)
Get the value of a configuration property from .config.

If property_name is not given in the database the default value is returned.

Return property value

Parameters

- `property_name`: Name of the property (aka. symbol)
- `default`: Property value to return per default.

properties (*self*)
Get all config properties.

The property names are the ones that are in the database.

Return A dictionary of config properties.

symbols (*self*)
Get all config symbols.

Return A dictionary of config symbols.

symbol_info (*self*, *symbol_name*, *default='<unknown>'*)
Get the info of a configuration symbol.

If symbol_name is not available or Kconfig was not processed the default value is returned.

Return symbol info

Parameters

- `symbol_name`: Name of the configuration symbol
- `default`: Symbol info to return per default.

kconfig_files (*self*)
Get all Kconfig files.

Get all Kconfig files used to *generate()* the database.

Return A list of kconfig file pathes.

Generate code with config

Kconfig Guide

This is a high-level guide to Kconfig and how to use it with Cogeno. See *config code generation functions* for an API reference.

Introduction to Kconfig

Projects can be configured by Kconfig at build time to adapt them for specific application and platform needs. The goal is to support configuration without having to change any source code. Kconfig is the configuration system used by the Linux kernel and several other projects.

Configuration options (often called symbols) are defined in Kconfig files, which also specify dependencies between symbols that determine what configurations are valid. Symbols can be grouped into menus and sub-menus to keep the interactive configuration interfaces organized.

The usual output from Kconfig is a header file `autoconf.h` with macros that can be tested at build time. Kconfig itself keeps the values of the configuration symbols in a config file usually named `.config`. Kconfig can either read the config file and produce output or generate or change the config file using one of several interactive configuration interfaces.

The values of the configuration symbols can also be changed by hand editing a Kconfig file and let Kconfig update the config file and the output.

(K)config access from Cogeno

Cogeno includes a complete Kconfig system (provided by Kconfiglib) but no interactive configuration interfaces.

The Cogeno *config module* creates it´s own database of configuration symbol values. It may either

1) extract the symbol names and values from a config file provided to Cogeno as an option or
2) run the whole process of symbol and value generation based on Kconfig files, and config file and/ or config fragment files also provided to Cogeno as options.

In the first case only the name and value of the symbols are known to the *config module*, whereas in the second case additional symbol information is available from the Kconfig files.

Property access

Use `cogeno.config_property(<symbol>)()` to get the value of a single symbol.

Use *`cogeno.config_properties()`* to get a dictionary of symbols with their associated values.

5.2.4 Extended device tree support (edts)

Description

The EDTS database module extracts device tree information from the device tree specification. See the *Devicetree Guide* for more information on device tree.

The EDTS database is a key value store. The keys are pathes to the device tree information. The EDTS database may be loaded from a json file, stored to a json file or extracted from the DTS files and the bindings yaml files of the project. The EDTS database is automatically available to Cogeno scripts. It can also be used as a standalone tool.

You may get access to the database by `cogeno.edts()`.

In case you want to use the extended device tree database in another Python project import it by:

```
import cogeno.modules.edts
```

Cogeno invocation options

--edts:bindings-dirs DIR [DIR ...] Use bindings from bindings DIR for device tree extraction. We allow multiple.

--edts:bindings-exclude [DIR ...] [FILE ...] Exclude bindings DIR or FILE from usage for device tree extraction. We allow multiple.

--edts:bindings-no-default Do not add EDTS database's generic bindings to bindings by default.

--edts:db FILE Write or read EDTS database to/ from FILE.

--edts:dts FILE Write (see dts-pp-sources) or read device tree specification to/ from this FILE.

--edts:dts-pp-sources FILE [FILE ...] Generate the DTS file by pre-processing the DTS source FILE(s).

--edts:dts-pp-include-dirs DIR [DIR ...] Define include DIR(s) to pre-processor.

--edts:dts-pp-defines DEFINE [DEFINE ...] DEFINE variable to pre-processor.

Code generation functions

There is a convenience function to access the EDTS database:

cogeno::stdmodules::StdModulesMixin.edts(self, force_extract=False)
Get the extended device tree database.

Return Extended device tree database.

The EDTS database itself provides a more rich set of methods for database access:

edtsdb API

edtsdb is a Python module with the primary class: `EDTSDb`.

The basis for the edtsb module is the `edtlib` library.

EDTSDb

cogeno::modules::edtsdb::consumer::EDTSConsumerMixin.device_id_by_name(self, name)
Get device id of activated device with given name.

Return device id

Parameters

- name:

cogeno.modules.edtsdb.database.EDTSDb : public cogeno.modules.edtsdb.consumer.EDTSConsumerM
Extended DTS database.

Database schema:

```
_edts dict(
  'aliases': dict(alias) : sorted list(device-id)),
  'chosen': dict(chosen),
  'devices':  dict(device-id :  device-struct),
  'compatibles':  dict(compatible : sorted list(device-id)),
  ...
)

device-struct dict(
  'device-id' : device-id,
  'compatible' : list(compatible) or compatible,
  'label' : label,
  '<property-name>' : property-value
)

property-value
  for "boolean", "string", "int" -> string
  for 'array', "string-array", "uint8-array" -> dict(index : string)
  for "phandle-array" -> dict(index : device-id)
```

Database types:

- device-id: opaque id for a device (do not use for other purposes),
- compatible: any of [‘st,stm32-spi-fifo’, . . .] - ‘compatibe’ from <binding>.yaml
- label: any of [‘UART_0’, ‘SPI_11’, . . .] - label directive from DTS

Subclassed by cogeno.modules.edts.EDTSDatabase

Public Functions

__init__ (*self*, *args*, *kw*)

__getitem__ (*self*, *key*)

__iter__ (*self*)

__len__ (*self*)

The `EDTSDb` class includes (sub-classes) several mixin classes:

cogeno.modules.edtsdb.consumer.EDTSConsumerMixin : public object
ETDS Database consumer.

Methods for ETDS database usage.

Subclassed by *cogeno.modules.edtsdb.database.EDTSDb*

Public Functions

info (*self*)
Get info.

Return edts 'info' dict

Parameters

- `None`:

compatibles (*self*)
Get compatibles.

Return edts 'compatibles' dict

Parameters

- `None`:

aliases (*self*)
Get aliases.

Return edts 'aliases' dict

Parameters

- `None`:

chosen (*self*)
Get chosen.

Return edts 'chosen' dict

Parameters

- `None`:

device_ids_by_compatible (*self*, *compatibles*)
Get device ids of all activated compatible devices.

Return list of device ids of activated devices that are compatible

Parameters

- `compatibles`: compatible(s)

device_ids_by_dependency_order(*self*)
Get device ids of activated devices sorted by dependency ordinal.

Device ids of devices that are not activated but have an ordinal associated are set to None.

Return list of device ids of activated devices sorted by dependency ordinal

Parameters

- `None`:

device_id_by_name(*self*, *name*)
Get device id of activated device with given name.

Return device id

Parameters

- `name`:

device_id_by_alias(*self*, *alias*)
Get device id of activated device for given alias.

Return device id or None

Parameters

- `alias`:

device_id_by_chosen(*self*, *chosen*)
Get device id of activated device for given chosen.

Return device id or None

Parameters

- `chosen`:

device_name_by_id(*self*, *device_id*)
Get label/ name of a device by device id.

If the label is omitted, the name is taken from the node name (including unit address).

Return name

Parameters

- `device_id`:

device_property(*self*, *device_id*, *property_path*, *default='<unset>'*)
Get device tree property value of a device.

Return property value

Parameters

- `device_id`:

- `property_path`: Path of the property to access (e.g. 'reg/0/address', 'interrupts/prio', 'device_id', ...)
- `default`: Default value provided if device or property not available

device_properties(*self*, *device_id*)
Get all device tree properties of a device.

Return Dictionary of properties

Parameters

- `device_id`:

device_properties_flattened(*self*, *device_id*, *path_prefix=''*)
Device properties flattened to property path : value.

Return dictionary of property_path and property_value

Parameters

- `device_id`:
- `path_prefix`:

device_template_substitute(*self*, *device_id*, *template*, *presets={}*, *aliases={}*)
Substitude device property value placeholders in template.

Local placeholders may be defined with direct and indirect path resolution:

- ${<property_path>}
- ${path/${<property_path>}}
- ${path/${<device-id>:<property_path>}}

Global placeholders may also be defined with direct and indirect path resolution:

- ${<device-id>:<property_path>}
- ${${<property_path>}:<property_path>}
- ${${path/${<property_path>}}:<property_path>}
- ${${path/${<device-id>:<property_path>}}:<property_path>}
- ${${<device-id>:<property_path>}:<property_path>}
- ${${<device-id>:path/${<property_path>}}:<property_path>}
- ${${<device-id>:path/${<device-id>:<property_path>}}:<property_path>}

Parameters

- `device_id`:
- `template`:
- `presets`: dict of preset property-path : property value items either of the local form "<property_path>" : value or the global form "<device-id>:<property_path>" : value
- `aliases`: dict of property path alias : property path items.

load(*self*, *file_path*)
Load extended device tree database from JSON file.

Parameters

- `file_path`: Path of JSON file

_DeviceGlobalTemplate : public Template

Public Static Attributes

idpattern = r'/[_a-z0-9\-/,@:]*'

_DeviceLocalTemplate : public Template

Public Static Attributes

idpattern = r'[_a-z][_a-z0-9\-/,]*'

cogeno.modules.edtsdb.extractor.EDTSExtractorMixin : public object

ETDS Database extractor.

Methods for ETDS database extraction from DTS.

Subclassed by *cogeno.modules.edtsdb.database.EDTSDb*

Public Functions

extract (*self*, *dts_path*, *bindings_dirs*, *bindings_exclude*, *bindings_no_default*)

Extract DTS info to database.

Parameters

- `dts_path`: DTS file
- `bindings_dirs`: YAML file directories, we allow multiple
- `bindings_exclude`: YAML file directories and YAML files, we allow multiple
- `bindings_no_default`: Do not use default bindings

dts_path (*self*)

Device tree file path.

Return Device tree file path

bindings_dirs (*self*)

Bindings directories paths.

Return List of binding directories

dts_source (*self*)

DTS source code.

DTS source code of the loaded devicetree after merging nodes and processing /delete-node/ and /delete-property/.

Return DTS source code as string

cogeno.modules.edtsdb.provider.EDTSProviderMixin : public object

ETDS Database provider.

Methods for ETDS database creation.

Subclassed by *cogeno.modules.edtsdb.database.EDTSDb*

Public Functions

insert_alias (*self*, *alias_path*, *alias_value*)

Insert an alias.

Parameters

- `alias_path`: Alias
- `alias_value`: The value the alias aliases.

insert_chosen (*self*, *chosen_path*, *chosen_value*)

insert_child_property (*self*, *device_id*, *child_name*, *property_path*, *property_value*)

Insert property value for the child of a device id.

Parameters

- `device_id`:
- `child_name`:
- `property_path`: Path of the property to access (e.g. 'reg/0', 'interrupts/prio', 'label', ...)
- `property_value`: value

insert_device_property (*self*, *device_id*, *property_path*, *property_value*)

Insert property value for the device of the given device id.

Parameters

- `device_id`:
- `property_path`: Path of the property to access (e.g. 'reg/0', 'interrupts/prio', 'label', ...)
- `property_value`: value

save (*self*, *file_path*)

Write json file.

Parameters

- `file_path`: Path of the file to write

edtlib

class edtlib.EDT(*dts*, *bindings_dirs*, ***, *warn_reg_unit_address_mismatch=True*, *default_prop_types=False*, *support_fixed_partitions_on_any_bus=True*, *infer_binding_for_paths=None*, *vendor_prefixes=None*)

Represents a devicetree augmented with information from bindings.

These attributes are available on EDT objects:

nodes:
  A list of Node objects for the nodes that appear in the devicetree

compat2nodes:
  A collections.defaultdict that maps each 'compatible' string that appears
  on some Node to a list of Nodes with that compatible.

compat2okay:
  Like compat2nodes, but just for nodes with status 'okay'.

  For example, edt.compat2okay["bar"] would include the 'foo' and 'bar'
  nodes below.

```
    foo {
            compatible = "bar";
            status = "okay";
            ...
    };
    bar {
            compatible = "foo", "bar", "baz";
            status = "okay";
            ...
    }
```

label2node:
  A collections.OrderedDict that maps a node label to the node with
  that label.

chosen_nodes:
  A collections.OrderedDict that maps the properties defined on the
  devicetree's /chosen node to their values. 'chosen' is indexed by
  property name (a string), and values are converted to Node objects.
  Note that properties of the /chosen node which can't be converted
  to a Node are not included in the value.

dts_path:
  The .dts path passed to __init__()

dts_source:
  The final DTS source code of the loaded devicetree after merging nodes
  and processing /delete-node/ and /delete-property/, as a string

bindings_dirs:
  The bindings directory paths passed to __init__()

Public Functions

__init__(*self*, *dts*, *bindings_dirs*, *warn_file=None*, *warn_reg_unit_address_mismatch=True*, *bindings_exclude=[]*)

```
EDT(dts, bindings_dirs, warn_file=None, warn_reg_unit_address_mismatch=True,
    bindings_exclude=[])

dts:
  Path to devicetree .dts file

bindings_dirs:
  List of paths to directories containing bindings, in YAML format.
  These directories are recursively searched for .yaml files.

warn_file (default: None):
  'file' object to write warnings to. If None, sys.stderr is used.

warn_reg_unit_address_mismatch (default: True):
  If True, a warning is printed if a node has a 'reg' property where
  the address of the first entry does not match the unit address of the
  node

bindings_exclude:
  List of paths to bindings directories and/ or files that shall be excluded
  from usage.
```

get_node(*self*, *path*)

```
Returns the Node at the DT path or alias 'path'. Raises EDTError if the
path or alias doesn't exist.
```

chosen_nodes(*self*)

chosen_node(*self*, *name*)

```
Returns the Node pointed at by the property named 'name' in /chosen, or
None if the property is missing
```

dts_source(*self*)

__repr__(*self*)

scc_order(*self*)

```
Returns a list of lists of Nodes where all elements of each list
depend on each other, and the Nodes in any list do not depend
on any Node in a subsequent list.  Each list defines a Strongly
Connected Component (SCC) of the graph.

For an acyclic graph each list will be a singleton.  Cycles
will be represented by lists with multiple nodes.  Cycles are
not expected to be present in devicetree graphs.
```

Public Members

dts_path

bindings_dirs

nodes

label2node

compat2nodes

compat2okay

```
class Node:
    """
    Represents a devicetree node, augmented with information from bindings, and
    with some interpretation of devicetree properties. There's a one-to-one
    correspondence between devicetree nodes and Nodes.

    These attributes are available on Node objects:

    edt:
      The EDT instance this node is from

    name:
      The name of the node

    unit_addr:
      An integer with the ...@<unit-address> portion of the node name,
      translated through any 'ranges' properties on parent nodes, or None if
      the node name has no unit-address portion

    description:
      The description string from the binding for the node, or None if the node
      has no binding. Leading and trailing whitespace (including newlines) is
      removed.

    path:
      The devicetree path of the node

    label:
      The text from the 'label' property on the node, or None if the node has
      no 'label'

    labels:
      A list of all of the devicetree labels for the node, in the same order
      as the labels appear, but with duplicates removed.
      This corresponds to the actual devicetree source labels, unlike the
      "label" attribute, which is the value of a devicetree property named
      "label".

    parent:
      The Node instance for the devicetree parent of the Node, or None if the
      node is the root node

    children:
      A dictionary with the Node instances for the devicetree children of the
      node, indexed by name

    dep_ordinal:
      A non-negative integer value such that the value for a Node is
      less than the value for all Nodes that depend on it.

      The ordinal is defined for all Nodes including those that are not
      'enabled', and is unique among nodes in its EDT 'nodes' list.

    required_by:
      A list with the nodes that directly depend on the node

    depends_on:
      A list with the nodes that the node directly depends on
```

(continues on next page)

(continued from previous page)

```
status:
  The node's status property value, as a string, or "okay" if the node
  has no status property set. If the node's status property is "ok",
  it is converted to "okay" for consistency.

read_only:
  True if the node has a 'read-only' property, and False otherwise

matching_compat:
  The 'compatible' string for the binding that matched the node, or None if
  the node has no binding

binding_path:
  The path to the binding file for the node, or None if the node has no
  binding

compats:
  A list of 'compatible' strings for the node, in the same order that
  they're listed in the .dts file

regs:
  A list of Register objects for the node's registers

props:
  A collections.OrderedDict that maps property names to Property objects.
  Property objects are created for all devicetree properties on the node
  that are mentioned in 'properties:' in the binding.

aliases:
  A list of aliases for the node. This is fetched from the /aliases node.

clocks:
  A list of ControllerAndData objects for the clock inputs on the
  node, sorted by index. The list is empty if the node does not have a
  clocks property.

clock_outputs:
  A list of ControllerAndData objects for the clock outputs on the
  node, sorted by index. The list is empty if the node does not have a
  #clock-cells property.

gpio_leds:
  A list of ControllerAndData objects of the leds a gpio leds controller
  controls. The list is empty if the node is not a gpio leds controller or
  it does not have and gpio led children.

interrupts:
  A list of ControllerAndData objects for the interrupts generated by the
  node. The list is empty if the node does not generate interrupts.

pinctrls:
  A list of PinCtrl objects for the pinctrl-<index> properties on the
  node, sorted by index. The list is empty if the node does not have any
  pinctrl-<index> properties.

pinctrl_states:
  A list with the Node instances for the 'pinctrl-state' children of
```

(continues on next page)

(continued from previous page)

```
  a pin controller node. The list is empty if the node does not have any
  pinctrl state children.

pinctrl_gpio_ranges:
  A list of ControllerAndData objects of the gpio ranges a pin controller
  controls. The list is empty if the node is not a pin controller or no
  'gpio-ranges' are defined by the gpio nodes.

pincfgs:
  A list of PinCfg objects for the 'pinctrl-state' node. The list is
  empty if the node is not a 'pinctrl-state' node.

pin_controller:
  The pin controller for the node. Only meaningful for nodes representing
  pinctrl states.

bus:
  If the node is a bus node (has a 'bus:' key in its binding), then this
  attribute holds the bus type, e.g. "i2c" or "spi". If the node is not a
  bus node, then this attribute is None.

on_bus:
  The bus the node appears on, e.g. "i2c" or "spi". The bus is determined
  by searching upwards for a parent node whose binding has a 'bus:' key,
  returning the value of the first 'bus:' key found. If none of the node's
  parents has a 'bus:' key, this attribute is None.

bus_node:
  Like on_bus, but contains the Node for the bus controller, or None if the
  node is not on a bus.

flash_controller:
  The flash controller for the node. Only meaningful for nodes representing
  flash partitions.

partitions:
  A list of Partition objects of the partitions of the 'partitions' node of
  a flash. Only meaningful for nodes representing a flash - otherwise an
  empty list.

spi_cs_gpio:
  The device's SPI GPIO chip select as a ControllerAndData instance, if it
  exists, and None otherwise. See
  Documentation/devicetree/bindings/spi/spi-controller.yaml in the Linux kernel.
```

Public Functions

name(*self*)

unit_addr(*self*)

description(*self*)

path(*self*)

label(*self*)

labels(*self*)

parent (*self*)

children (*self*)

required_by (*self*)

depends_on (*self*)

status (*self*)

read_only (*self*)

aliases (*self*)

bus (*self*)

clocks (*self*)

on_bus (*self*)

flash_controller (*self*)

spi_cs_gpio (*self*)

__repr__ (*self*)

Public Members

on_bus

compats

matching_compat

binding_path

props

pinctrl_gpio_ranges

clock_outputs

gpio_leds

regs

partitions

name

pinctrl_states

pincfgs

pinctrls

interrupts

class edtlib.**Register**

Represents a register in the devicetree. Attributes:

These attributes are available on Register objects:

node:
  The Node instance this register is from

name:

(continues on next page)

(continued from previous page)

```
  The name of the register as given in the 'reg-names' property, or None if
  there is no 'reg-names' property

addr:
  The starting address of the register, in the parent address space, or None
  if #address-cells is zero. Any 'ranges' properties are taken into account.

size:
  The length of the register in bytes
```

Public Functions

__repr__(*self*)

class cogeno.modules.edtlib.**ControllerAndData**

```
Represents a set of data in a 'interrupts', 'pwms', etc. property
value, e.g. <&ctrl-1 4 0> in

    cs-gpios = <&ctrl-1 4 0 &ctrl-2 3 4>;

These attributes are available on ControllerAndData objects:

node:
  The Node instance the property appears on

controller:
  The Node instance for the controller (e.g. the controller the interrupt
  gets sent to for interrupts)

data:
  A dictionary that maps names from the *-cells key in the binding for the
  controller to data values, e.g. {"pin": 4, "flags": 0} for the example
  above.

  'interrupts = <1 2>' might give {"irq": 1, "level": 2}.

name:
  The name of the entry as given in
  'interrupt-names'/'gpio-names'/'pwm-names'/etc., or None if there is no
  *-names property
```

Public Functions

__repr__(*self*)

Generate code with edts

Devicetree Guide

This is a high-level guide to devicetree and how to use it with Cogeno. See *edtsdb API* for an API reference.

The guide is a partial copy of the Zephyr devicetree guide adapted to the specific devicetree capabilities of the Cogeno *EDTS database module*.

Introduction to devicetree

Tip: This is a conceptual overview of devicetree and how the Cogeno *EDTS database module* provides the devicetree information.

A *devicetree* is a hierarchical data structure that describes hardware. The Devicetree specification defines its source and binary representations. Devicetree may be used to describe the hardware available on a board, as well as that hardware's initial configuration.

There are two types of devicetree input files: *devicetree sources* and *devicetree bindings*. The sources contain the devicetree itself. The bindings describe its contents, including data types. The *EDTS database module* uses devicetree sources and bindings to produce a key-value database, which you can use to get information from your devicetree. The keys are pathes to the device tree information.

Here is a view of the process executed by the *EDTS database module*:

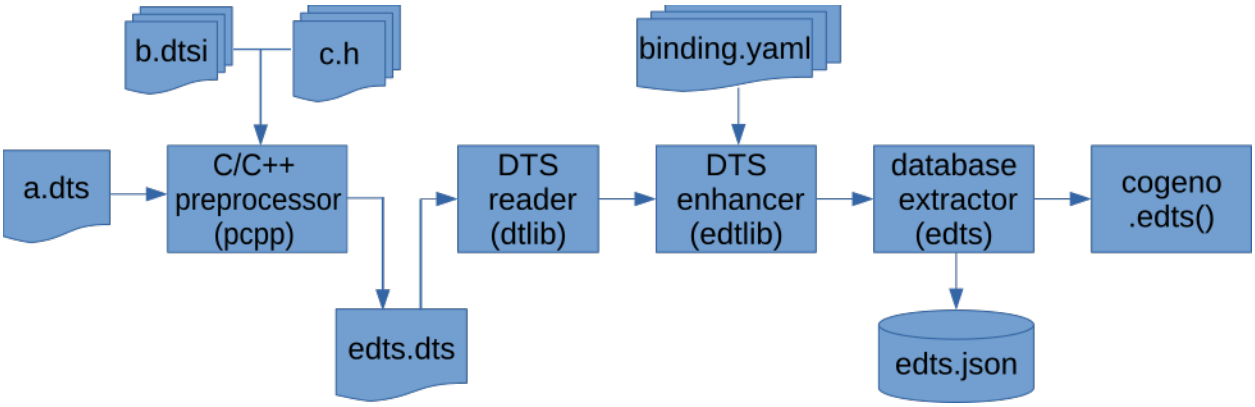


Fig. 1: EDTS database build flow

All Cogeno script snippets can access the EDTS database.

```
/* This is my C file. */
...
/**
 * @code{.cogeno.py}
 * edtsdb = cogeno.edts()
```

(continues on next page)

(continued from previous page)

```
 * ...
 * @endcode{.cogeno.py}
 */
/** @code{.cogeno.ins}@endcode */
...
```

Syntax and structure

As the name indicates, a devicetree is a tree. The human-readable text format for this tree is called DTS (for devicetree source), and is defined in the Devicetree specification.

Here is an example DTS file:

```
/dts-v1/;

/ {
        a-node {
                subnode_label: a-sub-node {
                        foo = <3>;
                };
        };
};
```

The `/dts-v1/;` line means the file's contents are in version 1 of the DTS syntax, which has replaced a now-obsolete "version 0".

The tree has three *nodes*:

1. A root node: `/`
2. A node named `a-node`, which is a child of the root node
3. A node named `a-sub-node`, which is a child of `a-node`

Nodes can be given *labels*, which are unique shorthands that can be used to refer to the labeled node elsewhere in the devicetree. Above, `a-sub-node` has label `subnode_label`. A node can have zero, one, or multiple node labels.

Devicetree nodes have *paths* identifying their locations in the tree. Like Unix file system paths, devicetree paths are strings separated by slashes (`/`), and the root node's path is a single slash: `/`. Otherwise, each node's path is formed by concatenating the node's ancestors' names with the node's own name, separated by slashes. For example, the full path to `a-sub-node` is `/a-node/a-sub-node`.

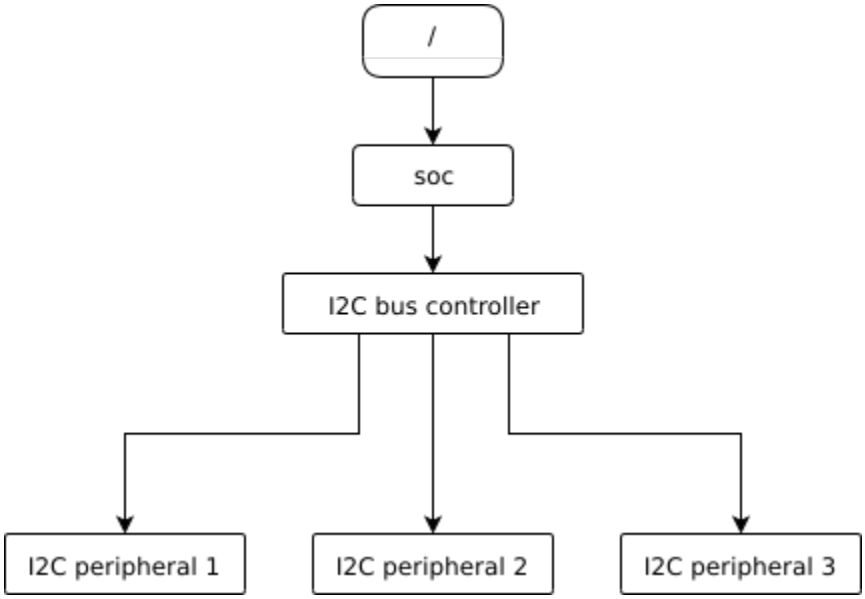
Devicetree nodes can also have *properties*. Properties are name/value pairs. Property values can be any sequence of bytes. In some cases, the values are an array of what are called *cells*. A cell is just a 32-bit unsigned integer.

Node `a-sub-node` has a property named `foo`, whose value is a cell with value 3. The size and type of `foo`'s value are implied by the enclosing angle brackets (< and >) in the DTS. See *Writing property values* below for more example property values.

In practice, devicetree nodes usually correspond to some hardware, and the node hierarchy reflects the hardware's physical layout. For example, let's consider a board with three I2C peripherals connected to an I2C bus controller on an SoC, like this:

Nodes corresponding to the I2C bus controller and each I2C peripheral would be present in the devicetree. Reflecting the hardware layout, the I2C peripheral nodes would be children of the bus controller node. Similar conventions exist for representing other types of hardware.

The DTS would look something like this:




```
/dts-v1/;

/ {
        soc {
                i2c-bus-controller {
                        i2c-peripheral-1 {
                        };
                        i2c-peripheral-2 {
                        };
                        i2c-peripheral-3 {
                        };
                };
        };
};
```

Properties are used in practice to describe or configure the hardware the node represents. For example, an I2C peripheral's node has a property whose value is the peripheral's address on the bus.

Here's a tree representing the same example, but with real-world node names and properties you might see when working with I2C devices.

This is the corresponding DTS:

```
/dts-v1/;

/ {
        soc {
                i2c@40003000 {
                        compatible = "nordic,nrf-twim";
                        label = "I2C_0";
                        reg = <0x40003000 0x1000>;

                        apds9960@39 {
                                compatible = "avago,apds9960";
                                label = "APDS9960";
                                reg = <0x39>;
```

(continues on next page)

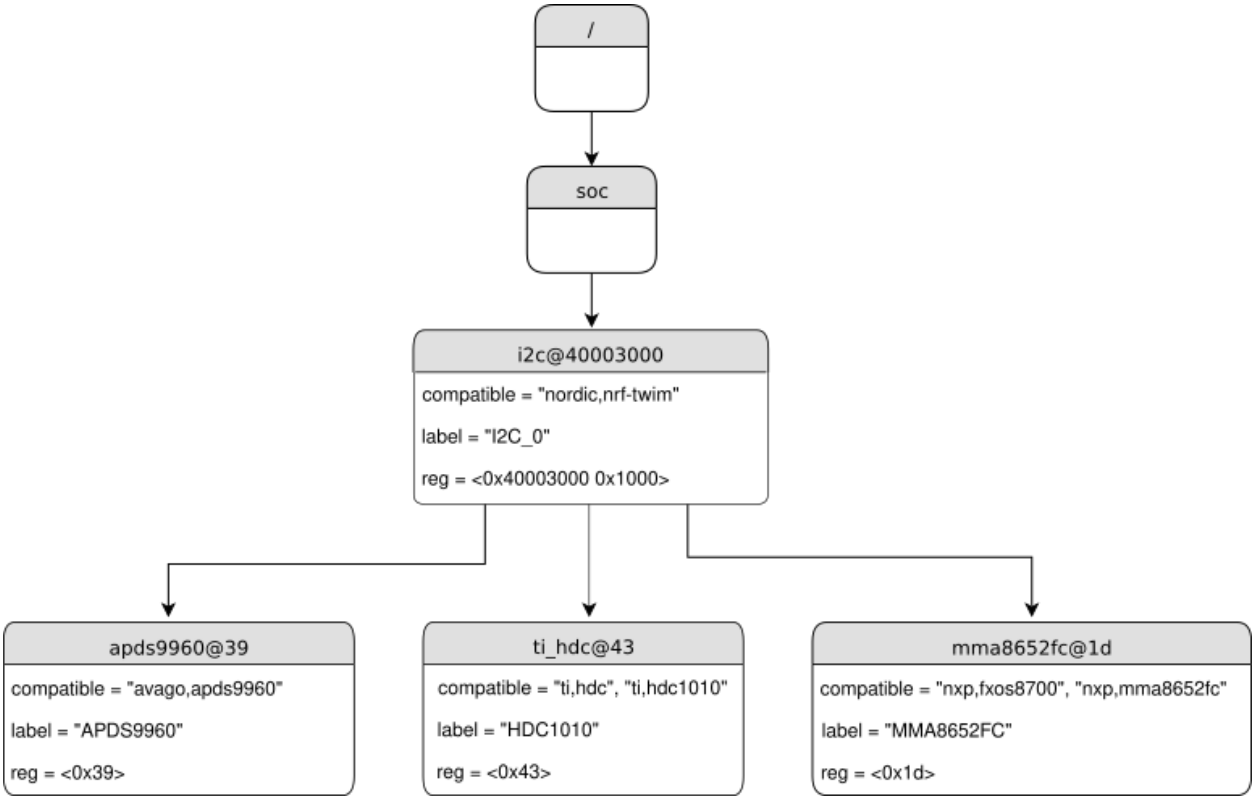


Fig. 2: I2C devicetree example with real-world names and properties. Node names are at the top of each node with a gray background. Properties are shown as "name=value" lines.

(continued from previous page)

```
                        };
                        ti_hdc@43 {
                                compatible = "ti,hdc", "ti,hdc1010";
                                label = "HDC1010";
                                reg = <0x43>;
                        };
                        mma8652fc@1d {
                                compatible = "nxp,fxos8700", "nxp,mma8652fc";
                                label = "MMA8652FC";
                                reg = <0x1d>;
                        };
                };
        };
};
```

In addition to showing more realistic names and properties, the above example introduces a new devicetree concept: unit addresses. Unit addresses are the parts of node names after an "at" sign (`@`), like `40003000` in `i2c@40003000`, or `39` in `apds9960@39`. Unit addresses are optional: the `soc` node does not have one.

Some more details about unit addresses and important properties follow.

Unit address examples

In devicetree, unit addresses give a node's address in the address space of its parent node. Here are some example unit addresses for different types of hardware.

Memory-mapped peripherals The peripheral's register map base address. For example, the node named `i2c@40003000` represents an I2C controller whose register map base address is 0x40003000.

I2C peripherals The peripheral's address on the I2C bus. For example, the child node `apds9960@39` of the I2C controller in the previous section has I2C address 0x39.

SPI peripherals An index representing the peripheral's chip select line number. (If there is no chip select line, 0 is used.)

Memory The physical start address. For example, a node named `memory@2000000` represents RAM starting at physical address 0x2000000.

Memory-mapped flash Like RAM, the physical start address. For example, a node named `flash@8000000` represents a flash device whose physical start address is 0x8000000.

Fixed flash partitions This applies when the devicetree is used to store a flash partition table. The unit address is the partition's start offset within the flash memory. For example, take this flash device and its partitions:

```
flash@8000000 {
    /* ... */
    partitions {
            partition@0 { /* ... */ };
            partition@20000 {  /* ... */ };
            /* ... */
    };
};
```

The node named `partition@0` has offset 0 from the start of its flash device, so its base address is 0x8000000. Similarly, the base address of the node named `partition@20000` is 0x8020000.

Important properties

Some important properties are:

compatible The name of the hardware device the node represents.

The recommended format is `"vendor,device"`, like `"avago,apds9960"`, or a sequence of these, like `"ti,hdc", "ti,hdc1010"`. The `vendor` part is an abbreviated name of the vendor. The file **:zephyr_file:`dts/bindings/vendor-prefixes.txt`** contains a list of commonly accepted `vendor` names. The `device` part is usually taken from the datasheet.

It is also sometimes a value like `gpio-keys`, `mmio-sram`, or `fixed-clock` when the hardware's behavior is generic.

The build system uses the compatible property to find the right bindings for the node. Device drivers use `devicetree.h` to find nodes with relevant compatibles, in order to determine the available hardware to manage.

The `compatible` property can have multiple values. Additional values are useful when the device is a specific instance of a more general family, to allow the system to match from most- to least-specific device drivers.

Within Zephyr's bindings syntax, this property has type `string-array`.

label The device's name according to Zephyr's device_model_api. The value can be passed to `device_get_binding()` to retrieve the corresponding driver-level struct device*. This pointer can then be passed to the correct driver API by application code to interact with the device. For example, calling `device_get_binding("I2C_0")` would return a pointer to a device structure which could be passed to I2C API functions like `i2c_transfer()`. The generated C header will also contain a macro which expands to this string.

reg Information used to address the device. The value is specific to the device (i.e. is different depending on the compatible property).

The `reg` property is a sequence of `(address, length)` pairs. Each pair is called a "register block". Here are some common patterns:

- Devices accessed via memory-mapped I/O registers (like `i2c@40003000`): `address` is usually the base address of the I/O register space, and `length` is the number of bytes occupied by the registers.
- I2C devices (like `apds9960@39` and its siblings): `address` is a slave address on the I2C bus. There is no `length` value.
- SPI devices: `address` is a chip select line number; there is no `length`.

You may notice some similarities between the `reg` property and common unit addresses described above. This is not a coincidence. The `reg` property can be seen as a more detailed view of the addressable resources within a device than its unit address.

status A string which describes whether the node is enabled.

The devicetree specification allows this property to have values `"okay"`, `"disabled"`, `"reserved"`, `"fail"`, and `"fail-sss"`. Only the values `"okay"` and `"disabled"` are currently relevant to Zephyr; use of other values currently results in undefined behavior.

A node is considered enabled if its status property is either `"okay"` or not defined (i.e. does not exist in the devicetree source). Nodes with status `"disabled"` are explicitly disabled. (For backwards compatibility, the value `"ok"` is treated the same as `"okay"`, but this usage is deprecated.) Devicetree nodes which correspond to physical devices must be enabled for the corresponding `struct device` in the Zephyr driver model to be allocated and initialized.

interrupts Information about interrupts generated by the device, encoded as an array of one or more *interrupt specifiers*. Each interrupt specifier has some number of cells. See section 2.4, *Interrupts and Interrupt Mapping*, in the Devicetree Specification release v0.3 for more details.

Zephyr's devicetree bindings language lets you give a name to each cell in an interrupt specifier.

Aliases and chosen nodes

There are two additional ways beyond *node labels* to refer to a particular node without specifying its entire path: by alias, or by chosen node.

Here is an example devicetree which uses both:

```
/dts-v1/;

/ {
      chosen {
              zephyr,console = &uart0;
      };

      aliases {
              my-uart = &uart0;
      };

      soc {
              uart0: serial@12340000 {
                      ...
              };
      };
};
```

The `/aliases` and `/chosen` nodes do not refer to an actual hardware device. Their purpose is to specify other nodes in the devicetree.

Above, `my-uart` is an alias for the node with path `/soc/serial@12340000`. Using its node label `uart0`, the same node is set as the value of the chosen `zephyr,console` node.

Zephyr sample applications sometimes use aliases to allow overriding the particular hardware device used by the application in a generic way. For example, blinky-sample uses this to abstract the LED to blink via the `led0` alias.

The `/chosen` node's properties are used to configure system- or subsystem-wide values. See devicetree-chosen-nodes for more information.

Input and output files

This section describes the input and output files shown in the figure at the *top of this introduction* in more detail.

Input files

There are three basic types of devicetree input files:

- sources (`.dts`)
- includes (`.dtsi`)
- bindings (`.yaml`)

As the devicetree files are preprocessed by the C/C++ style preprocessor any includes may de done.

- includes (`.h`, ...)

Usually every board has a `<board>.dts` file describing its hardware.

`<board>.dts` includes one or more `.dtsi` files. These `.dtsi` files describe the CPU or system-on-chip the board is made of, perhaps by including other `.dtsi` files. They can also describe other common hardware features shared by multiple boards. In addition to these includes, `<board>.dts` also describes the board's specific hardware.

The C/C++ style preprocessor is run on all devicetree files to expand macro references, and includes should generally be done with `#include <filename>` directives, even though DTS has a `/include/ "<filename>"` syntax.

Some project build systems may support to extend and/ or modify the `<board>.dts` by using *overlays*. *Overlays* are also DTS files; The build system concatenates the *overlay* files to the `<board>.dts`, with the *overlays* put last. This relies on DTS syntax which allows merging overlapping definitions of nodes in the devicetree.

Devicetree bindings (which are YAML files) are essentially glue. They describe the contents of devicetree sources, includes, and overlays in a way that allows to extract the *EDTS database* usable by Cogeno scripts. The *EDTS database module* provides a set of *generic bindings*.

Output files

`edts.dts` The final merged devicetree. This file is output by the *EDTS database module* as a debugging aid, and is unused otherwise.

`edts.json` The *EDTS database* as a .json export. This file, if available, is read in by the *EDTS database module* to avoid multiple evaluation of the devicetree. Dependency changes are regarded and lead to a full re-evaluation.

Writing property values

Here are some example ways to write property values in DTS format. Some specifics are skipped in the interest of keeping things simple; if you're curious about details, see the devicetree specification.

Arrays of 32-bit unsigned integers, or *cells*, can be written between angle brackets (< and >) and separated by spaces:

```
foo = <0xdeadbeef 1234 0>;
```

The `foo` property value is three cells with values 0xdeadbeef, 1234, and 0, in that order. Note that hexadecimal and decimal numbers are allowed and can be intermixed. Since Zephyr transforms DTS to C sources, it is not necessary to specify the endianness of an individual cell here.

64-bit integers are written as two 32-bit cells in big-endian order. The value 0xaaaa0000bbbb1111 would be written `<0xaaaa0000 0xbbbb1111>`.

Parentheses, arithmetic operators, and bitwise operators are allowed. The `bar` property contains a single cell with value 64:

```
bar = <2 * (1 << 5)>;
```

Strings are double quoted:

```
a-string = "hello, world!";
```

String arrays are separated by commas:

```
a-string-array = "string one", "string two", "string three";
```

Arrays of bytes are written in hexadecimal *without* leading `0x` between square brackets (`[` and `]`). Property `a-byte-array` is the three bytes 0x00, 0x01, 0xab, in that order:

```
a-byte-array = [00 01 ab];
```

Properties can refer to other nodes in the devicetree by their *phandles*. You can write a phandle using `&label`, like in this devicetree fragment:

```
baz: device@0 {
        /* ... */
};
device@1 {
        sibling = <&baz 1 2>;
        /* ... */
};
```

The `sibling` property of node `device@1` contains three cells:

- The `device@0` node's phandle. Each phandle occupies an entire cell. The `baz` label is used to write the phandle `&baz` inside the `sibling` property value.
- The values 1 and 2, each in its own cell, in that order.

Devicetree bindings

A devicetree on its own is only half the story for describing hardware. The devicetree format itself is relatively unstructured, and doesn't tell the Cogeno EDTS module which pieces of information in a particular devicetree are useful to be integrated into the EDTS database.

Devicetree bindings provide the other half of this information. Cogeno EDTS devicetree bindings are YAML files in a custom format (Cogeno uses the Zephyr bindings and extends them in some rare cases. It does not use the dt-schema tools used by the Linux kernel). Cogeno uses bindings when generating the EDTS database.

Mapping nodes to bindings

During the *EDTS database build flow*, the *EDTS database module* tries to map each node in the devicetree to a binding file. The *EDTS database module* only generates database entries for devicetree nodes which have matching bindings. Nodes are mapped to bindings by their *compatible properties*. Take the following node as an example:

```
bar-device {
     compatible = "foo-company,bar-device";
     /* ... */
};
```

The *EDTS database module* will try to map the `bar-device` node to a YAML binding with this `compatible:` line:

```
compatible: "foo-company,bar-device"
```

Built-in *generic bindings* are provided by the *EDTS database module*. Binding file names usually match their `compatible:` lines, so the above binding would be named `foo-company,bar-device.yaml`.

If a node has more than one string in its `compatible` property, the *EDTS database module* looks for compatible bindings in the listed order and uses the first match. Take this node as an example:

```
baz-device {
     compatible = "foo-company,baz-device", "generic-baz-device";
};
```

The `baz-device` node would get mapped to the binding for compatible `"generic-baz-device"` if the build system can't find a binding for `"foo-company,baz-device"`.

Nodes without compatible properties can be mapped to bindings associated with their parent nodes. For an example, see the `pwmleds` node in the bindings file format described below.

If a node describes hardware on a bus, like I2C or SPI, then the bus type is also taken into account when mapping nodes to bindings. See the comments near `on-bus:` in the bindings syntax for details.

Bindings file syntax

Below is a template (`binding-template.yaml`) that shows the bindings file syntax. It is taken from Zephyr.

```
description: |
    Free-form description of the device/node. Can have multiple
    lines/paragraphs.

    See https://yaml-multiline.info/ for formatting help.

# Used to map nodes to bindings
compatible: "manufacturer,device"

# The 'compatible' above would match this node:
#
#     device {
#         compatible = "manufacturer,device";
#         ...
#     };
#
# Assuming no binding has 'compatible: "manufacturer,device-v2"', it would also
# match this node:
#
#     device {
#         compatible = "manufacturer,device-v2", "manufacturer,device";
#         ...
#     };
#
# Strings in 'compatible' properties on nodes are tried from left to right, and
# the first binding found is used.
#
# If more than one binding for a compatible is found, an error is raised.
```

(continues on next page)

(continued from previous page)

```
# Bindings can include other files, which can be used to share common
# definitions between bindings.
#
# Included files are merged into bindings with a simple recursive dictionary
# merge. It is up to the binding author to make sure that the final merged
# binding is well-formed, though it is checked by the code as well.
#
# It is an error if a key appears with a different value in a binding and in a
# file it includes, with one exception: A binding can have 'required: true' for
# some property for which the included file has 'required: false' (see the
# description of 'properties' below). The 'required: true' from the binding
# takes precedence, allowing bindings to strengthen requirements from included
# files.
#
# Note that weakening requirements by having 'required: false' where the
# included file has 'required: true' is an error. This is meant to keep the
# organization clean.
#
# The file base.yaml contains definitions for many common properties. When
# writing a new binding, it is a good idea to check if base.yaml already
# defines some of the needed properties, and including it in that case. Note
# that you can make a property defined in base.yaml obligatory like this
# (taking 'reg' as an example):
#
#     reg:
#         required: true
#
# This relies on the dictionary merge to fill in the other keys for 'reg', like
# 'type'.
#
# When including multiple files, any overlapping 'required' keys on properties
# in the included files are ORed together. This makes sure that a
# 'required: true' is always respected.
include: other.yaml # or [other1.yaml, other2.yaml]

# If the node describes a bus, then the bus type should be given, like below
bus: <string describing bus type, e.g. "i2c">

# If the node appears on a bus, then the bus type should be given, like below.
#
# When looking for a binding for a node, the code checks if the binding for the
# parent node contains 'bus: <bus type>'. If it does, then only bindings with a
# matching 'on-bus: <bus type>' are considered. This allows the same type of
# device to have different bindings depending on what bus it appears on.
on-bus: <string describing bus type, e.g. "i2c">

# 'properties' describes properties on the node, e.g.
#
#   reg = <1 2>;
#   current-speed = <115200>;
#   label = "foo";
#
# This is used to check that required properties appear, and to
# control the format of output generated for them. Except for some
# special-cased properties like 'reg', only properties listed here will
# generate output.
#
```

(continues on next page)

(continued from previous page)

```
# A typical property entry looks like this:
#
#   <property name>:
#     required: <true | false>
#     type: <string | int | boolean | array | uint8-array | string-array |
#            phandle | phandles | phandle-array | path | compound>
#     description: <description of the property>
#     enum:
#       - <item1>
#       - <item2>
#       ...
#       - <itemN>
#     const: <string | int>
#     default: <default>
#
# These types are available:
#
#   - 'type: string' is for properties that are assigned a single string, like
#
#         ident = "foo";
#
#   - 'type: int' is for properties that are assigned a single 32-bit value,
#     like
#
#         frequency = <100>;
#
#   - 'type: boolean' is for properties used as flags that don't take a value,
#     like
#
#         hw-flow-control;
#
#     The macro generated for the property gets set to 1 if the property exists
#     on the node, and to 0 otherwise. When combined with 'required: true',
#     this type just forces the flag to appear on the node. The output will
#     always be 1 in that case.
#
#     Warning: Since a macro is always generated for 'type: boolean'
#     properties, don't use #ifdef in tests. Do this instead:
#
#         #if DT_SOME_BOOLEAN_PROP == 1
#
#   - 'type: array' is for properties that are assigned zero or more 32-bit
#     values, like
#
#         pin-config = <1 2 3>;
#
#   - 'type: uint8-array' is for properties that are assigned zero or more
#     bytes with the [] syntax, like
#
#         lookup-table = [89 AB CD EF];
#
#     Each byte is given in hex.
#
#     This type is called 'bytestring' in the Devicetree specification.
#
#   - 'type: string-array' if for properties that are assigned zero or more
#     strings, like
```

(continues on next page)

(continued from previous page)

```
#
#         idents = "foo", "bar", "baz";
#
#   - 'type: phandle' is for properties that are assigned a single phandle,
#     like
#
#         foo = <&label>;
#
#   - 'type: phandles' is for properties that are assigned zero or more
#     phandles, like
#
#         foo = <&label1 &label2 ...>;
#
#   - 'type: phandle-array' is for properties that take a list of phandles and
#     (possibly) 32-bit numbers, like
#
#         pwms = <&ctrl-1 1 2 &ctrl-2 3 4>;
#
#     This type requires that the property works in the standard way that
#     devicetree properties like pwms, clocks, *-gpios, and io-channels work.
#     Taking 'pwms' as an example, the final -s is stripped from the property
#     name, and #pwm-cells is looked up in the node for the controller
#     (&ctrl-1/&ctrl-2) to determine the number of data values after the
#     phandle. The binding for each controller must also have a *-cells key
#     (e.g. pwm-cells), giving names to data values. See below for an
#     explanation of *-cells.
#
#     A *-names (e.g. pwm-names) property can appear on the node as well,
#     giving a name to each entry (the 'pwms' example above has two entries,
#     <&ctrl-1 1 2> and <&ctrl-2 3 4>).
#
#     Because other property names are derived from the name of the property by
#     removing the final -s, the property name must end in -s. An error is
#     raised if it doesn't.
#
#     *-gpios properties are special-cased so that e.g. foo-gpios resolves to
#     #gpio-cells rather than #foo-gpio-cells.
#
#     All phandle-array properties support mapping through *-map properties,
#     e.g. gpio-map. See the devicetree spec.
#
#   - 'type: path' is for properties that are assigned a path. Usually, this
#     would be done with a path reference:
#
#         foo = &label;
#
#     Plain strings are accepted too, and are verified to be a path to an
#     existing node:
#
#         foo = "/path/to/some/node";
#
#   - 'type: compound' is a catch-all for more complex types, e.g.
#
#         foo = <&label>, [01 02];
#
# 'type: array' and the other array types also allow splitting the value into
# several <> blocks, e.g. like this:
```

(continues on next page)

(continued from previous page)

```
#
#     foo = <1 2>, <3 4>;                          // Okay for 'type: array'
#     foo = <&label1 &label2>, <&label3 &label4>; // Okay for 'type: phandles'
#     foo = <&label1 1 2>, <&label2 3 4>;          // Okay for 'type: phandle-array'
#     etc.
#
# The optional 'default:' setting gives a value that will be used if the
# property is missing from the device tree node. If 'default: <default>' is
# given for a property <prop> and <prop> is missing, then the output will be as
# if '<prop> = <default>' had appeared (except YAML data types are used for the
# default value).
#
# Note that it only makes sense to combine 'default:' with 'required: false'.
# Combining it with 'required: true' will raise an error.
#
# See below for examples of 'default:'. Putting 'default:' on any property type
# besides those used in the examples will raise an error.
properties:
    # Describes a property like 'current-speed = <115200>;'. We pretend that
    # it's obligatory for the example node and set 'required: true'.
    current-speed:
        type: int
        required: true
        description: Initial baud rate for bar-device

    # Describes an optional property like 'keys = "foo", "bar";'
    keys:
        type: string-array
        required: false
        description: Keys for bar-device

    # Describes an optional property like 'maximum-speed = "full-speed";
    # the enum specifies known values that the string property may take
    maximum-speed:
        type: string
        required: false
        description: Configures USB controllers to work up to a specific speed.
        enum:
            - "low-speed"
            - "full-speed"
            - "high-speed"
            - "super-speed"

    # Describes a required property '#address-cells = <1>';  the const
    # specifies that the value for the property is expected to be the value 1
    "#address-cells":
        type: int
        required: true
        const: 1

    int-with-default:
        type: int
        required: false
        default: 123

    array-with-default:
        type: array
```

(continues on next page)

(continued from previous page)

```
        required: false
        default: [1, 2, 3] # Same as 'array-with-default = <1 2 3>'

    string-with-default:
        type: string
        required: false
        default: "foo"

    string-array-with-default:
        type: string-array
        required: false
        default: ["foo", "bar"] # Same as 'string-array-with-default = "foo", "bar"'

    uint8-array-with-default:
        type: uint8-array
        required: false
        default: [0x12, 0x34] # Same as 'uint8-array-with-default = [12 34]'

# 'child-binding' can be used when a node has children that all share the same
# properties. Each child gets the contents of 'child-binding' as its binding
# (though an explicit 'compatible = ...' on the child node takes precedence, if
# a binding is found for it).
#
# The example below is for a binding for PWM LEDs, where the child nodes are
# required to have a 'pwms' property. It corresponds to this .dts structure
# (assuming the binding has 'compatible: "pwm-leds"'):
#
# pwmleds {
#         compatible = "pwm-leds";
#
#         red_pwm_led {
#                 pwms = <&pwm3 4 15625000>;
#         };
#         green_pwm_led {
#                 pwms = <&pwm3 0 15625000>;
#         };
#         ...
# };
child-binding:
    description: LED that uses PWM

    properties:
        pwms:
            type: phandle-array
            required: true

# 'child-binding' also works recursively. For example, the binding below would
# provide a binding for the 'grandchild' node in this .dts (assuming
# 'compatible: "foo"'):
#
# parent {
#         compatible = "foo";
#         child {
#                 grandchild {
#                         prop = <123>;
#                 };
#         };
```

(continues on next page)

(continued from previous page)

```
# }
child-binding:
    description: ...

    ...

    child-binding:
        description: ...

        properties:
            prop:
                type: int
                required: true

# If the binding describes an interrupt controller, GPIO controller, pinmux
# device, or any other node referenced by other nodes via 'phandle-array'
# properties, then *-cells should be given.
#
# To understand the purpose of *-cells, assume that some node has
#
#     pwms = <&pwm-ctrl 1 2>;
#
# , where &pwm-ctrl refers to a node whose binding is this file.
#
# The <1 2> part of the property value is called a *specifier* (this
# terminology is from the devicetree specification), and contains additional
# data associated with the GPIO. Here, the specifier has two cells, and the
# node pointed at by &gpio-ctrl is expected to have '#pwm-cells = <2>'.
#
# *-cells gives a name to each cell in the specifier. These names are used when
# generating identifiers.
#
# In this example, assume that 1 refers to a pin and that 2 is a flag value.
# This gives a *-cells assignment like below.
pwm-cells:
    - channel # name of first cell
    - period  # name of second cell

# If the specifier is empty (e.g. '#clock-cells = <0>'), then *-cells can
# either be omitted (recommended) or set to an empty array. Note that an empty
# array is specified as e.g. 'clock-cells: []' in YAML.

# As a special case, all *-gpio properties map to the key 'gpio-cells',
# regardless of prefix
gpio-cells:
    - pin
    - flags
```

Devicetree access from Cogeno

This guide describes Cogeno's *edtsdb API* for reading the devicetree information from the EDTS database. It assumes you're familiar with the concepts in *Introduction to devicetree* and *Devicetree bindings*. See *edtsdb API* for API reference documentation.

Device identifiers

To get information about a particular devicetree device node, you need a *device identifier* for it. This is just a key that refers to the device node.

These are the main ways to get a device identifier:

By path Use the the device node's `full path in the devicetree`, starting from the root node as device identifier (e.g. `/soc/i2c@40002000`). This is mostly useful if you happen to know the exact node you're looking for.

By node name Use `cogeno.edts().device_id_by_name(<node name>)` to get a device identifier from a device *node name* (e.g. `i2c@40002000`).

By node label Use `cogeno.edts().device_id_by_name(<node label name>)` to get a device identifier from the *node label* of a device node. The function first searches for a matching *node name* before looking into the *node labels* of a device node. *node labels* are often provided by SoC `.dtsi` to give nodes names that match the SoC datasheet, like `i2c1`, `spi2`, etc.

By label property Use `cogeno.edts().device_id_by_name(<node label property name>)` to get a device identifier from the *label property* of a device node. The function first searches for a matching *node name* before looking into the *node label* and finally checks the *label* property. *label* properties are often provided by board `.dtsi` to give device nodes names that match the board interface, like `I2C_1`, `SPI_2`, etc.

By alias Use `cogeno.edts().device_id_by_alias(<alias name>)` to get a device node identifier for a property of the special */aliases* node (e.g. `sensor-controller`).

By instance number Use `cogeno.edts().device_ids_by_compatible(<compatible>)[<instance number>]` to get the device identifier of an individual device node based on a matching compatible, but be careful doing so. See below.

By chosen node Use `cogeno.edts().device_id_by_chosen(<chosen property>)` to get a device identifier for a */chosen* node property (e.g. `zephyr,console`).

By parent Use `cogeno.edts().device_property(<device id>, 'parent')` to get a device identifier for a parent device node, starting from a device identifier you already have.

Two device identifiers which refer to the same device node are identical and can be used interchangeably.

Here's a DTS fragment for some imaginary hardware we'll return to throughout this file for examples:

```
/dts-v1/;

/ {

        aliases {
                sensor-controller = &i2c1;
        };

        soc {
                i2c1: i2c@40002000 {
                        compatible = "vnd,soc-i2c";
```

(continues on next page)

(continued from previous page)

```
                        label = "I2C_1";
                        reg = <0x40002000 0x1000>;
                        status = "okay";
                        clock-frequency = < 100000 >;
                };
        };
};
```

Here are a few ways to get the device identifier for the `i2c@40002000` node:

- `'/soc/i2c@40002000'`
- `cogeno.edts().device_id_by_name('i2c@40002000')`
- `cogeno.edts().device_id_by_name('i2c1')`
- `cogeno.edts().device_id_by_name('I2C_1')`
- `cogeno.edts().device_id_by_alias('sensor-controller')`
- `cogeno.edts().device_ids_by_compatible('vnd,soc_i2c')[x]` for some unknown number `x`.

Property access

The right API to use to read property values depends on the format the property values shall be provided.

Checking properties and values

You can use `cogeno.edts().device_property(<device id>, <property path>, None)` to check if a device node has a property <*property path*>. `None` will be returned if the property does not exist.

Single property

Use `cogeno.edts().device_property(<device id>, <property path>, <default value>)` to read a single property.

Multiple properties

Multiple properties at the same time can be read in several ways.

Use `cogeno.edts().device_properties(<device_id>)` to get a dictionary of dictionaries representing the properties of a device node.

Use `cogeno.edts().device_properties_flattened(<device_id>. <path prefix>)` to get the properties flattened to **one** key-value dictionary where the keys represent the property path.

Use `cogeno.edts().device_template_substitute(<device_id>, <template>, <presets>, <aliases>)` to get the property values requested by the placeholders in the template.

EDTS bindings

The EDTS database module uses *devicetree bindings* (a kind of data schema) to know what data to extract and to know the kind of data. A set of *generic devicetree bindings* in conjunction with project specific bindings control the extraction process.

The *generic bindings* are part of the EDTS database module.

- *EDTS Bindings Index*

5.2.5 Protocol buffer code generation (protobuf)

Description

The protobuf module supports code generation for protocol buffers.

To use the module in inline code generation import it by:

```
cogeno.import_module('protobuf')
```

In case you want to use the protobuf module in another Python project import it by:

```
import cogeno.modules.protobuf
```

Cogeno invocation options

--protobuf:db-dir DIR Write or read protocol buffer code databases to/ from DIR.

--protobuf:sources FILE [FILE ...] The *.proto source FILE(s) to generate the protocol buffer code for.

--protobuf:include-dirs DIR [DIR ...] Search for *.proto import files in the protocol buffer include DIR(s).

Code generation functions

cogeno.modules.protobuf.**proto_gen**(*generator='nanopb'*, *force_generate=False*)
Get protocol buffer code prepared for cogeno use.

Return Protocol buffer code database

Parameters

- `generator`: The protocol buffer generator to use to generate the protocol buffer code. One of 'nanopb', 'pbtools', 'protobluff', 'cpp', 'python'. Default is 'nanopb'.

- `force_generate`: Force generation of the protocol buffer code, even if already generated. Default is False.

Generate code with protobuf

Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data. There exist several implementations for a lot of programming languages.

The protobuf module supports the following code generators:

Language	Generator
C	nanopb
C	pbtools
C	protobluff
C++	cpp
Python	python

The protocol buffer code that is generated by a generator is added to the protocol buffer code database.

cogeno.modules.protobuf.ProtoCodeDb : public MutableMapping

Protocol buffers code database.

All generated code is cached in dictionary.

Usage:

```
cogeno.module_import('protobuf')
db = protobuf.protodb("nanopb")
db2 = protobuf.protodb("py")
cogeno.out(db['my_message.pb.h'])
cogeno.out(db('pb_encode.c'])
```

Public Functions

__init__(*self*, *generator=None*, *sources=None*, *include_dirs=None*, *dst_dir=None*)

Initialize protocol buffer code database.

Parameters

- `generator`: Any of 'nonopb', ...
- `sources`: *.proto sources
- `include_dirs`: Directories to search for *.proto import
- `dst_dir`: Destination directory for generated code

__getitem__(*self*, *key*)

__setitem__(*self*, *key*, *value*)

__delitem__(*self*, *key*)

__iter__(*self*)

__len__(*self*)

add_sources(*self*, *sources*)
Add sources to the list of sources for generation.

Return True in case new sources were added, False otherwise.

Parameters

- `sources`: List of source file pathes

add_include_dirs(*self*, *include_dirs*)
Add include directories to the list for generation.

Return True in case new include dirs were added, False otherwise.

Parameters

- `include_dirs`: List of include dir pathes

load(*self*)
Load generated code into database.

generate(*self*)
Generate protocol buffer code.

Note This routine will use grpcio-provided protoc if it exists, and is using the system-installed protoc as a fallback.

Return protoc invocation return value

Public Static Functions

protoc_gen_nanopb_path()

has_grpcio_protoc()
Check if grpcio-tools protoc is installed.

Return True if installed, False otherwise

Generate protobuf code using protobluff

There is no uptodate PyPI package available for protobluff. The protobluff plugin for protoc has to be build and installed from source as described in the protobluff documentation. The source code is available on GitHub.

On Ubuntu you might use this script (`setup_protobluff.sh`):

```
#! /usr/bin/env sh
# Install protobluff to user local

# Ubuntu
sudo apt install git gcc automake libtool protobuf-compiler libprotoc-dev libprotobuf-
↪dev check

# get the sources
git clone https://github.com/squidfunk/protobluff protobluff.git
cd protobluff.git

# configure
```

(continues on next page)

(continued from previous page)

```
./autogen.sh
./configure --prefix ${HOME}/.local

# build & test & install to user dir
make
make test
make install

# remove the git repository
cd ..
rm -rf protobluff.git
```

The protobuf module searches the pathes defined in the environment for protoc-gen-protobluff.

5.2.6 ReST/ RST code generation (rstcode)

Description

The rstcode module supports code generation for the reST (RST) markup language.

To use the module in inline code generation import it by:

```
cogeno.import_module('rstcode')
```

In case you want to use the rstcode module in another Python project import it by:

```
import cogeno.modules.rstcode
```

Cogeno invocation options

There are **NO** Cogeno invocation options.

Code generation functions

cogeno.modules.rstcode.**sanitize_target**(*target*)
Sanitize target string.

Replace weired characters in the target string. Prevent _ at start and end of target string.

Return Sanitized target string.

cogeno.modules.rstcode.**link_reference**(*target*)
Get target reference.

Get the target reference for target string. The target string is sanitized by sanitize_target() before.

Return Target reference.

cogeno.modules.rstcode.Node : public object
A node of a RST document.

Subclassed by *cogeno.modules.rstcode.BulletList*, *cogeno.modules.rstcode.CodeBlock*, *cogeno.modules.rstcode.Comment*, *cogeno.modules.rstcode.Document*, *cogeno.modules.rstcode.LinkTarget*, *cogeno.modules.rstcode.OrderedList*, *cogeno.modules.rstcode.Paragraph*, *cogeno.modules.rstcode.Section*, *cogeno.modules.rstcode.Table*, *cogeno.modules.rstcode.Text*

Public Functions

text_begin(*self*)
Get the text to be issued at beginning of node output.

Maybe overloaded by child classes to dynamically adapt the output text at *out()* processing.

Return text

text_end(*self*)
Get the text to be issued at end of node output.

Maybe overloaded by child classes to dynamically adapt the output text at *out()* processing.

Return text

add_child(*self*, *node*)
Add a `Node` object to the current node.

Return `True` in case of success.

Parameters

- `node`: *Node* to add as a child of this node.

out(*self*, *out_redirect=None*)
Output node content.

Parameters

- `output_redirect`: Optional - redirect output, default is cogeno.out.

cogeno.modules.rstcode.Text : public cogeno.modules.rstcode.Node
A pure text node.

Public Functions

__init__(*self*, *text*)
Initialise the text node.

Parameters

- `text`: Multiline text

cogeno.modules.rstcode.Comment : public cogeno.modules.rstcode.Node
A comment node.

Public Functions

__init__ (*self*, *text*)
Initialise the comment node.

Parameters

- `text`: Multiline text

cogeno.modules.rstcode.CodeBlock : public cogeno.modules.rstcode.Node
A code block node.

Public Functions

__init__ (*self*, *code*, *code_type='none'*)
Initialise the code block node.

Parameters

- `code`: Multiline code
- `code_type`: The type of code as understood by the code-block directive

cogeno.modules.rstcode.Paragraph : public cogeno.modules.rstcode.Node
A paragraph node.

Public Functions

__init__ (*self*, *text*)
Initialise the paragraph node.

Parameters

- `text`: Multiline text

cogeno.modules.rstcode.Section : public cogeno.modules.rstcode.Node
A section node.

Public Functions

__init__ (*self*, *title*, *depth=1*)
Initialise the section node.

Parameters

- `title`: Title of the section
- `depth`: Either depth of the section, default is 1, or section mark character (e.g. '#').

cogeno.modules.rstcode.BulletList : public cogeno.modules.rstcode.Node
A bullet list node.

Public Functions

__init__(*self*)
Initialise the bullet list node.

add_item(*self*, *text*)
Add new text block to the bullet list.

Parameters

- `text`: Multiline text

cogeno.modules.rstcode.OrderedList : public cogeno.modules.rstcode.Node
A ordered list node.

Public Functions

__init__(*self*)
Initialise the ordered list node.

add_item(*self*, *text*)
Add new text block to the ordered list.

Parameters

- `text`: Multiline text

cogeno.modules.rstcode.Table : public cogeno.modules.rstcode.Node
A table node.

Output will be done in csv-table style.

Public Functions

__init__(*self*, *title=''*, *headers=[]*, *widths=[]*)
Initialise the paragraph node.

Parameters

- `title`: the table title
- `headers`: list of header items in the table
- `widths`: list of column widths in the table

add_row(*self*, *row*)
Add new row to the table.

Parameters

- `row`: List of column items in the row.

cogeno.modules.rstcode.LinkTarget : public cogeno.modules.rstcode.Node
A link target node.

Public Functions

__init__(*self*, *target*)
Initialise the link target node.

The target string will be santized using sanitize_target().

Parameters

- `target`: Target

cogeno.modules.rstcode.Document : public cogeno.modules.rstcode.Node
A document node.

Public Functions

__init__(*self*, *file_path=None*)
Initialise the document node.

Parameters

- `file_path`: Optional, file to write the document to

out(*self*, *out_redirect=None*)
Output document content.

If no out_redirect is provided the content is written to the file_path specified on initialisation. If also the file_path is not specified the output is re-directed to cogeno.out().

Parameters

- `output_redirect`: Optional - redirect output, default is file_path.

5.2.7 Zephyr support (zephyr)

Description

The Zephyr module supports code generation for the Zephyr RTOS.

To use the module in inline code generation import it by:

```
cogeno.import_module('zephyr')
```

In case you want to use the Zephyr module in another Python project import it by:

```
import cogeno.modules.zephyr
```

Cogeno invocation options

There are **NO** Cogeno invocation options.

Code generation functions

cogeno.modules.zephyr.**str2ident**(*s*)
Converts 's' to a form suitable for (part of) an identifier.

Return identifier

Parameters

- s: string

cogeno.modules.zephyr.**device_name_by_id**(*device_id*)
Get device name from device id.

Return device name

Parameters

- device_id: device id

cogeno.modules.zephyr.**device_declare_single**(*device_config_symbol*, *driver_name*, *device_init*, *device_pm_control*, *device_level*, *device_prio*, *device_api*, *device_info*, *device_defaults={}*)
Declare a single device instance.

Generate device instances code for a device instance that:

- match the driver names that
- is activated ('status' = 'ok') in the board device tree file and that is
- configured by Kconfig.

The device name is derived from the device tree label property or - if not avalable - the node name.

Return True if device is declared, False otherwise

Parameters

- device_config_symbol: A configuration symbol for device instantiation. (e.g. 'CONFIG_SPI_0')
- driver_name: The name this instance of the driver can be looked up from user mode with device_get_binding().
- device_init: Address to the init function of the driver.
- device_pm_control: The device power management function
- device_level: The initialization level at which configuration occurs. Must be one of the following symbols, which are listed in the order they are performed by the kernel:
  - PRE_KERNEL_1: Used for devices that have no dependencies, such as those that rely solely on hardware present in the processor/SOC. These devices cannot use any kernel services during configuration, since they are not yet available.

  - PRE_KERNEL_2: Used for devices that rely on the initialization of devices initialized as part of the PRE_KERNEL_1 level. These devices cannot use any kernel services during configuration, since they are not yet available.
  - POST_KERNEL: Used for devices that require kernel services during configuration.
  - POST_KERNEL_SMP: Used for initialization objects that require kernel services during configuration after SMP initialization
  - APPLICATION: Used for application components (i.e. non-kernel components) that need automatic configuration. These devices can use all services provided by the kernel during configuration.

- `device_prio`: The initialization priority of the device, relative to other devices of the same initialization level. Specified as an integer value in the range 0 to 99; lower values indicate earlier initialization. Must be a decimal integer literal without leading zeroes or sign (e.g. 32), or an equivalent symbolic name (e.g. #define MY_INIT_PRIO 32 or e.g. CONFIG_KERNEL_INIT_PRIORITY_DEFAULT + 5).
- `device_api`: Identifier of the device api. (e.g. 'spi_stm32_driver_api')
- `device_info`: Device info template for device driver config, data and interrupt initialisation.
- `device_defaults`: Device default property values. `device_defaults` is a dictionary of property path : property value (e.g. { 'label' : 'My default label' }).

`cogeno.modules.zephyr.`**`device_declare_multi`**(*device_config_symbols*, *driver_names*, *device_inits*, *device_pm_controls*, *device_levels*, *device_prios*, *device_api*, *device_info*, *device_defaults={}*)

Declare multiple device instances.

Generate device instances code for all device instances that:

- match the driver names that
- are activated ('status' = 'ok') in the board device tree file and that are
- configured by Kconfig.

Parameters

- `device_config_symbols`: A list of configuration symbols for device instantiation. (e.g. ['CONFIG_SPI_0', 'CONFIG_SPI_1'])
- `driver_names`: A list of driver names for device instantiation. The list shall be ordered as the list of device configs. (e.g. ['SPI_0', 'SPI_1'])
- `device_inits`: A list of device initialisation functions or a single function. The list shall be ordered as the list of device configs. (e.g. 'spi_stm32_init')
- `device_pm_controls`: A list of device power management functions or a single function. The list shall be ordered as the list of device configs. (e.g. 'device_pm_control_nop')
- `device_levels`: A list of driver initialisation levels or one single level definition. The list shall be ordered as the list of device configs. (e.g. 'PRE_KERNEL_1')
- `device_prios`: A list of driver initialisation priorities or one single priority definition. The list shall be ordered as the list of device configs. (e.g. 32)
- `device_api`: Identifier of the device api. (e.g. 'spi_stm32_driver_api')
- `device_info`: Device info template for device driver config, data and interrupt initialisation.

- `device_defaults`: Device default property values. `device_defaults` is a dictionary of property path : property value.

Generate code with zephyr

Zephyr device declaration

The Zephyr module provides functions to generate device driver instantiations.

```
cogeno.import_module('zephyr')
```

The device delaration functions generate device instances code for all devices activated ('status' = 'ok') in the board device tree file matching the provided compatibles.

Most of the parameters aim at filling the DEVICE_AND_API_INIT macro. Other parameters are there to help code generation to fit driver specifics.

Instance code will only be generated if the Kconfig variable is set. The variable name is build with the device node label name (e.g: CONFIG_I2C_1).

Driver info templates

The device declaration functions work on templates that feature placeholder substitution.

Device instance property placeholders:

Placeholder	Substitution	Remark
${device-name}	device instance name	Name is generated by the declaration function.
${driver-name}	device instance driver name	Name is taken from the device tree node property 'label'
${device-data}	device instance data structure name	Name is generated by the declaration function.
${device-config-info}	device instance config structure name	Name is generated by the declaration function.
${device-config-irq}	device instance interrupt configuration funtion name	Name is generated by the declaration function.

Device instance device tree property placeholders:

Placeholder	Substitution	Remark
${[path to DTS property]}	device instance device	see below (*)

- The property path supports every node property that is documented in the node yaml bindings. It also supports yaml heuristics, like 'bus-master' and will use documented '"#cells"'.

Device tree property placeholders:

Placeholder	Substitution	Remark
${[device id]:[path to DTS property]}	device node property value	see below (*)

- The device node property is defined by the property path of the device given by the device id. The device id is usually also taken from a DTS property e.g. ${${clock/0/controller}:device-name}.

KConfig configuration parameter placeholders:

Placeholder	Substitution	Remark
${CONFIG_[configuration parameter]}	KConfig configuration parameter value	

Driver info template Example:

'c' template code between triple quotes ("""" """") that should provide the expected code to be generated for the driver structures.

```
"""
#if CONFIG_SPI_STM32_INTERRUPT
DEVICE_DECLARE(${device-name});
static void ${device-config-irq}(struct device *dev)
{
        IRQ_CONNECT(${interrupts/0/irq}, ${interrupts/0/priority}, \\
                    spi_stm32_isr, \\
                    DEVICE_GET(${device-name}), 0);
        irq_enable(${interrupts/0/irq});
}
#endif
static const struct spi_stm32_config ${device-config-info} = {
        .spi = (SPI_TypeDef *)${reg/0/address/0},
        .pclken.bus = ${clocks/0/bus},
        .pclken.enr = ${clocks/0/bits},
```

(continues on next page)

(continued from previous page)

```
#if CONFIG_SPI_STM32_INTERRUPT
        .config_irq = ${device-config-irq},
#endif
};
static struct spi_stm32_data ${device-data} = {
        SPI_CONTEXT_INIT_LOCK(${device-data}, ctx),
        SPI_CONTEXT_INIT_SYNC(${device-data}, ctx),
};
"""
```

Declaration of a single device instance

cogeno.modules.zephyr.**device_declare_single**(*device_config_symbol*, *driver_name*, *device_init*, *device_pm_control*, *device_level*, *device_prio*, *device_api*, *device_info*, *device_defaults={}*)

Declare a single device instance.

Generate device instances code for a device instance that:

- match the driver names that
- is activated ('status' = 'ok') in the board device tree file and that is
- configured by Kconfig.

The device name is derived from the device tree label property or - if not avalable - the node name.

Return True if device is declared, False otherwise

Parameters

- `device_config_symbol`: A configuration symbol for device instantiation. (e.g. 'CONFIG_SPI_0')
- `driver_name`: The name this instance of the driver can be looked up from user mode with device_get_binding().
- `device_init`: Address to the init function of the driver.
- `device_pm_control`: The device power management function
- `device_level`: The initialization level at which configuration occurs. Must be one of the following symbols, which are listed in the order they are performed by the kernel:
  - PRE_KERNEL_1: Used for devices that have no dependencies, such as those that rely solely on hardware present in the processor/SOC. These devices cannot use any kernel services during configuration, since they are not yet available.
  - PRE_KERNEL_2: Used for devices that rely on the initialization of devices initialized as part of the PRE_KERNEL_1 level. These devices cannot use any kernel services during configuration, since they are not yet available.
  - POST_KERNEL: Used for devices that require kernel services during configuration.
  - POST_KERNEL_SMP: Used for initialization objects that require kernel services during configuration after SMP initialization

  - APPLICATION: Used for application components (i.e. non-kernel components) that need automatic configuration. These devices can use all services provided by the kernel during configuration.

- `device_prio`: The initialization priority of the device, relative to other devices of the same initialization level. Specified as an integer value in the range 0 to 99; lower values indicate earlier initialization. Must be a decimal integer literal without leading zeroes or sign (e.g. 32), or an equivalent symbolic name (e.g. #define MY_INIT_PRIO 32 or e.g. CONFIG_KERNEL_INIT_PRIORITY_DEFAULT + 5).
- `device_api`: Identifier of the device api. (e.g. 'spi_stm32_driver_api')
- `device_info`: Device info template for device driver config, data and interrupt initialisation.
- `device_defaults`: Device default property values. `device_defaults` is a dictionary of property path : property value (e.g. { 'label' : 'My default label' }).

Declaration of multiple device instances

`cogeno.modules.zephyr.`**`device_declare_multi`**(*device_config_symbols*, *driver_names*, *device_inits*, *device_pm_controls*, *device_levels*, *device_prios*, *device_api*, *device_info*, *device_defaults={}*)

Declare multiple device instances.

Generate device instances code for all device instances that:

- match the driver names that
- are activated ('status' = 'ok') in the board device tree file and that are
- configured by Kconfig.

Parameters

- `device_config_symbols`: A list of configuration symbols for device instantiation. (e.g. ['CONFIG_SPI_0', 'CONFIG_SPI_1'])
- `driver_names`: A list of driver names for device instantiation. The list shall be ordered as the list of device configs. (e.g. ['SPI_0', 'SPI_1'])
- `device_inits`: A list of device initialisation functions or a single function. The list shall be ordered as the list of device configs. (e.g. 'spi_stm32_init')
- `device_pm_controls`: A list of device power management functions or a single function. The list shall be ordered as the list of device configs. (e.g. 'device_pm_control_nop')
- `device_levels`: A list of driver initialisation levels or one single level definition. The list shall be ordered as the list of device configs. (e.g. 'PRE_KERNEL_1')
- `device_prios`: A list of driver initialisation priorities or one single priority definition. The list shall be ordered as the list of device configs. (e.g. 32)
- `device_api`: Identifier of the device api. (e.g. 'spi_stm32_driver_api')
- `device_info`: Device info template for device driver config, data and interrupt initialisation.
- `device_defaults`: Device default property values. `device_defaults` is a dictionary of property path : property value.

Example:

```
/**
 * @code{.cogeno.py}
 * cogeno.import_module('zephyr')
 *
 * device_configs = ['CONFIG_SPI_{}'.format(x) for x in range(1, 4)]
 * driver_names = ['SPI_{}'.format(x) for x in range(1, 4)]
 * device_inits = 'spi_stm32_init'
 * device_pm_controls = 'device_pm_control_nop'
 * device_levels = 'POST_KERNEL'
 * device_prios = 'CONFIG_SPI_INIT_PRIORITY'
 * device_api = 'spi_stm32_driver_api'
 * device_info = \
 * """
 * #if CONFIG_SPI_STM32_INTERRUPT
 * DEVICE_DECLARE(${device-name});
 * static void ${device-config-irq}(struct device *dev)
 * {
 *         IRQ_CONNECT(${interrupts/0/irq}, ${interrupts/0/priority}, \\
 *                     spi_stm32_isr, \\
 *                     DEVICE_GET(${device-name}), 0);
 *         irq_enable(${interrupts/0/irq});
 * }
 * #endif
 * static const struct spi_stm32_config ${device-config-info} = {
 *         .spi = (SPI_TypeDef *)${reg/0/address/0},
 *         .pclken.bus = ${clocks/0/bus},
 *         .pclken.enr = ${clocks/0/bits},
 * #if CONFIG_SPI_STM32_INTERRUPT
 *         .config_irq = ${device-config-irq},
 * #endif
 * };
 * static struct spi_stm32_data ${device-data} = {
 *         SPI_CONTEXT_INIT_LOCK(${device-data}, ctx),
 *         SPI_CONTEXT_INIT_SYNC(${device-data}, ctx),
 * };
 * """
 *
 * zephyr.device_declare_multi( \
 *     device_configs,
 *     driver_names,
 *     device_inits,
 *     device_pm_controls,
 *     device_levels,
 *     device_prios,
 *     device_api,
 *     device_info)
 * @endcode{.cogeno.py}
 */
/** @code{.codeins}@endcode */
```

5.3 Code Generation Templates

Code generation templates provide sophisticated code generation functions.

Templates are simply text files. They may be hierarchical organized. There is always one top level template. All the other templates have to be included to gain access to the template's functions and variables.

A template file usually contains normal text and templating commands intermixed. A bound sequence of templating commands is called a script snippet. As a special case a template file may be a script snippet as a whole.

Cogeno supports two flavours of script snippets: Python and Jinja2. A script snippet has to be written in one of the two scripting languages. Within a template file snippets of different language can coexist.

5.3.1 Template Snippets

```
/* This file uses templates. */
...
/**
 * @code{.cogeno.py}
 * template_in_var = 1
 * cogeno.out_include('templates/template_tmpl.c')
 * if template_out_var not None:
 *     cogeno.outl("int x = %s;" % template_out_var)
 * @endcode{.cogeno.py}
 */
/** @code{.cogeno.ins}@endcode */
...
```

CHAPTER

SIX

COGENO EXTENSIONS

6.1 Search for Cogeno extensions

On invocation Cogeno searches the list of directories given in the `--extensions DIR [DIR...]` option for Cogeno extensions.

If a `cogeno/<extension_name>.py` Python module is found in an extension directory the module is imported by Cogeno. All Python commands within the <*extension_name*> module are executed. Scripts and script snippets can import the module by it's name <*extension_name*>.

There may be several modules within the `cogeno` directory. All of them will be imported.

6.2 Script Cogeno extension modules

The `cogeno/<extension_name>.py` file is a regular Python module. All of Python can be used.

6.2.1 Add options to the Cogeno invocation

A typical use case is to add further options to the Cogeno invocation:

```
from pathlib import Path

import cogeno

component_path = Path(__file__).parent.parent

cogeno.options_argv_append('--edts:dts-pp-defines', 'MY_EXTENSION=1')
cogeno.options_argv_append('--edts:dts-pp-include-dirs', str(component_path.joinpath(
↪'include')))
cogeno.options_argv_append('--edts:bindings-dirs', str(component_path.joinpath('edts/
↪bindings')))
cogeno.options_argv_append('--protobuf:sources', str(component_path.joinpath('proto/
↪myproto.proto')))
```

6.2.2 Manage extensions hirarchy with Kconfig variables

Another use case is to have a hirarchy of extensions possibly controlled by Kconfig variables.

Toplevel <extension directory>/`cogeno/<toplevel_name>.py` file:

```
from pathlib import Path

import cogeno

component_path = Path(__file__).parent.parent

if cogeno.config_property('CONFIG_MY_SUB_EXTENSION', None):
    cogeno.import_extensions(component_path.joinpath('whatever/sub-extension'))

...
```

Sublevel <extension directory/whatever/sub-extension>/`cogeno/<sub_extension_name>.py` file:

```
from pathlib import Path

import cogeno

component_path = Path(__file__).parent.parent

cogeno.options_argv_append('--edts:dts-pp-defines', 'MY_SUB_EXTENSION=1')

...
```

CHAPTER

SEVEN

BUILD WITH COGENO

Code generation has to be invoked as part of the build process of a project.

7.1 Build with CMake and Cogeno

Projects that use CMake to manage building the project can add the cogeno.cmake script (`cogeno.cmake`).

By this a file that contains inline code generation can be added to the project using the `cogeno_sources` command in the respective `CMakeList.txt` file.

```
cogeno_sources(target [EXTERN] [DELETE_SOURCE] [source_file..]
               [INCLUDES include_file...] [TXTFILES text_file...]
               [SOURCE_DIR dir] [INCLUDE_DIR dir] [TXTFILE_DIR dir]
               [COGENO_DEFINES defines..] [DEPENDS target.. file.. dir..])
```

- **EXTERN** `EXTERN` has to be given in case the target was not created in the same `CMakeList.txt` file as the `cogeno_sources` command is issued.
- **DELETE_SOURCE** If `DELETE_SOURCE` is given the generator code is removed from the output file.
- **SOURCE_DIR** `SOURCE_DIR` specifies the directory to write the generated source file(s) to. A relative path is relative to `${CMAKE_CURRENT_BINARY_DIR})`. It defaults to `${CMAKE_CURRENT_BINARY_DIR}` or to the Cogeno `SOURCE_DIR` property if given. Do not set `SOURCE_DIR` in case the generated file is a compilable source file (eg. *.c, *.cpp), this will trigger CMake issue #14633.
- **INCLUDE_DIR** `INCLUDE_DIR` specifies the directory to write the generated include file(s) to. A relative path is relative to `${CMAKE_CURRENT_BINARY_DIR})`. It defaults to

`${CMAKE_CURRENT_BINARY_DIR}` or to the Cogeno `INCLUDE_DIR` property if given.

- **TXTFILE_DIR** `TXTFILE_DIR` specifies the directory to write the generated text file(s) to. A relative path is relative to `${CMAKE_CURRENT_BINARY_DIR}`). It defaults to `${CMAKE_CURRENT_BINARY_DIR}` or to the Cogeno `TXTFILE_DIR` property if given.
- **INCLUDES** `INCLUDES` marks files to be handled as include files (vs. source files).
- **TXTFILES** `TXTFILES` marks files to be handled as text files (vs. source files). Text files are added as a dependency to the target.
- **COGENO_DEFINES** The arguments given by the `COGENO_DEFINES` keyword have to be of the form `define_name=define_value`. The arguments become globals in the python snippets and can be accessed by `define_name`.
- **DEPENDS** Dependencies given by the `DEPENDS` keyword are added to the dependencies of the generated file. Adding a dependency to a directory adds all files in that directory as a dependency.

7.1.1 Use Cogeno modules with CMake

The options provided to the modules are set by Cogeno CMake properties.

Module	Property	Usage
ccode	–	–
cmake	CMAKE_CACHE	Path to CMake cache file
	CMAKE_DEFINES	CMake variable defines
config	:ref:`CONFIG_DB `	Path to config database file
	CONFIG_FILE	Path to config file
	CONFIG_KCONFIG_FILE	Path to Kconfig file
edts	EDTS_ARCH	*Use the Cogeno EDTS module with CMake*
	EDTS_ARCH_FLAVOUR	
	EDTS_DTS	
	EDTS_DTS_ROOT	
	EDTS_DTS_PP_DEFINES	
	EDTS_DTS_PP_INCLUDE_DIRS	
	EDTS_DTS_PP_SOURCES	
	EDTS_BINDINGS_DIRS	
	EDTS_BINDINGS_EXCLUDE	

continues on next page

Table 1 – continued from previous page

Module	Property	Usage
	EDTS_BINDINGS_NO_DEFAULT	
	EDTS_DB	
protobuf	PROTOBUF_DB_DIR	*Use the Cogeno protobuf module with CMake*
	PROTOBUF_SOURCES	
	PROTOBUF_INCLUDE_DIRS	
zephyr	–	–

Use the Cogeno EDTS module with CMake

The cogeno.cmake script searches the list of directories in the EXTENSION_DIRS property and the `edts` or `dts` sub-directories within for device tree root pathes. If the directories exist they are added to the EDTS_DTS_ROOT property.

EXTENSION_DIRS sub-directory	Property/ usage
.	EDTS_DTS_ROOT
`dts`	EDTS_DTS_ROOT
`edts`	EDTS_DTS_ROOT

The EDTS_DTS_ROOT pathes are searched for a set of standard sub-directories. If available these sub-directories are added to the respective Cogeno EDTS properties.

EDTS_DTS_ROOT sub-directory	Property/ usage
`include`	EDTS_DTS_PP_INCLUDE_DIRS
`include` /dt-bindings	EDTS_DTS_PP_INCLUDE_DIRS
`common`	EDTS_DTS_PP_INCLUDE_DIRS
EDTS_ARCH	EDTS_DTS_PP_INCLUDE_DIRS
EDTS_ARCH/EDTS_ARCH_FLAVOUR	EDTS_DTS_PP_INCLUDE_DIRS
`bindings`	EDTS_BINDINGS_DIRS

Use the Cogeno protobuf module with CMake

The cogeno.cmake script searches the list of directories in the EXTENSION_DIRS property for a `proto` sub-directory. If the directory exists it is added to the PROTOBUF_INCLUDE_DIRS property.

The protobuf sources (*.proto) shall be added to the dependencies of the source file that is using the protobuf code database. The protobuf sources may either be given with absolute path or with a path relative to the `CMakeList.txt` directory.

The include directories that contain protobuf files to be imported by the protobuf sources, and that are not listed in the PROTOBUF_INCLUDE_DIRS property, shall also be added to the dependencies. The enclosing directories of the protobuf sources are automatically added.

`CMakeList.txt`

```
cogeno_sources('proto_user.c' DEPENDS proto/myproto.proto ../../general/proto)
```

7.1.2 Cogeno CMake properties

- DEFINES
- MODULES
- TEMPLATES
- EXTENSION_DIRS
- CMAKE_CACHE
- CMAKE_DEFINES
- CONFIG_FILE
- EDTS_ARCH
- EDTS_ARCH_FLAVOUR
- EDTS_DTS
- EDTS_DTS_ROOT
- EDTS_DTS_PP_DEFINES
- EDTS_DTS_PP_INCLUDE_DIRS
- EDTS_DTS_PP_SOURCES
- EDTS_BINDINGS_DIRS
- EDTS_BINDINGS_EXCLUDE
- EDTS_BINDINGS_NO_DEFAULT
- EDTS_DB
- PROTOBUF_DB_DIR
- PROTOBUF_SOURCES
- PROTOBUF_INCLUDE_DIRS

7.2 Build with Zephyr and Cogeno

Zephyr uses the CMake build system with custom extensions. *Build with CMake and Cogeno* also applies to Zephyr builds.

To use Cogeno you have to include cogeno.cmake in your project's `CMakeList.txt` file.

The easiest way to get cogeno.cmake and all of Cogeno with Zephyr is to install Cogeno as a Zephyr module.

`west.yml`

```
manifest:

  remotes:

    - name: gitlab_b0661
      url-base: https://gitlab.com/b0661

  projects:

    - name: cogeno
      remote: gitlab_b0661
      revision: master
      path: cogeno
```

Assure that on build the Cogeno module is processed before the modules that use it. If the sequence created by *west –list* does not suite your needs explicitly set *ZEPHYR_MODULES* in your project.

`CMakeList.txt`

```
set(ZEPHYR_MODULES
    ${CMAKE_CURRENT_SOURCE_DIR}/../cogeno
    ...
)

include($ENV{ZEPHYR_BASE}/cmake/app/boilerplate.cmake NO_POLICY_SCOPE)
```

In Zephyr the processing of source files is controlled by the CMake extension functions `zephyr_..._sources_cogeno..(..)` or `zephyr_..includes_cogeno..(..)`. During build the source files are processed by Cogeno and the generated source files are written to the CMake binary directory. The generated source files are added to the Zephyr sources.

A file that contains inline code generation has to be added to the project by one of the following commands in a `CMakeList.txt` file:

zephyr_sources_cogeno (*file [COGENO_DEFINES defines..] [DEPENDS target.. file..]*)

zephyr_sources_cogeno_ifdef (*ifguard file [COGENO_DEFINES defines..] [DEPENDS target.. file..]*)

zephyr_library_sources_cogeno (*file [COGENO_DEFINES defines..] [DEPENDS target.. file..]*)

zephyr_library_sources_cogeno_ifdef (*ifguard file [COGENO_DEFINES defines..] [DEPENDS target.. file..]*)

zephyr_library_includes_cogeno (*file.. [COGENO_DEFINES defines..] [DEPENDS target.. file..]*)

zephyr_library_includes_cogeno_ifdef (*ifguard file.. [COGENO_DEFINES defines..] [DEPENDS target.. file..]*)

7.2.1 Use Cogeno modules with Zephyr

Use the Cogeno EDTS module with Zephyr

The project device tree file is processed twice by gen_defines.py of Zephyr and cogeno.edts() of Cogeno.

Usually the both should work on the same properties - but in some rare cases gen_defines.py of Zephyr may not understand certain properties because

- the property is not foreseen for extraction
- or the include directory for an include file for *device tree preprocessing* provided to the EDTS module is not provided to gen_defines.py (`edts` vs. `dts`)
- or a binding file provided to the EDTS module is different from the one provided to gen_defines.py

In these cases you can use defines to exclude parts of the device tree file from processing by gen_defines.py. `COGENO_EDTS` is only defined if *device tree preprocessing* is done on behalf of Cogeno. If you need a more fine grained control you should add a define in the `cogeno/<extension>.py` Cogeno extension module of your project or Zephyr module (see Search for Cogeno extensions and Script Cogeno extension modules):

```
import cogeno
cogeno.option_argv_append('--edts:dts-pp-defines', "MY_DTS_DEFINE=1")
```

In the project device tree file you can control the DTS preprocessing in the usual way:

```
#if defined(MY_DTS_DEFINE)
#include <dt-bindings/my_special_driver/my_special_driver.h>
#endif

&my_special_driver {
#if defined(MY_DTS_DEFINE)
        clock-output-names = MY_SPECIAL_DRIVER_CLOCK_1 MY_SPECIAL_DRIVER_CLOCK_2
#endif
};
```

7.3 Build with ESP-IDF/ MDF and Cogeno

Cogeno can be integrated as a component into ESP32 projects using the ESP-IDF or ESP-MDF framework and the CMake build system. *Build with CMake and Cogeno* also applies to ESP-IDF/ ESP-MDF builds.

To create the Cogeno component make Cogeno a git submodule within the project's components directory.

```
cd <project>/components
git submodule add https://gitlab.com/b0661/cogeno
```

Source files of components may use inline code generation. To generate the inline code the component sources have to be marked using the `cogeno_sources()` command in the respective `CMakeList.txt` file.

`cogeno_sources()` adds the generated source files to `COGENO_COMPONENT_SRCS`. It also adds include directories to `COGENO_COMPONENT_INCLUDE_DIRS`. The `COGENO_COMPONENT_SRCS` and the `COGENO_COMPONENT_INCLUDE_DIRS` have to be registered for the component.

Components should have the Cogeno component given in their `PRIV_REQUIRES` property.

```
if(NOT CMAKE_BUILD_EARLY_EXPANSION)
    cogeno_sources(${COMPONENT_NAME} inline_code_filea.c inline_code_fileb.c ...)
endif()
idf_component_register(
    SRCS ${COGENO_COMPONENT_SRCS} ...
    PRIV_REQUIRES cogeno ...
    INCLUDE_DIRS ${COGENO_COMPONENT_INCLUDE_DIRS} ...)
```

7.3.1 Use Cogeno modules with ESP-IDF and ESP-MDF

Use the Cogeno EDTS module with ESP-IDF and ESP-MDF

To use the Cogeno Extended Device Tree Specification (EDTS) module your project has to provide the top level device tree specification `edts/project.dts`. The `project.dts` file should include device tree source files from the main component and other components provided in their respective `edts` directory.

`edts/project.dts`

```
#include <boards/esp32/esp32.dts>
```

A special component holds most of the generic device tree source files for inclusion:

- edts (see examples/esp_idf/edts)

7.4 cogeno.cmake

```
# Cogeno CMake example for build system integration.
# See examples/cmake/cogeno.cmake
#
# Cogeno properties - leave untouched for default:
# - DELETE_CODE
# - DEFINES
# - MODULES
# - TEMPLATES
# - EXTENSION_DIRS
# - CMAKE_CACHE
# - CMAKE_DEFINES
# - CONFIG_DB
# - CONFIG_FILE
# - CONFIG_INPUTS
# - CONFIG_KCONFIG_FILE
# - CONFIG_KCONFIG_SRCTREE
# - CONFIG_KCONFIG_DEFINES
# - EDTS_ARCH
# - EDTS_ARCH_FLAVOUR
# - EDTS_BINDINGS_DIRS
# - EDTS_BINDINGS_EXCLUDE
# - EDTS_BINDINGS_NO_DEFAULT
# - EDTS_DB
# - EDTS_DTS
# - EDTS_DTS_ROOT
# - EDTS_DTS_PP_DEFINES
# - EDTS_DTS_PP_INCLUDE_DIRS
```

(continues on next page)

(continued from previous page)

```
# - EDTS_DTS_PP_SOURCES
# - PROTOBUF_DB_DIR
# - PROTOBUF_INCLUDE_DIRS
# - PROTOBUF_SOURCES

# protect against multiple inclusions
include_guard(GLOBAL)

# utilities
function(cogeno_unique_target_name_from_filename filename target_name)
    get_filename_component(basename ${filename} NAME)
    string(REPLACE "." "_" x ${basename})
    string(REPLACE "@" "_" x ${x})

    string(MD5 unique_chars ${filename})

    set(${target_name} cogeno_${x}_${unique_chars} PARENT_SCOPE)
endfunction()

##
# @brief Retrieve cogeno property of a project.
#
# @param var
# @param property
function(cogeno_get_property var property)
    get_property(is_set GLOBAL PROPERTY "COGENO_${property}" SET)
    if(${is_set})
        get_property(val GLOBAL PROPERTY "COGENO_${property}")
        set(${var} "${val}" PARENT_SCOPE)
    endif()
endfunction()

##
# @brief Set cogeno property on a project.
#
# @param property
# @param val
# @param APPEND
# @param UNIQUE
function(cogeno_set_property property val)
    set(options APPEND UNIQUE)
    cmake_parse_arguments(_ "${options}" "" "" ${ARGN})

    # Allow for relative path type properties
    cogeno_get_property(path_properties PATH_PROPERTIES)
    if(NOT path_properties)
        # Init standard cogeno path type properties
        list(APPEND path_properties
            MODULES TEMPLATES EXTENSION_DIRS CMAKE_CACHE
            CONFIG_DB CONFIG_FILE CONFIG_INPUTS
            CONFIG_KCONFIG_FILE CONFIG_KCONFIG_SRCTREE
            EDTS_DTS EDTS_DTS_PP_INCLUDE_DIRS EDTS_DTS_PP_SOURCES
            EDTS_BINDINGS_DIRS EDTS_DB PROTOBUF_DB_DIR PROTOBUF_INCLUDE_DIRS
            PROTOBUF_SOURCES)
        set_property(GLOBAL PROPERTY "COGENO_PATH_PROPERTIES" "${path_properties}")
    endif()
    if(${property} IN_LIST path_properties)
```

(continues on next page)

(continued from previous page)

```
        if(NOT IS_ABSOLUTE ${val})
            set(val "${CMAKE_CURRENT_LIST_DIR}/${val}")
        endif()
        get_filename_component(val ${val} REALPATH)
    endif()

    if(__APPEND AND __UNIQUE)
        cogeno_get_property(property_val ${property})
        if(${val} IN_LIST property_val)
            return()
        endif()
    endif()

    if(__APPEND)
        set_property(GLOBAL APPEND PROPERTY "COGENO_${property}" "${val}")
    else()
        set_property(GLOBAL PROPERTY "COGENO_${property}" "${val}")
    endif()

    # Keep track of set cogeno properties
    cogeno_get_property(properties PROPERTIES)
    if(NOT property IN_LIST properties)
        cogeno_set_property(PROPERTIES ${property} APPEND)
    endif()
endfunction()

# Find Cogeno
#
# In case cogeno is installed at a custom location the cogeno INSTALL_PATH property
# may be set before calling FindCogeno().
# ``cogeno_set_property(INSTALL_PATH /install/path)``
#
# Result variables:
# ``Cogeno_FOUND``
# ``Cogeno_EXECUTABLE``
# ``Cogeno_EXECUTABLE_ARGS``
# ``Cogeno_BASE``
function(FindCogeno)
    cogeno_get_property(found FOUND)
    if(NOT DEFINED found)
        # Search for cogeno
        set(found FALSE)

        # We do the simple check - the example file was included - first.
        # If this is not the case we search for cogeno.py later on.
        #
        # If this file
        # - is named cogeno.cmake
        # - and is in examples/cmake/
        # - and there exists a cogeno.py in ../../cogeno
        # we know about the cogeno file.
        get_filename_component(cogeno_cmake_example ${CMAKE_CURRENT_LIST_FILE}␣
↪ABSOLUTE)
        if(cogeno_cmake_example MATCHES ".*/examples/cmake/cogeno\.cmake")
            set(found TRUE)
            get_filename_component(base "${CMAKE_CURRENT_LIST_DIR}/../.." ABSOLUTE)
            set(cogeno_py "${base}/cogeno/cogeno.py")
```

(continues on next page)

(continued from previous page)

```
        endif()

        # Looking for Zephyr specific installation
        if(NOT ${found} AND DEFINED ZEPHYR_BASE AND EXISTS "${ZEPHYR_BASE}")
            # Preferred solution is cogeno as a west managed module
            if(ZEPHYR_MODULES)
                foreach(module ${ZEPHYR_MODULES})
                    set(full_path ${module}/cogeno/cogeno.py)
                    if(EXISTS ${full_path})
                        set(found TRUE)
                        set(base "${module}")
                        set(cogeno_py ${full_path})
                        break()
                    endif()
                endforeach()
            endif()
        endif()

        # Looking for ESP-IDF/MDF specific installation
        if(NOT ${found} AND DEFINED ESP_PLATFORM AND ESP_PLATFORM)
            # Preferred solution is cogeno as a component.
            # Git submodule of cogeno is in {component}/cogeno
            foreach(component_dir ${COMPONENT_DIRS})
                FILE(GLOB components LIST_DIRECTORIES true "${component_dir}")
                foreach(component ${components})
                    set(full_path ${component}/cogeno/cogeno/cogeno.py)
                    if(EXISTS ${full_path})
                        set(found TRUE)
                        set(base "${component}/cogeno" CACHE INTERNAL "cogeno base␣
↪directory")
                        set(cogeno_py ${full_path})
                        break()
                    endif()
                endforeach()
                if(${found})
                    break()
                endif()
            endforeach()
        endif()

        # Looking for custom installation provided by cogeno property
        cogeno_get_property(install_path INSTALL_PATH)
        if(NOT ${found} AND DEFINED install_path)
            # take install path as top level cogeno directory
            set(full_path ${install_path}/cogeno/cogeno.py)
            if(EXISTS ${full_path})
                set(found TRUE)
                set(base "${install_path}")
                set(cogeno_py ${full_path})
            # search sub directories of install path
            else()
                FILE(GLOB modules LIST_DIRECTORIES true "${install_path}/*")
                foreach(module ${modules})
                    set(full_path ${module}/cogeno/cogeno.py)
                    if(EXISTS ${full_path})
                        set(found TRUE)
                        set(base "${module}")
```

(continues on next page)

(continued from previous page)

```
                        set(cogeno_py ${full_path})
                        break()
                    endif()
                endforeach()
            endif()
        endif()

        # Do some further heuristics in case we have not found cogeno
        # Priorities:
        # - 1st priority: cogeno installed side by side to the project that hosts␣
↪this file
        # - 2nd priority: cogeno installed on host
        # - 3rd priority: get cogeno from the git repository

        # cogeno installed side by side to the project that hosts this file
        # We do not know the nesting - just go up two or one level and search
        if(NOT ${found})
            # - 2 levels
            FILE(GLOB modules LIST_DIRECTORIES true "${CMAKE_SOURCE_DIR}/../../*")
            foreach(module ${modules})
                set(full_path ${module}/cogeno/cogeno.py)
                if(EXISTS ${full_path})
                    set(found TRUE)
                    set(base "${module}")
                    set(cogeno_py ${full_path})
                    break()
                endif()
            endforeach()
        endif()
        if(NOT ${found})
            # - 1 level
            FILE(GLOB modules LIST_DIRECTORIES true "${CMAKE_SOURCE_DIR}/../*")
            foreach(module ${modules})
                set(full_path ${module}/cogeno/cogeno.py)
                if(EXISTS ${full_path})
                    set(found TRUE)
                    set(base "${module}")
                    set(cogeno_py ${full_path})
                    break()
                endif()
            endforeach()
        endif()

        # cogeno installed on host
        if(NOT ${found})
            find_program(executable_found cogeno)

            if(EXISTS "${executable_found}")
                # Ask cogeno itself for base
                execute_process(${executable_found} "--base"
                                OUTPUT_VARIABLE base RESULT_VARIABLE ret)
                if(NOT ${ret} EQUAL 0)
                    message(FATAL_ERROR ${ret})
                endif()
                set(executable "${executable_found}")
                set(found TRUE)
            endif()
```

(continues on next page)

(continued from previous page)

```
        endif()

        # Get cogeno from the git repository
        # Only install if a install path is given as a cogeno property.
        if(NOT ${found} AND DEFINED install_path)
            find_package(Git)
            if(NOT GIT_FOUND)
                message(FATAL_ERROR "git not found!")
            endif()
            if(NOT EXISTS "${install_path}")
                message(FATAL_ERROR "install path '${install_path}' does not exist!")
            endif()
            execute_process(
                COMMAND             ${GIT_EXECUTABLE} clone https://gitlab.com/b0661/
↪cogeno.git --recursive
                WORKING_DIRECTORY   "${install_path}"
                OUTPUT_VARIABLE     git_output)
            message(STATUS "${git_output}")
            set(module "${install_path}/cogeno")
            if (EXISTS ${module})
                execute_process(
                    COMMAND             ${GIT_EXECUTABLE} checkout master
                    WORKING_DIRECTORY   "${module}"
                    OUTPUT_VARIABLE     git_output)
                message(STATUS "${git_output}")
                set(full_path ${module}/cogeno/cogeno.py)
                if(EXISTS ${full_path})
                    set(found TRUE)
                    set(base "${module}")
                    set(cogeno_py ${full_path})
                endif()
            endif()
        endif()

        cogeno_set_property(FOUND ${found})
        if(${found})
            # We may need a Python interpreter for cogeno
            if(DEFINED executable)
                # We do not need the Python 3 interpreter
                set(executable_args)
            else()
                if(${CMAKE_VERSION} VERSION_LESS "3.12")
                    set(Python_ADDITIONAL_VERSIONS 3.7 3.6 3.5)
                    find_package(PythonInterp)

                    set(Python3_Interpreter_FOUND ${PYTHONINTERP_FOUND})
                    set(Python3_EXECUTABLE ${PYTHON_EXECUTABLE})
                    set(Python3_VERSION ${PYTHON_VERSION_STRING})
                else()
                    # CMake >= 3.12
                    find_package(Python3 COMPONENTS Interpreter)
                endif()

                if(NOT ${Python3_Interpreter_FOUND})
                    message(FATAL_ERROR "Python 3 not found")
                endif()
```

(continues on next page)

(continued from previous page)

```
                set(executable "${Python3_EXECUTABLE}")
                set(executable_args "${cogeno_py}")
            endif()
            cogeno_set_property(EXECUTABLE ${executable})
            cogeno_set_property(EXECUTABLE_ARGS ${executable_args})
            cogeno_set_property(BASE ${base})
        endif()
    endif()

    set(Cogeno_FOUND ${found} CACHE INTERNAL "cogeno found" PARENT_SCOPE)
    if(${found})
        cogeno_get_property(executable EXECUTABLE)
        set(Cogeno_EXECUTABLE ${executable}
            CACHE INTERNAL "cogeno executable"  PARENT_SCOPE)
        cogeno_get_property(executable_args EXECUTABLE_ARGS)
        set(Cogeno_EXECUTABLE_ARGS ${executable_args}
            CACHE INTERNAL "cogeno executable arguments" PARENT_SCOPE)
        cogeno_get_property(base BASE)
        set(Cogeno_BASE ${base}
            CACHE INTERNAL "cogeno base directory" PARENT_SCOPE)
    endif()
endfunction()

function(cogeno_init_properties_zephyr)
    # Set directories to be searched for cogeno extensions
    foreach(extension_dir ${DTS_ROOT} ${ZEPHYR_MODULES})
        get_filename_component(extension_dir "${extension_dir}" REALPATH)
        cogeno_set_property(EXTENSION_DIRS ${extension_dir} APPEND UNIQUE)
    endforeach()

    if(EXISTS "${APPLICATION_SOURCE_DIR}/templates")
        cogeno_set_property(TEMPLATES "${APPLICATION_SOURCE_DIR}/templates" APPEND␣
↪UNIQUE)
        cogeno_set_property(MODULES "${APPLICATION_SOURCE_DIR}/templates" APPEND␣
↪UNIQUE)
    endif()
    if(EXISTS "${PROJECT_SOURCE_DIR}/templates")
        cogeno_set_property(TEMPLATES "${PROJECT_SOURCE_DIR}/templates" APPEND UNIQUE)
        cogeno_set_property(MODULES "${PROJECT_SOURCE_DIR}/templates" APPEND UNIQUE)
    endif()

    # DOTCONFIG and merge_config_files is created by Zephyr - use it to get the␣
↪config fragements
    set(config_file "${DOTCONFIG}")
    cogeno_set_property(CONFIG_FILE "${config_file}")
    foreach(config_fragment ${merge_config_files})
        if(${config_fragment} STREQUAL ${config_file})
            continue()
        endif()
        cogeno_set_property(CONFIG_INPUTS "${config_fragment}" APPEND UNIQUE)
    endforeach()
    cogeno_set_property(CONFIG_KCONFIG_FILE "${KCONFIG_ROOT}")
    cogeno_set_property(CONFIG_KCONFIG_SRCTREE "${ZEPHYR_BASE}")
    cogeno_set_property(CONFIG_KCONFIG_DEFINES "ZEPHYR_BASE=${ZEPHYR_BASE}" APPEND␣
↪UNIQUE)
    cogeno_set_property(CONFIG_KCONFIG_DEFINES "KERNELVERSION=${KERNELVERSION}"␣
↪APPEND UNIQUE)
```

(continues on next page)

(continued from previous page)

```
    cogeno_set_property(CONFIG_KCONFIG_DEFINES "KCONFIG_CONFIG=${config_file}" APPEND␣
↪UNIQUE)
    cogeno_set_property(CONFIG_KCONFIG_DEFINES "PYTHON_EXECUTABLE=${PYTHON_EXECUTABLE}
↪" APPEND UNIQUE)
    cogeno_set_property(CONFIG_KCONFIG_DEFINES "ARCH=${ARCH}" APPEND UNIQUE)
    cogeno_set_property(CONFIG_KCONFIG_DEFINES "BOARD_DIR=${BOARD_DIR}" APPEND UNIQUE)
    cogeno_set_property(CONFIG_KCONFIG_DEFINES "SHIELD_AS_LIST=${SHIELD_AS_LIST}"␣
↪APPEND UNIQUE)
    cogeno_set_property(CONFIG_KCONFIG_DEFINES "KCONFIG_BINARY_DIR=${KCONFIG_BINARY_
↪DIR}" APPEND UNIQUE)
    cogeno_set_property(CONFIG_KCONFIG_DEFINES "ARCH_DIR=${ARCH_DIR}" APPEND UNIQUE)
    cogeno_set_property(CONFIG_KCONFIG_DEFINES "TOOLCHAIN_KCONFIG_DIR=${TOOLCHAIN_
↪KCONFIG_DIR}" APPEND UNIQUE)
    cogeno_set_property(CONFIG_KCONFIG_DEFINES "EDT_PICKLE=${EDT_PICKLE}" APPEND␣
↪UNIQUE)

    cogeno_set_property(CMAKE_DEFINES "APPLICATION_SOURCE_DIR" APPEND UNIQUE)
    cogeno_set_property(CMAKE_DEFINES "APPLICATION_BINARY_DIR" APPEND UNIQUE)

    cogeno_set_property(EDTS_ARCH ${ARCH})

    # Set EDTS_DTS_PP_SOURCES property
    if(NOT DEFINED DTS_SOURCE)
        message(STATUS "Cogeno can't create EDTS - Zephyr 'DTS_SOURCE' not provided")
    elseif(NOT EXISTS ${DTS_SOURCE})
        message(STATUS "Cogeno can't create EDTS - '${DTS_SOURCE}' not found")
    else()
        cogeno_set_property(EDTS_DTS_PP_SOURCES "${DTS_SOURCE}")
        if(DTC_OVERLAY_FILE)
            # DTC_OVERLAY_FILE is a space-separated list
            foreach(overlay_file ${DTC_OVERLAY_FILE})
                if(EXISTS ${overlay_file})
                    cogeno_set_property(EDTS_DTS_PP_SOURCES "${overlay_file}" APPEND␣
↪UNIQUE)
                else()
                    message(STATUS "Cogeno can't create EDTS - '${overlay_file}' not␣
↪found")
                endif()
            endforeach()
        endif()
    endif()

    # Set EDTS_DTS_PP_DEFINES
    cogeno_set_property(EDTS_DTS_PP_DEFINES "__DTC__" APPEND UNIQUE)

    # Set EDTS_DTS_ROOT property
    if(NOT DEFINED DTS_ROOT)
        message(STATUS "Cogeno may not create EDTS - Zephyr 'DTS_ROOT' not provided")
    endif()

    # Set EDTS_BINDINGS_DIRS property
    if(NOT DEFINED DTS_ROOT_BINDINGS)
        message(STATUS "Cogeno may not create EDTS - Zephyr 'DTS_ROOT_BINDINGS' not␣
↪provided")
    else()
        string(REPLACE "?" ";" root_bindings "${DTS_ROOT_BINDINGS}")
        list(APPEND edts_bindings_dirs ${root_bindings})
```

(continues on next page)

(continued from previous page)

```
        list(REMOVE_DUPLICATES edts_bindings_dirs)
        foreach(binding_dir ${edts_bindings_dirs})
            if(EXISTS ${binding_dir})
                cogeno_set_property(EDTS_BINDINGS_DIRS ${binding_dir} APPEND UNIQUE)
            endif()
        endforeach()
    endif()

    # Set EDTS_BINDINGS_EXCLUDE property
    # Exclude generic bindings that are provided by cogeno
    foreach(binding_exclude
        "${ZEPHYR_BASE}/dts/bindings/iio/adc/adc-controller.yaml"
        "${ZEPHYR_BASE}/dts/bindings/base/base.yaml"
        "${ZEPHYR_BASE}/dts/bindings/can/can-controller.yaml"
        "${ZEPHYR_BASE}/dts/bindings/can/can-device.yaml"
        "${ZEPHYR_BASE}/dts/bindings/clock/clock-controller.yaml"
        "${ZEPHYR_BASE}/dts/bindings/clock/fixed-clock.yaml"
        "${ZEPHYR_BASE}/dts/bindings/cpu/cpu.yaml"
        "${ZEPHYR_BASE}/dts/bindings/dma/dma-controller.yaml"
        "${ZEPHYR_BASE}/dts/bindings/espi/espi-controller.yaml"
        "${ZEPHYR_BASE}/dts/bindings/ethernet/ethernet.yaml"
        "${ZEPHYR_BASE}/dts/bindings/flash_controller/flash-controller.yaml"
        "${ZEPHYR_BASE}/dts/bindings/gpio/gpio-controller.yaml"
        "${ZEPHYR_BASE}/dts/bindings/gpio/gpio-keys.yaml"
        "${ZEPHYR_BASE}/dts/bindings/gpio/gpio-leds.yaml"
        "${ZEPHYR_BASE}/dts/bindings/gpio/gpio-nexus.yaml"
        "${ZEPHYR_BASE}/dts/bindings/i2c/i2c-controller.yaml"
        "${ZEPHYR_BASE}/dts/bindings/i2c/i2c-device.yaml"
        "${ZEPHYR_BASE}/dts/bindings/i2s/i2s-controller.yaml"
        "${ZEPHYR_BASE}/dts/bindings/i2s/i2s-device.yaml"
        "${ZEPHYR_BASE}/dts/bindings/interrupt-controller/interrupt-controller.yaml"
        "${ZEPHYR_BASE}/dts/bindings/interrupt-controller/shared-irq.yaml"
        "${ZEPHYR_BASE}/dts/bindings/kscan/kscan.yaml"
        "${ZEPHYR_BASE}/dts/bindings/led/pwm-leds.yaml"
        "${ZEPHYR_BASE}/dts/bindings/mmc/mmc-spi-slot.yaml"
        "${ZEPHYR_BASE}/dts/bindings/mmc/mmc.yaml"
        "${ZEPHYR_BASE}/dts/bindings/mtd/eeprom-base.yaml"
        "${ZEPHYR_BASE}/dts/bindings/mtd/eeprom-spi-i2c.yaml"
        "${ZEPHYR_BASE}/dts/bindings/mtd/partition.yaml"
        "${ZEPHYR_BASE}/dts/bindings/mtd/soc-nv-flash.yaml"
        "${ZEPHYR_BASE}/dts/bindings/phy/phy-controller.yaml"
        "${ZEPHYR_BASE}/dts/bindings/ps2/ps2.yaml"
        "${ZEPHYR_BASE}/dts/bindings/pwm/pwm-controller.yaml"
        "${ZEPHYR_BASE}/dts/bindings/rtc/rtc.yaml"
        "${ZEPHYR_BASE}/dts/bindings/serial/uart-controller.yaml"
        "${ZEPHYR_BASE}/dts/bindings/serial/uart-device.yaml"
        "${ZEPHYR_BASE}/dts/bindings/spi/spi-controller.yaml"
        "${ZEPHYR_BASE}/dts/bindings/spi/spi-device.yaml"
        "${ZEPHYR_BASE}/dts/bindings/sram/mmio-sram.yaml"
        "${ZEPHYR_BASE}/dts/bindings/usb/usb-controller.yaml"
        "${ZEPHYR_BASE}/dts/bindings/usb/usb-ep.yaml")
        cogeno_set_property(EDTS_BINDINGS_EXCLUDE ${binding_exclude} APPEND UNIQUE)
    endforeach()

endfunction()

function(cogeno_postprocess_properties_zephyr)
```

(continues on next page)

(continued from previous page)

```
    # Make the bindings include dirs publically available
    cogeno_get_property(edts_dts_pp_include_dirs EDTS_DTS_PP_INCLUDE_DIRS)
    zephyr_include_directories(${edts_dts_pp_include_dirs})
endfunction()

function(cogeno_init_properties_esp_platform)

    # Search for components
    # - extension dirs
    # - cogeno component
    # - main component
    # - edts component ???
    #
    # Take always the last component directory found.
    #
    # Explanation from IDF documentation:
    # This allows, for example, overriding ESP-IDF components with
    # a modified version by copying that component from the ESP-IDF
    # components directory to the project components directory and
    # then modifying it there. If used in this way, the ESP-IDF
    # directory itself can remain untouched.
    idf_build_get_property(build_component_targets __BUILD_COMPONENT_TARGETS)
    foreach(component_target ${build_component_targets})
        __component_get_property(component_dir ${component_target} COMPONENT_DIR)
        __component_get_property(component_name ${component_target} COMPONENT_NAME)
        # Set directories to be searched for cogeno extensions
        cogeno_set_property(EXTENSION_DIRS ${component_dir} APPEND UNIQUE)
        # cogeno component
        set(full_path ${component_dir}/cogeno/cogeno/cogeno.py)
        if(EXISTS ${full_path})
            cogeno_set_property(COMPONENT_COGENO "${component}")
        endif()
    endforeach()
    # main component
    if(EXISTS "${PROJECT_SOURCE_DIR}/main")
        cogeno_set_property(COMPONENT_MAIN "${PROJECT_SOURCE_DIR}/main")
    endif()

    cogeno_set_property(CONFIG_FILE "${SDKCONFIG}")

    cogeno_set_property(CMAKE_DEFINES "SDKCONFIG" APPEND)
    cogeno_set_property(CMAKE_DEFINES "SDKCONFIG_DEFAULTS" APPEND)
    cogeno_set_property(CMAKE_DEFINES "BUILD_DIR" APPEND)
    cogeno_set_property(CMAKE_DEFINES "COMPONENTS" APPEND)

    cogeno_set_property(CMAKE_DEFINES "COMPONENT_ALIAS" APPEND)
    cogeno_set_property(CMAKE_DEFINES "COMPONENT_DIR" APPEND)
    cogeno_set_property(CMAKE_DEFINES "COMPONENT_LIB" APPEND)
    cogeno_set_property(CMAKE_DEFINES "COMPONENT_NAME" APPEND)
    cogeno_set_property(CMAKE_DEFINES "COMPONENT_TYPE" APPEND)
    cogeno_set_property(CMAKE_DEFINES "EMBED_FILES" APPEND)
    cogeno_set_property(CMAKE_DEFINES "EMBED_TXTFILES" APPEND)
    cogeno_set_property(CMAKE_DEFINES "INCLUDE_DIRS" APPEND)
    cogeno_set_property(CMAKE_DEFINES "KCONFIG" APPEND)
    cogeno_set_property(CMAKE_DEFINES "KCONFIG_PROJBUILD" APPEND)
    cogeno_set_property(CMAKE_DEFINES "LDFRAGMENTS" APPEND)
    cogeno_set_property(CMAKE_DEFINES "PRIV_INCLUDE_DIRS" APPEND)
```

(continues on next page)

(continued from previous page)

```
    cogeno_set_property(CMAKE_DEFINES "PRIV_REQUIRES" APPEND)
    cogeno_set_property(CMAKE_DEFINES "REQUIRED_IDF_TARGETS" APPEND)
    cogeno_set_property(CMAKE_DEFINES "REQUIRES" APPEND)
    cogeno_set_property(CMAKE_DEFINES "SRCS" APPEND)

    # Set EDTS system architecture
    cogeno_set_property(EDTS_ARCH "xtensa")
    cogeno_set_property(EDTS_ARCH_FLAVOUR "espressif")

    # Set EDTS_DTS_PP_SOURCES property
    set(full_path "${PROJECT_SOURCE_DIR}/edts/project.dts")
    if(NOT EXISTS ${full_path})
        message(STATUS "Cogeno can't create EDTS - '${full_path}' not found")
    else()
        cogeno_set_property(EDTS_DTS_PP_SOURCES "${full_path}")
    endif()

endfunction()

function(cogeno_postprocess_properties_esp_platform)
endfunction()

function(cogeno_init_properties_unknown_platform)
endfunction()

function(cogeno_postprocess_properties_unknown_platform)
endfunction()

function(cogeno_init_properties)
    # Generic properties provided by FindCogeno are:
    # - FOUND
    # - EXECUTABLE
    # - EXECUTABLE_ARGS
    # - BASE
    # Properties that are set to default values before platform init:
    # - DELETE_CODE
    # - EXTENSION_DIRS
    # - CONFIG_DB
    # - CMAKE_DEFINES
    # - EDTS_DB
    # - EDTS_DTS
    # - PLATFORM
    # - PROTOBUF_DB_DIR
    # Generic properties provided by platform init:
    # - CONFIG_FILE
    # - CONFIG_INPUTS
    # - CONFIG_KCONFIG_FILE
    # - CONFIG_KCONFIG_SRCTREE
    # - CONFIG_KCONFIG_DEFINES
    # - CMAKE_CACHE
    # - EDTS_ARCH
    # - EDTS_ARCH_FLAVOUR
    # - EDTS_DTS_PP_SOURCES
    # - EDTS_DTS_PP_DEFINES
    # Properties that are set to default values after platform init:
    # - MODULES
    # - TEMPLATES
```

(continues on next page)

(continued from previous page)

```
    # - EDTS_BINDINGS_DIRS
    # - EDTS_DTS_PP_INCLUDE_DIRS
    # - EDTS_DTS_ROOT
    # - PROTOBUF_INCLUDE_DIRS

    # Assure cogeno is available
    # --------------------------
    cogeno_get_property(Cogeno_FOUND FOUND)
    if(NOT DEFINED Cogeno_FOUND)
        FindCogeno()
    endif()
    if(NOT Cogeno_FOUND)
        message(FATAL "Cogeno not found")
    endif()

    # directories to be searched for extended device tree sources, bindings, includes
    set(edts_dirs "edts" "dts")

    # Set pre platform init default properties
    # ----------------------------------------
    cogeno_set_property(DELETE_CODE FALSE)

    cogeno_set_property(EXTENSION_DIRS "${PROJECT_SOURCE_DIR}" APPEND)

    cogeno_set_property(LOG "${CMAKE_BINARY_DIR}${CMAKE_FILES_DIRECTORY}/cogeno.log")
    cogeno_set_property(LOCK "${CMAKE_BINARY_DIR}${CMAKE_FILES_DIRECTORY}/cogeno.lock
↪")

    cogeno_set_property(CMAKE_CACHE "${CMAKE_BINARY_DIR}/CMakeCache.txt")

    cogeno_set_property(CMAKE_DEFINES "PROJECT_NAME" APPEND)
    cogeno_set_property(CMAKE_DEFINES "PROJECT_SOURCE_DIR" APPEND)
    cogeno_set_property(CMAKE_DEFINES "PROJECT_BINARY_DIR" APPEND)
    cogeno_set_property(CMAKE_DEFINES "CMAKE_SOURCE_DIR" APPEND)
    cogeno_set_property(CMAKE_DEFINES "CMAKE_BINARY_DIR" APPEND)
    cogeno_set_property(CMAKE_DEFINES "CMAKE_CURRENT_SOURCE_DIR" APPEND)
    cogeno_set_property(CMAKE_DEFINES "CMAKE_CURRENT_BINARY_DIR" APPEND)
    cogeno_set_property(CMAKE_DEFINES "CMAKE_CURRENT_LIST_DIR" APPEND)
    cogeno_set_property(CMAKE_DEFINES "CMAKE_FILES_DIRECTORY" APPEND)
    cogeno_set_property(CMAKE_DEFINES "CMAKE_PROJECT_NAME" APPEND)
    cogeno_set_property(CMAKE_DEFINES "CMAKE_SYSTEM" APPEND)
    cogeno_set_property(CMAKE_DEFINES "CMAKE_SYSTEM_NAME" APPEND)
    cogeno_set_property(CMAKE_DEFINES "CMAKE_SYSTEM_VERSION" APPEND)
    cogeno_set_property(CMAKE_DEFINES "CMAKE_SYSTEM_PROCESSOR" APPEND)
    cogeno_set_property(CMAKE_DEFINES "CMAKE_C_COMPILER" APPEND)
    cogeno_set_property(CMAKE_DEFINES "CMAKE_CXX_COMPILER" APPEND)
    cogeno_set_property(CMAKE_DEFINES "CMAKE_COMPILER_IS_GNUCC" APPEND)
    cogeno_set_property(CMAKE_DEFINES "CMAKE_COMPILER_IS_GNUCXX" APPEND)

    cogeno_set_property(CONFIG_DB "${CMAKE_BINARY_DIR}/config.json")

    cogeno_set_property(EDTS_DB "${CMAKE_BINARY_DIR}/edts.json")
    cogeno_set_property(EDTS_DTS "${CMAKE_BINARY_DIR}/edts.dts")
    cogeno_set_property(EDTS_DTS_PP_DEFINES "COGENO_EDTS=1")

    cogeno_set_property(PROTOBUF_DB_DIR "${CMAKE_BINARY_DIR}/protobuf")
```

(continues on next page)

(continued from previous page)

```
    # Set platform init properties
    # ----------------------------
    if(DEFINED ZEPHYR_BASE AND EXISTS "${ZEPHYR_BASE}")
        cogeno_set_property(PLATFORM "zephyr")
        cogeno_init_properties_zephyr()
    elseif(DEFINED ESP_PLATFORM AND ESP_PLATFORM)
        cogeno_set_property(PLATFORM "esp")
        cogeno_init_properties_esp_platform()
    else()
        cogeno_set_property(PLATFORM "generic")
        cogeno_init_properties_unknown_platform()
    endif()

    # Set post platform init default properties
    # -----------------------------------------

    # Update EXTENSION_DIRS for cogeno extensions
    cogeno_get_property(extension_dirs EXTENSION_DIRS)
    # Report EXTENSION_DIRS
    foreach(extension_dir ${extension_dirs})
        message(VERBOSE "Cogeno found extension dir '${extension_dir}'")
    endforeach()

    # Set MODULES and TEMPLATES
    foreach(extension_dir ${extension_dirs})
        set(full_path "${extension_dir}/cogeno/modules")
        if(EXISTS ${full_path})
            cogeno_set_property(MODULES "${full_path}" APPEND UNIQUE)
        endif()
        set(full_path "${extension_dir}/cogeno/templates")
        if(EXISTS ${full_path})
            cogeno_set_property(TEMPLATES "${full_path}" APPEND UNIQUE)
        endif()
    endforeach()
    cogeno_get_property(modules MODULES)
    cogeno_get_property(templates TEMPLATES)
    # Report MODULES
    foreach(module ${modules})
        message(VERBOSE "Cogeno found module dir '${module}'")
    endforeach()
    # Report TEMPLATES
    foreach(template ${templates})
        message(VERBOSE "Cogeno found template dir '${template}'")
    endforeach()

    # Set EDTS_DTS_ROOT property
    foreach(extension_dir ${extension_dirs})
        # Use <dir>/include/dt-bindings as DTS root dir
        set(full_path "${extension_dir}/include/dt-bindings")
        if(EXISTS ${full_path})
            cogeno_set_property(EDTS_DTS_ROOT "${extension_dir}" APPEND UNIQUE)
        endif()
        # Search for <dir>/edts and/or <dir>/dts
        foreach(edts_dir ${edts_dirs})
            set(full_path "${extension_dir}/${edts_dir}")
            if(EXISTS ${full_path})
                cogeno_set_property(EDTS_DTS_ROOT "${full_path}" APPEND UNIQUE)
```

(continues on next page)

(continued from previous page)

```
            endif()
        endforeach()
    endforeach()
    cogeno_get_property(edts_dts_root EDTS_DTS_ROOT)
    # Report EDTS_DTS_ROOT
    foreach(dts_root ${edts_dts_root})
        message(VERBOSE "Cogeno found EDT specification root '${dts_root}'")
    endforeach()

    # Set EDTS_BINDINGS_DIRS property
    foreach(dts_root ${edts_dts_root})
        # Bindings given in dts root
        set(full_path "${dts_root}/bindings")
        if(EXISTS ${full_path})
            cogeno_set_property(EDTS_BINDINGS_DIRS "${full_path}" APPEND UNIQUE)
        endif()
        # This is the device tree component - use skeleton.dtsi to detect
        set(full_path "${dts_root}/common/skeleton.dtsi")
        if(EXISTS ${full_path})
            cogeno_set_property(COMPONENT_EDTS "${dts_root}" APPEND UNIQUE)
        endif()
    endforeach()
    cogeno_get_property(edts_bindings_dirs EDTS_BINDINGS_DIRS)
    cogeno_get_property(component_edts COMPONENT_EDTS)
    # Report EDTS_BINDINGS_DIRS
    foreach(edts_bindings_dir ${edts_bindings_dirs})
        message(VERBOSE "Cogeno found EDT bindings dir '${edts_bindings_dir}'")
    endforeach()
    # Report COMPONENT_EDTS
    foreach(component ${component_edts})
        message(VERBOSE "Cogeno found EDT specification component '${component}'")
    endforeach()

    # Get system architecture and create sub dts root architecture standard pathes
    cogeno_get_property(edts_arch EDTS_ARCH)
    if(DEFINED edts_arch)
        set(edts_arch_path "${edts_arch}")
        cogeno_get_property(edts_arch_flavour EDTS_ARCH_FLAVOUR)
        if(DEFINED edts_arch_flavour)
            set(edts_arch_flavour_path "${edts_arch_path}/${edts_arch_flavour}")
        endif()
    endif()

    # Set EDTS_DTS_PP_INCLUDE_DIRS property
    foreach(dts_root ${edts_dts_root})
        cogeno_set_property(EDTS_DTS_PP_INCLUDE_DIRS "${dts_root}" APPEND UNIQUE)
        set(full_path "${dts_root}/include/dt-bindings")
        if(EXISTS ${full_path})
            cogeno_set_property(EDTS_DTS_PP_INCLUDE_DIRS "${dts_root}/include" APPEND␣
↪UNIQUE)
        endif()
        foreach(dts_root_standard_path include common ${edts_arch_path} ${edts_arch_
↪flavour_path})
            set(full_path ${dts_root}/${dts_root_standard_path})
            if(EXISTS ${full_path})
                cogeno_set_property(EDTS_DTS_PP_INCLUDE_DIRS "${full_path}" APPEND␣
↪UNIQUE)
```

(continues on next page)

(continued from previous page)

```
            endif()
        endforeach()
    endforeach()
    cogeno_get_property(edts_dts_pp_include_dirs EDTS_DTS_PP_INCLUDE_DIRS)
    # Report EDTS_DTS_PP_INCLUDE_DIRS
    foreach(edts_bindings_include_dir ${edts_dts_pp_include_dirs})
        message(VERBOSE "Cogeno found EDT bindings include dir '${edts_bindings_
↪include_dir}'")
    endforeach()

    # Set EDTS_DTS_PP_SOURCES property - already set by platform code
    cogeno_get_property(edts_dts_pp_sources EDTS_DTS_PP_SOURCES)
    # Report EDTS_DTS_PP_SOURCES
    foreach(edts_dts_pp_source ${edts_dts_pp_sources})
        message(VERBOSE "Cogeno found DTS source '${edts_dts_pp_source}'")
    endforeach()

    # Set EDTS_DTS_PP_DEFINES
    # - add platform defines
    cogeno_get_property(platform PLATFORM)
    if(DEFINED platform)
        string(TOUPPER "${platform}" platform_uc)
        cogeno_set_property(EDTS_DTS_PP_DEFINES "COGENO_${platform_uc}=1" APPEND␣
↪UNIQUE)
    endif()
    # - add architecture defines
    cogeno_get_property(edts_arch EDTS_ARCH)
    if(DEFINED edts_arch)
        string(TOUPPER "${edts_arch}" edts_arch_uc)
        cogeno_set_property(EDTS_DTS_PP_DEFINES "COGENO_EDTS_ARCH_${edts_arch_uc}=1"␣
↪APPEND UNIQUE)
        cogeno_get_property(edts_arch_flavour EDTS_ARCH_FLAVOUR)
        if(DEFINED edts_arch_flavour)
            string(TOUPPER "${edts_arch_flavour}" edts_arch_flavour_uc)
            cogeno_set_property(EDTS_DTS_PP_DEFINES "COGENO_EDTS_ARCH_${edts_arch_uc}_
↪${edts_arch_flavour_uc}=1" APPEND UNIQUE)
        endif()
    endif()
    cogeno_get_property(edts_dts_pp_defines EDTS_DTS_PP_DEFINES)
    # Report EDTS_DTS_PP_DEFINES
    foreach(edts_dts_pp_define ${edts_dts_pp_defines})
        message(VERBOSE "Cogeno found DTS preprocessor define '${edts_dts_pp_define}'
↪")
    endforeach()

    # Set PROTOBUF_INCLUDE_DIRS property
    foreach(extension_dir ${extension_dirs})
        # Use <dir>/proto as include dir
        set(full_path "${extension_dir}/proto")
        if(EXISTS ${full_path})
            cogeno_set_property(PROTOBUF_INCLUDE_DIRS "${extension_dir}" APPEND␣
↪UNIQUE)
        endif()
    endforeach()
    cogeno_get_property(protobuf_include_dirs PROTOBUF_INCLUDE_DIRS)
    # Report PROTOBUF_INCLUDE_DIRS
    foreach(protobuf_include_dir ${protobuf_include_dirs})
```

(continues on next page)

(continued from previous page)

```
        message(VERBOSE "Cogeno found protobuf include directory '${protobuf_include_
↪dir}'")
    endforeach()

    # Set platform post init properties
    # ---------------------------------
    if(DEFINED ZEPHYR_BASE AND EXISTS "${ZEPHYR_BASE}")
        cogeno_postprocess_properties_zephyr()
    elseif(DEFINED ESP_PLATFORM AND ESP_PLATFORM)
        cogeno_postprocess_properties_esp_platform()
    else()
        cogeno_postprocess_properties_unknown_platform()
    endif()

    cogeno_set_property(PROPERTIES_INITIALIZED TRUE)
endfunction()


function(cogeno_set_options
    args_source_dir
    args_include_dir
    args_txtfile_dir
    args_delete_code
    args_cogeno_defines
    args_depends)

    cogeno_get_property(properties_initialized PROPERTIES_INITIALIZED)
    if(NOT DEFINED properties_initialized)
        # Global cogeno properties not initialized - init now
        cogeno_init_properties()
    endif()

    # cogeno executable
    cogeno_get_property(executable EXECUTABLE)
    set(cogeno_opt_executable ${executable}  PARENT_SCOPE)
    cogeno_get_property(executable_args EXECUTABLE_ARGS)
    set(cogeno_opt_executable_args ${executable_args} PARENT_SCOPE)
    cogeno_get_property(base BASE)
    set(cogeno_opt_base ${base} PARENT_SCOPE)

    # directory to put the sources in
    cogeno_get_property(source_dir SOURCE_DIR)
    if(NOT "${args_source_dir}" STREQUAL "")
        set(cogeno_opt_source_dir "${args_source_dir}")
    elseif(DEFINED source_dir AND source_dir)
        set(cogeno_opt_source_dir "${source_dir}")
    else()
        set(cogeno_opt_source_dir "${CMAKE_CURRENT_BINARY_DIR}")
    endif()
    if(NOT IS_ABSOLUTE ${cogeno_opt_source_dir})
        # relative path - use whole relative path
        set(cogeno_opt_source_dir ${CMAKE_CURRENT_BINARY_DIR}/${cogeno_opt_source_dir}
↪)
    endif()
    set(cogeno_opt_source_dir ${cogeno_opt_source_dir} PARENT_SCOPE)

    # directory to put the includes in
```

(continues on next page)

(continued from previous page)

```
    cogeno_get_property(include_dir INCLUDE_DIR)
    if(NOT "${args_include_dir}" STREQUAL "")
        set(cogeno_opt_include_dir "${args_include_dir}")
    elseif(DEFINED include_dir AND include_dir)
        set(cogeno_opt_include_dir "${include_dir}")
    else()
        set(cogeno_opt_include_dir "${CMAKE_CURRENT_BINARY_DIR}")
    endif()
    if(NOT IS_ABSOLUTE ${cogeno_opt_include_dir})
        # relative path - use whole relative path
        set(cogeno_opt_include_dir ${CMAKE_CURRENT_BINARY_DIR}/${cogeno_opt_include_
↪dir})
    endif()
    set(cogeno_opt_include_dir ${cogeno_opt_include_dir} PARENT_SCOPE)

    # directory to put the text files in
    cogeno_get_property(txtfile_dir TXTFILE_DIR)
    if(NOT "${args_txtfile_dir}" STREQUAL "")
        set(cogeno_opt_txtfile_dir "${args_txtfile_dir}")
    elseif(DEFINED txtfile_dir AND txtfile_dir)
        set(cogeno_opt_txtfile_dir "${txtfile_dir}")
    else()
        set(cogeno_opt_txtfile_dir "${CMAKE_CURRENT_BINARY_DIR}")
    endif()
    if(NOT IS_ABSOLUTE ${cogeno_opt_txtfile_dir})
        # relative path - use whole relative path
        set(cogeno_opt_txtfile_dir ${CMAKE_CURRENT_BINARY_DIR}/${cogeno_opt_txtfile_
↪dir})
    endif()
    set(cogeno_opt_txtfile_dir ${cogeno_opt_txtfile_dir} PARENT_SCOPE)

    # generated files dependencies
    set(cogeno_opt_depends)
    foreach(depend ${args_depends})
        if(TARGET ${depend})
            list(APPEND cogeno_opt_depends ${depend})
        endif()
        # Find path of dependency
        if(NOT IS_ABSOLUTE ${depend})
            # relative path - use whole relative path
            set(depend ${CMAKE_CURRENT_SOURCE_DIR}/${depend})
        endif()
        if(IS_DIRECTORY "${depend}")
            FILE(GLOB depends LIST_DIRECTORIES false "${depend}/*")
            if(depends)
                list(APPEND cogeno_opt_depends ${depends})
            endif()
        else()
            list(APPEND cogeno_opt_depends ${depend})
        endif()
    endforeach()
    set(cogeno_opt_depends ${cogeno_opt_depends} PARENT_SCOPE)

    # --lock
    cogeno_get_property(lock LOCK)
    if(DEFINED lock AND lock)
        set(cogeno_opt_lock "--lock" ${lock})
```

(continues on next page)

(continued from previous page)

```
    else()
        set(cogeno_opt_lock)
    endif()
    set(cogeno_opt_lock ${cogeno_opt_lock} PARENT_SCOPE)

    # --lock
    cogeno_get_property(log LOG)
    if(DEFINED log AND log)
        set(cogeno_opt_log "--log" ${log})
    else()
        set(cogeno_opt_log)
    endif()
    set(cogeno_opt_log ${cogeno_opt_log} PARENT_SCOPE)

    # -x
    cogeno_get_property(delete_code DELETE_CODE)
    if((DEFINED delete_code AND delete_code) OR args_delete_code)
        set(cogeno_opt_delete_code '-x')
    else()
        set(cogeno_opt_delete_code)
    endif()
    set(cogeno_opt_delete_code ${cogeno_opt_delete_code} PARENT_SCOPE)

    # -D
    cogeno_get_property(defines DEFINES)
    if((DEFINED defines AND defines) OR args_cogeno_defines)
        string(REGEX REPLACE "([^;]+)" "-D;\\1"
               cogeno_opt_defines "${defines};${args_cogeno_defines}")
    else()
        set(cogeno_opt_defines)
    endif()
    set(cogeno_opt_defines ${cogeno_opt_defines} PARENT_SCOPE)

    # --extensions
    cogeno_get_property(extensions EXTENSION_DIRS)
    if(DEFINED extensions AND extensions)
        set(cogeno_opt_extensions "--extensions" ${extensions})
    else()
        set(cogeno_opt_extensions)
    endif()
    set(cogeno_opt_extensions ${cogeno_opt_extensions} PARENT_SCOPE)

    # --modules
    cogeno_get_property(modules MODULES)
    if(DEFINED modules AND modules)
        set(cogeno_opt_modules "--modules" ${modules})
    else()
        set(cogeno_opt_modules)
    endif()
    set(cogeno_opt_modules ${cogeno_opt_modules} PARENT_SCOPE)

    # --templates
    cogeno_get_property(templates TEMPLATES)
    if(DEFINED templates AND templates)
        set(cogeno_opt_templates "--templates" ${templates})
    else()
        set(cogeno_opt_templates)
```

(continues on next page)

(continued from previous page)

```
    endif()
    set(cogeno_opt_templates ${cogeno_opt_templates} PARENT_SCOPE)

    # --cmake:define
    cogeno_get_property(cmake_defines CMAKE_DEFINES)
    list(REMOVE_DUPLICATES cmake_defines)
    # Add current values
    foreach(cmake_define ${cmake_defines})
        if(DEFINED ${cmake_define})
            list(APPEND cogeno_opt_cmake_defines
                        --cmake:define "\"${cmake_define}=${${cmake_define}}\"")
        endif()
    endforeach()
    set(cogeno_opt_cmake_defines ${cogeno_opt_cmake_defines} PARENT_SCOPE)

    # --cmake:cache
    cogeno_get_property(cmake_cache CMAKE_CACHE)
    if(DEFINED cmake_cache)
        set(cogeno_opt_cmake_cache "--cmake:cache" "${cmake_cache}")
    else()
        set(cogeno_opt_cmake_cache)
    endif()
    set(cogeno_opt_cmake_cache ${cogeno_opt_cmake_cache} PARENT_SCOPE)

    # --config:db
    cogeno_get_property(config_db CONFIG_DB)
    if(DEFINED config_db)
        set(cogeno_opt_config_db "--config:db" "${config_db}")
    else()
        set(cogeno_opt_config_db)
    endif()
    set(cogeno_opt_config_db ${cogeno_opt_config_db} PARENT_SCOPE)

    # --config:file
    cogeno_get_property(config_file CONFIG_FILE)
    if(DEFINED config_file)
        set(cogeno_opt_config_file "--config:file" "${config_file}")
    else()
        set(cogeno_opt_config_file)
    endif()
    set(cogeno_opt_config_file ${cogeno_opt_config_file} PARENT_SCOPE)

    # --config:inputs
    cogeno_get_property(config_inputs CONFIG_INPUTS)
    if(DEFINED config_inputs)
        set(cogeno_opt_config_inputs "--config:inputs" ${config_inputs})
    else()
        set(cogeno_opt_config_inputs)
    endif()
    set(cogeno_opt_config_inputs ${cogeno_opt_config_inputs} PARENT_SCOPE)

    # --config:kconfig-file
    cogeno_get_property(config_kconfig_file CONFIG_KCONFIG_FILE)
    if(DEFINED config_kconfig_file)
        set(cogeno_opt_config_kconfig_file "--config:kconfig-file" "${config_kconfig_
↪file}")
    else()
```

(continues on next page)

(continued from previous page)

```
        set(cogeno_opt_config_kconfig_file)
    endif()
    set(cogeno_opt_config_kconfig_file ${cogeno_opt_config_kconfig_file} PARENT_SCOPE)

    # --config:kconfig-srctree
    cogeno_get_property(config_kconfig_srctree CONFIG_KCONFIG_SRCTREE)
    if(DEFINED config_kconfig_srctree)
        set(cogeno_opt_config_kconfig_srctree "--config:kconfig-srctree" "${config_
↪kconfig_srctree}")
    else()
        set(cogeno_opt_config_kconfig_srctree)
    endif()
    set(cogeno_opt_config_kconfig_srctree ${cogeno_opt_config_kconfig_srctree} PARENT_
↪SCOPE)

    # --config:kconfig-defines
    cogeno_get_property(config_kconfig_defines CONFIG_KCONFIG_DEFINES)
    if(DEFINED config_kconfig_defines)
        set(cogeno_opt_config_kconfig_defines "--config:kconfig-defines")
        # Add current values
        foreach(config_kconfig_define ${config_kconfig_defines})
            list(APPEND cogeno_opt_config_kconfig_defines "\"${config_kconfig_define}\
↪"")
        endforeach()
    else()
        set(cogeno_opt_config_kconfig_defines)
    endif()
    set(cogeno_opt_config_kconfig_defines ${cogeno_opt_config_kconfig_defines} PARENT_
↪SCOPE)

    # --edts:dts
    cogeno_get_property(edts_dts EDTS_DTS)
    if(DEFINED edts_dts)
        set(cogeno_opt_edts_dts "--edts:dts" "${edts_dts}")
    else()
        set(cogeno_opt_edts_dts)
    endif()
    set(cogeno_opt_edts_dts ${cogeno_opt_edts_dts} PARENT_SCOPE)

    # --edts:dts-pp-sources
    cogeno_get_property(edts_dts_pp_sources EDTS_DTS_PP_SOURCES)
    if(DEFINED edts_dts_pp_sources)
        set(cogeno_opt_edts_dts_pp_sources "--edts:dts-pp-sources" ${edts_dts_pp_
↪sources})
    else()
        set(cogeno_opt_edts_dts_pp_sources)
    endif()
    set(cogeno_opt_edts_dts_pp_sources ${cogeno_opt_edts_dts_pp_sources} PARENT_SCOPE)

    # --edts:dts-pp-defines
    cogeno_get_property(edts_dts_pp_defines EDTS_DTS_PP_DEFINES)
    if(DEFINED edts_dts_pp_defines)
        set(cogeno_opt_edts_dts_pp_defines "--edts:dts-pp-defines")
        # Add current values
        foreach(edts_dts_pp_define ${edts_dts_pp_defines})
            list(APPEND cogeno_opt_edts_dts_pp_defines "\"${edts_dts_pp_define}\"")
        endforeach()
```

(continues on next page)

(continued from previous page)

```
    else()
        set(cogeno_opt_edts_dts_pp_defines)
    endif()
    set(cogeno_opt_edts_dts_pp_defines ${cogeno_opt_edts_dts_pp_defines} PARENT_SCOPE)

    # --edts:dts-pp-include-dirs
    cogeno_get_property(edts_dts_pp_include_dirs EDTS_DTS_PP_INCLUDE_DIRS)
    if(DEFINED edts_dts_pp_include_dirs)
        set(cogeno_opt_edts_dts_pp_include_dirs "--edts:dts-pp-include-dirs" ${edts_
↪dts_pp_include_dirs})
    else()
        set(cogeno_opt_edts_dts_pp_include_dirs)
    endif()
    set(cogeno_opt_edts_dts_pp_include_dirs ${cogeno_opt_edts_dts_pp_include_dirs}␣
↪PARENT_SCOPE)

    # --edts:bindings-dirs
    cogeno_get_property(edts_bindings_dirs EDTS_BINDINGS_DIRS)
    if(DEFINED edts_bindings_dirs)
        set(cogeno_opt_edts_bindings_dirs "--edts:bindings-dirs" ${edts_bindings_dirs}
↪)
    else()
        set(cogeno_opt_edts_bindings_dirs)
    endif()
    set(cogeno_opt_edts_bindings_dirs ${cogeno_opt_edts_bindings_dirs} PARENT_SCOPE)

    # --edts:bindings-exclude
    cogeno_get_property(edts_bindings_exclude EDTS_BINDINGS_EXCLUDE)
    if(DEFINED edts_bindings_exclude)
        set(cogeno_opt_edts_bindings_exclude "--edts:bindings-exclude" ${edts_
↪bindings_exclude})
    else()
        set(cogeno_opt_edts_bindings_exclude)
    endif()
    set(cogeno_opt_edts_bindings_exclude ${cogeno_opt_edts_bindings_exclude} PARENT_
↪SCOPE)

    # --edts:bindings-no-default
    cogeno_get_property(edts_bindings_no_default EDTS_BINDINGS_NO_DEFAULT)
    if(DEFINED edts_bindings_no_default)
        set(cogeno_opt_edts_bindings_no_default "--edts:bindings-no-default" ${edts_
↪bindings_no_default})
    else()
        set(cogeno_opt_edts_bindings_no_default)
    endif()
    set(cogeno_opt_edts_bindings_no_default ${cogeno_opt_edts_bindings_no_default}␣
↪PARENT_SCOPE)

    # --edts:db
    cogeno_get_property(edts_db EDTS_DB)
    if(DEFINED edts_db)
        set(cogeno_opt_edts_db "--edts:db" "${edts_db}")
    else()
        set(cogeno_opt_edts_db)
    endif()
    set(cogeno_opt_edts_db ${cogeno_opt_edts_db} PARENT_SCOPE)
```

(continues on next page)

(continued from previous page)

```
    # --protobuf:db-dir
    cogeno_get_property(protobuf_db_dir PROTOBUF_DB_DIR)
    if(DEFINED protobuf_db_dir)
        set(cogeno_opt_protobuf_db_dir "--protobuf:db-dir" "${protobuf_db_dir}")
    else()
        set(cogeno_opt_protobuf_db_dir)
    endif()
    set(cogeno_opt_protobuf_db_dir ${cogeno_opt_protobuf_db_dir} PARENT_SCOPE)

    # Search for protobuf dependencies
    set(args_protobuf_sources)
    set(args_protobuf_include_dirs)
    foreach(depend ${args_depends})
        if(TARGET ${depend})
            continue()
        endif()
        # Find path of dependency
        if(NOT IS_ABSOLUTE ${depend})
            # relative path - use whole relative path
            set(depend ${CMAKE_CURRENT_SOURCE_DIR}/${depend})
        endif()
        if(IS_DIRECTORY "${depend}")
            FILE(GLOB protos LIST_DIRECTORIES false "${depend}/*.proto")
            if(protos)
                list(APPEND args_protobuf_include_dirs ${depend})
            endif()
        else()
            get_filename_component(depend_ext ${depend} EXT)
            if("${depend_ext}" STREQUAL ".proto")
                list(APPEND args_protobuf_sources ${depend})
                get_filename_component(depend_dir ${depend} DIRECTORY)
                list(APPEND args_protobuf_include_dirs ${depend_dir})
            endif()
        endif()
    endforeach()

    # --protobuf:include-dirs
    cogeno_get_property(protobuf_include_dirs PROTOBUF_INCLUDE_DIRS)
    if(NOT DEFINED protobuf_include_dirs)
        set(protobuf_include_dirs ${args_protobuf_include_dirs})
    else()
        set(protobuf_include_dirs ${protobuf_include_dirs} ${args_protobuf_include_
↪dirs})
    endif()
    if(protobuf_include_dirs)
        set(cogeno_opt_protobuf_include_dirs "--protobuf:include-dirs" ${protobuf_
↪include_dirs})
    else()
        set(cogeno_opt_protobuf_include_dirs)
    endif()
    set(cogeno_opt_protobuf_include_dirs ${cogeno_opt_protobuf_include_dirs} PARENT_
↪SCOPE)

    # --protobuf:sources
    cogeno_get_property(protobuf_sources PROTOBUF_SOURCES)
    if(NOT DEFINED protobuf_sources)
        set(protobuf_sources ${args_protobuf_sources})
```

(continues on next page)

(continued from previous page)

```
    else()
        set(protobuf_sources ${protobuf_sources} ${args_protobuf_sources})
    endif()
    if(protobuf_sources)
        set(cogeno_opt_protobuf_sources "--protobuf:sources" ${protobuf_sources})
    else()
        set(cogeno_opt_protobuf_sources)
    endif()
    set(cogeno_opt_protobuf_sources ${cogeno_opt_protobuf_sources} PARENT_SCOPE)
endfunction()

# Get all the files that make up cogeno for dependency reasons.
#file(GLOB_RECURSE cogeno_sources LIST_DIRECTORIES false
#      ${COGENO_BASE}/cogeno/*.py
#      ${COGENO_BASE}/cogeno/*.yaml
#      ${COGENO_BASE}/cogeno/*.c
#      ${COGENO_BASE}/cogeno/*.jinja2)


function(cogeno_sources
    target          # The CMake target that depends on the generated file
    )
    # Prepare arguments
    set(options EXTERN DELETE_CODE)
    set(oneValueArgs SOURCE_DIR INCLUDE_DIR TXTFILE_DIR)
    set(multiValueArgs INCLUDES TXTFILES COGENO_DEFINES DEPENDS)
    cmake_parse_arguments(SOURCES "${options}" "${oneValueArgs}"
                          "${multiValueArgs}" ${ARGN})

    # Prepare all options
    cogeno_set_options("${SOURCES_SOURCE_DIR}" "${SOURCES_INCLUDE_DIR}" "${SOURCES_
↪TXTFILE_DIR}"
                       "${SOURCES_DELETE_CODE}" "${SOURCES_COGENO_DEFINES}" "$
↪{SOURCES_DEPENDS}")

    message(STATUS "Cogeno will generate for target ${target}")
    set(include_file FALSE)
    set(text_file FALSE)
    foreach(arg ${SOURCES_UNPARSED_ARGUMENTS} "!includes!" ${SOURCES_INCLUDES} "!
↪txtfiles!" ${SOURCES_TXTFILES})
        if("${arg}" STREQUAL "!includes!")
            # We are now processing include files
            set(text_file FALSE)
            set(include_file TRUE)
            continue()
        endif()
        if("${arg}" STREQUAL "!txtfiles!")
            # We are now processing text files
            set(text_file TRUE)
            set(include_file FALSE)
            continue()
        endif()
        get_filename_component(generated_file_name ${arg} NAME)

        # Find path for generated file
        if(${include_file})
            # This is an include file - we got an output directory for include files
```

(continues on next page)

(continued from previous page)

```
            # -> put into include directory
            set(generated_file ${cogeno_opt_include_dir}/${generated_file_name})
            set(generated_dir ${cogeno_opt_include_dir})
        elseif(${text_file})
            # This is a text file - we got an output directory for text files
            # -> put into txtfiles directory
            set(generated_file ${cogeno_opt_txtfile_dir}/${generated_file_name})
            set(generated_dir ${cogeno_opt_txtfile_dir})
        else()
            # This is a source file - we got an output directory for source files
            # -> put into source directory but take care that
            #    compilable source files must be generated to the current binary␣
↪directory.
            #    Otherwise this would trigger CMake issue #14633:
            #    https://gitlab.kitware.com/cmake/cmake/issues/14633
            get_filename_component(generated_ext ${generated_file_name} EXT)
            if(("${generated_ext}" STREQUAL ".c"
                OR "${generated_ext}" STREQUAL ".cpp")
               AND NOT "${cogeno_opt_source_dir}" STREQUAL "${CMAKE_CURRENT_BINARY_
↪DIR}")
                message(WARNING "Compilable source ${arg} generated to non CMAKE_
↪CURRENT_BINARY_DIR (${cogeno_opt_source_dir})")
            endif()
            set(generated_file ${cogeno_opt_source_dir}/${generated_file_name})
            set(generated_dir ${cogeno_opt_source_dir})
        endif()

        # Find path of template file
        if(IS_ABSOLUTE ${arg})
            set(template_file ${arg})
        else()
            # relative path - use whole relative path
            set(template_file ${CMAKE_CURRENT_SOURCE_DIR}/${arg})
        endif()
        get_filename_component(template_dir ${template_file} DIRECTORY)

        if(IS_DIRECTORY ${template_file})
            message(FATAL_ERROR "cogeno_sources() was called on a directory")
        endif()

        # Remove common template extensions from generated file name
        get_filename_component(generated_ext ${generated_file} EXT)
        get_filename_component(generated_name_we ${generated_file} NAME_WE)
        foreach(gen_ext ".in" ".cogeno" ".py" ".jinja")
            foreach(ext ".h" ".hpp" ".c" ".cpp" ".html" ".txt" ".rst" ".md")
                if("${generated_ext}" STREQUAL "${gen_ext}${ext}")
                    set(generated_file "${generated_dir}/${generated_name_we}${ext}")
                    set(generated_ext ${ext})
                    break()
                endif()
            endforeach()
        endforeach()

        # Generate file from template
        message(STATUS " from '${template_file}'")
        message(STATUS " to   '${generated_file}'")
        add_custom_command(
```

(continues on next page)

(continued from previous page)

```
            COMMENT "cogeno ${generated_file}"
            OUTPUT ${generated_file}
            MAIN_DEPENDENCY ${template_file}
            DEPENDS
            "${cogeno_opt_base}/cogeno/cogeno.py"
            ${cogeno_opt_depends}
            COMMAND
            ${cogeno_opt_executable}
            ${cogeno_opt_executable_args}
            ${cogeno_opt_defines}
            ${cogeno_opt_delete_code}
            ${cogeno_opt_extensions}
            ${cogeno_opt_cmake_defines}
            ${cogeno_opt_cmake_cache}
            ${cogeno_opt_config_db}
            ${cogeno_opt_config_file}
            ${cogeno_opt_config_inputs}
            ${cogeno_opt_config_kconfig_file}
            ${cogeno_opt_config_kconfig_srctree}
            ${cogeno_opt_config_kconfig_defines}
            ${cogeno_opt_edts_bindings_dirs}
            ${cogeno_opt_edts_bindings_exclude}
            ${cogeno_opt_edts_bindings_no_default}
            ${cogeno_opt_edts_db}
            ${cogeno_opt_edts_dts}
            ${cogeno_opt_edts_dts_pp_defines}
            ${cogeno_opt_edts_dts_pp_sources}
            ${cogeno_opt_edts_dts_pp_include_dirs}
            ${cogeno_opt_protobuf_db_dir}
            ${cogeno_opt_protobuf_include_dirs}
            ${cogeno_opt_protobuf_sources}
            ${cogeno_opt_modules}
            ${cogeno_opt_templates}
            ${cogeno_opt_log}
            ${cogeno_opt_lock}
            --input "${template_file}"
            --output "${generated_file}"
            WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
        )

        set_source_files_properties(${generated_file} PROPERTIES GENERATED 1)

        # 1) Add generated files to target component.
        # 2) Add template directory to include path to allow includes with
        #    relative path in generated file to work
        # 3) Add directory of generated file to include path to allow includes
        #    of generated header file with relative path.
        #
        # Standard way to add to target is using CMake target_sources() and
        # target_include_directories().
        #
        # Some platforms need a different strategy to achieve the above 1,2,3.
        if(DEFINED ESP_PLATFORM AND ESP_PLATFORM AND "${target}" STREQUAL "$
↪{COMPONENT_NAME}")
            # ESP platforms use idf_component_register
            cogeno_get_property(component_name COMPONENT_NAME)
            if(NOT "${component_name}" STREQUAL "${COMPONENT_NAME}")
```

(continues on next page)

(continued from previous page)

```
                # We are working on a new component - assure everything is prepared
                cogeno_set_property(COMPONENT_SRCS "")
                cogeno_set_property(COMPONENT_INCLUDE_DIRS "")
                cogeno_set_property(COMPONENT_NAME "${COMPONENT_NAME}")
            endif()
            if(${include_file})
                cogeno_set_property(COMPONENT_INCLUDE_DIRS ${generated_dir} APPEND)
                # We add include file to SRCS to get a dependency
                cogeno_set_property(COMPONENT_SRCS ${generated_file} APPEND)
            else()
                cogeno_set_property(COMPONENT_SRCS ${generated_file} APPEND)
            endif()
            cogeno_set_property(COMPONENT_INCLUDE_DIRS ${template_dir} APPEND)
            # Return the values to be used by idf_component_register
            cogeno_get_property(component_srcs COMPONENT_SRCS)
            set(COGENO_COMPONENT_SRCS ${component_srcs} PARENT_SCOPE)
            cogeno_get_property(component_include_dirs COMPONENT_INCLUDE_DIRS)
            list(REMOVE_DUPLICATES component_include_dirs)
            set(COGENO_COMPONENT_INCLUDE_DIRS ${component_include_dirs} PARENT_SCOPE)
        else()
            if(${SOURCES_EXTERN} OR ${text_file})
                # EXTERN:
                # We are adding to a target that was not created in the same␣
↪CMakelLists.txt file.
                # - The generated property is not visible to the target in this case.
                # - The generated file has to be generated first because of that.␣
↪target_sources
                #   expects a file in this case.
                # - A dependency has to be added to assure the generated file is build
                #   before the target
                # TXTFILE:
                # Usually text files are added to custom targets.
                # - A dependency has to be added to assure the generated file is build
                #   before the (custom) target
                cogeno_unique_target_name_from_filename(${generated_file} generated_
↪target_name)
                add_custom_target(${generated_target_name} ALL DEPENDS ${generated_
↪file})
                add_dependencies(${target} ${generated_target_name})
                if(NOT ${include_file} AND NOT ${text_file})
                    # We are adding a source file
                    # Assure the ouput directory exists
                    file(MAKE_DIRECTORY ${generated_dir})
                    # Generate the file - add_custom_target will always force (re-
↪)generation
                    file(TOUCH ${generated_file})
                endif()
            endif()
            if(${include_file})
                # Add output directory for generated file to include path to allow␣
↪includes
                # of generated header file with relative path.
                target_include_directories(${target} SYSTEM BEFORE INTERFACE $
↪{generated_dir})
            elseif(${text_file})
                # Nothing to do
            else()
```

(continues on next page)

(continued from previous page)

```
                target_sources(${target} PRIVATE ${generated_file})
            endif()
            # Add template directory to include path to allow includes with
            # relative path in generated file to work
            if(${include_file})
                # target maybe interface only target
                target_include_directories(${target} PRIVATE INTERFACE ${template_dir}
↪)
            elseif(${text_file})
                # Nothing to do
            else()
                target_include_directories(${target} PRIVATE ${template_dir})
            endif()
        endif()

    endforeach()
endfunction()


# Assure cogeno is available
# --------------------------
FindCogeno()
if(NOT Cogeno_FOUND)
    message(FATAL "Cogeno not found")
endif()


# This is a Zephyr project.
# -------------------------
if(DEFINED ZEPHYR_BASE AND EXISTS "${ZEPHYR_BASE}")
    message(STATUS "Cogeno added to Zephyr")

    function(zephyr_sources_cogeno)
        cogeno_sources(zephyr EXTERN ${ARGN})
    endfunction()

    function(zephyr_sources_cogeno_ifdef feature_toggle)
        if(${${feature_toggle}})
            zephyr_sources_cogeno(${ARGN})
        endif()
    endfunction()

    function(zephyr_library_sources_cogeno)
        cogeno_sources(${ZEPHYR_CURRENT_LIBRARY} EXTERN ${ARGN})
    endfunction()

    function(zephyr_library_sources_cogeno_ifdef feature_toggle)
        if(${${feature_toggle}})
            zephyr_library_sources_cogeno(${ARGN})
        endif()
    endfunction()

    function(zephyr_library_includes_cogeno)
        cogeno_sources(zephyr_interface EXTERN INCLUDES ${ARGN}
                       INCLUDE_DIR "${CMAKE_BINARY_DIR}/zephyr/include/generated")
    endfunction()
```

(continues on next page)

(continued from previous page)

```
    function(zephyr_library_includes_cogeno_ifdef feature_toggle)
        if(${${feature_toggle}})
            zephyr_library_includes_cogeno(${ARGN})
        endif()
    endfunction()

    # Add include directory to the directories scanned for syscall include files
    # - workaround
    macro(zephyr_syscall_include_dirs)
        foreach(dir ${ARGN})
            get_filename_component(dir "${dir}" REALPATH)
            if(${dir} IN_LIST SYSCALL_INCLUDE_DIRS)
                continue()
            endif()
            list(APPEND SYSCALL_INCLUDE_DIRS ${dir})
        endforeach()
        SET(SYSCALL_INCLUDE_DIRS  "${SYSCALL_INCLUDE_DIRS}" CACHE INTERNAL "SYSCALL_
↪INCLUDE_DIRS")
    endmacro()

# This is an ESP IDF project
# --------------------------
elseif(DEFINED ESP_PLATFORM AND ESP_PLATFORM)
    message(STATUS "Cogeno added to ESP platform")

# Could not identify the type of project
# --------------------------------------
else()
    message(STATUS "Cogeno added to unknown platform")

endif()
```

CHAPTER

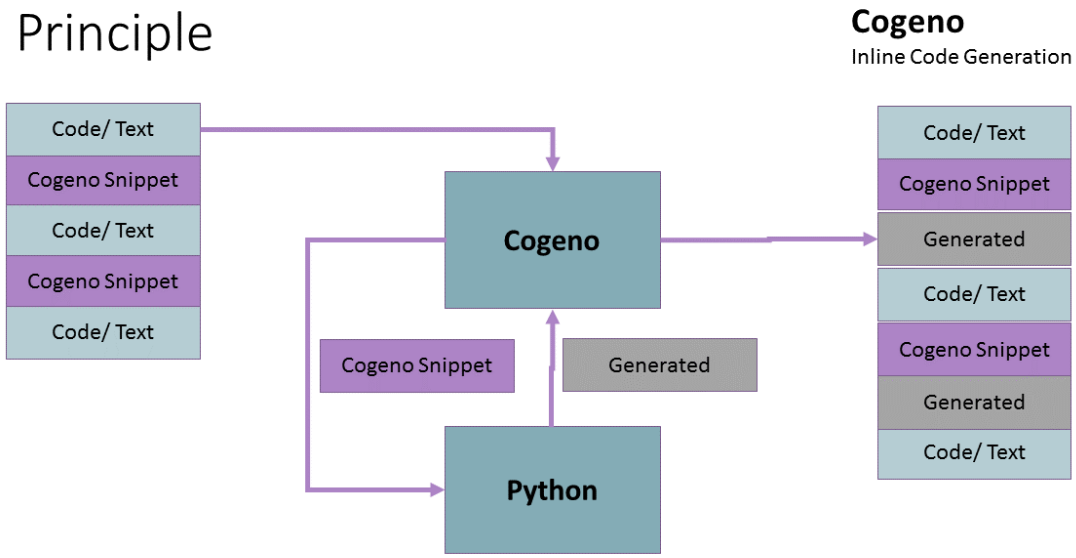
EIGHT

DEVELOPMENT

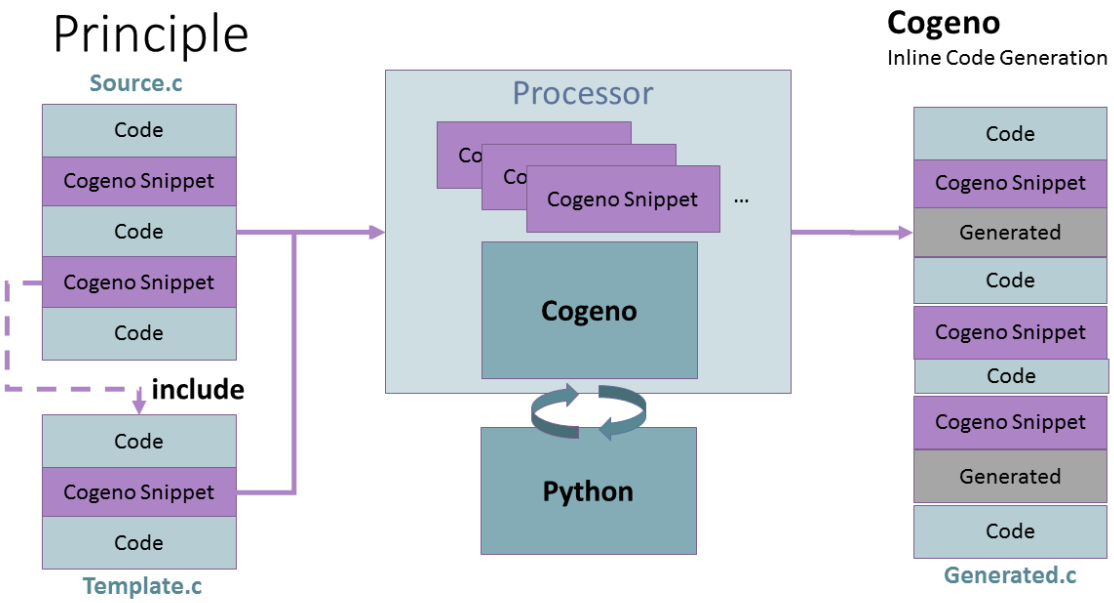
8.1 Code generation principle

How code generation works with cogeno.

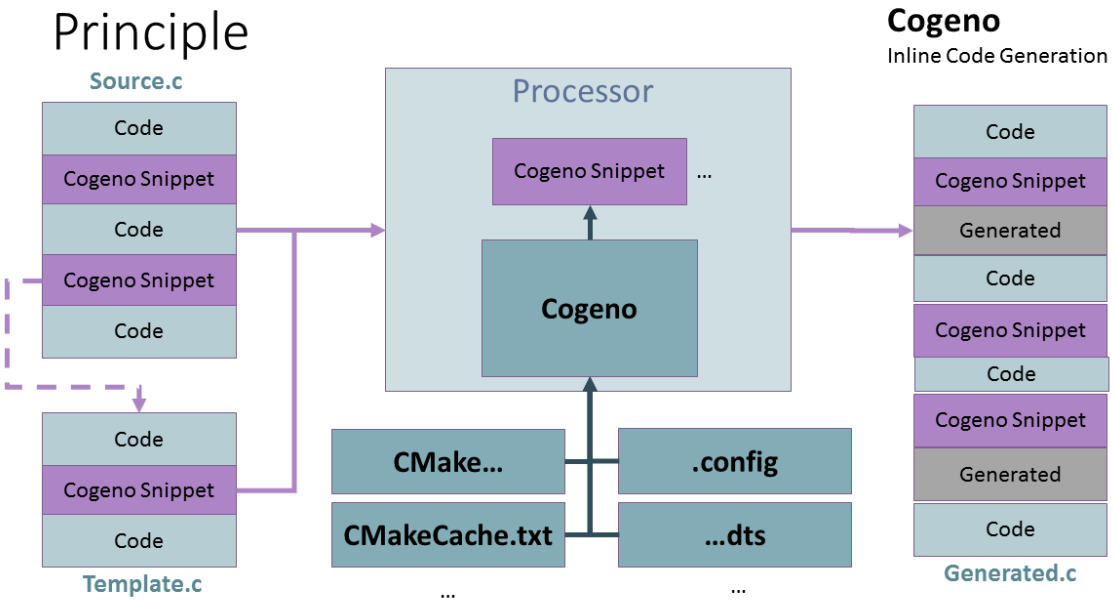
8.1.1 Principle



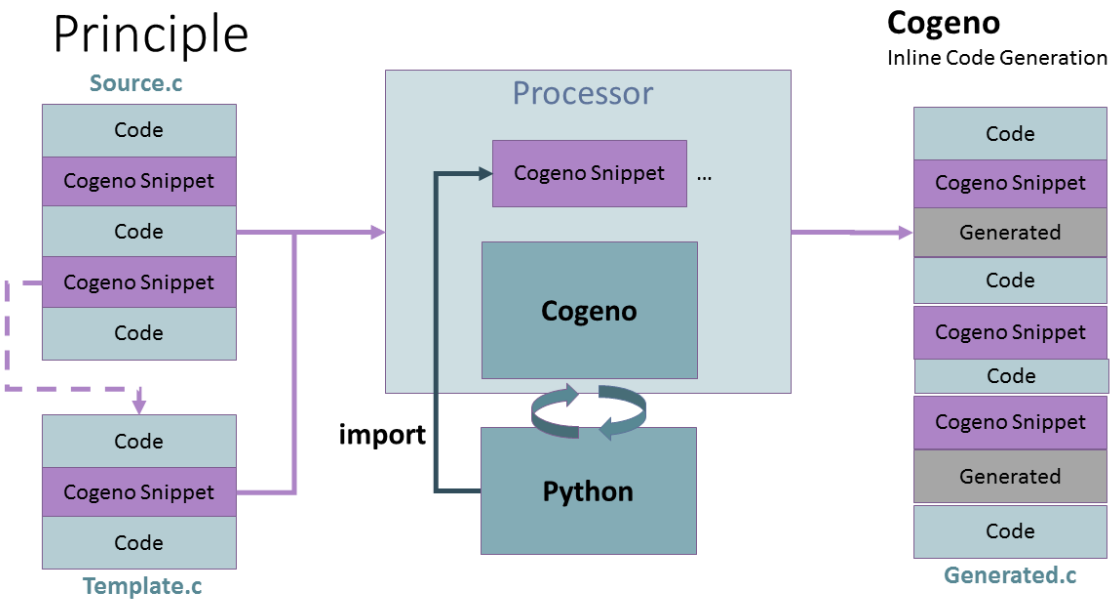
8.1.2 Inclusion of other inline code



8.1.3 Access to project data



8.1.4 Import of Python modules



8.2 Cogeno API

cogeno is a Python module that provides access to the public functions of the class: `CodeGenerator` and the sub-classes of it. See *Code generation functions* for a description of all `cogeno` module's functions.

The interfaces listed hereafter are the internal interfaces of Cogeno. The Cogeno `CodeGenerator` is made of a set of `Mixin` classes that bundle specific generator functionality. The `CodeGenerator` stores it´s states in `Context` class objects.

- *CodeGenerator*
  - *ContextMixin*
  - *ErrorMixin*
  - *ImportMixin*
  - *IncludeMixin*
  - *InlineGenMixin*
  - *Jinja2GenMixin*
  - *LockMixin*
  - *LogMixin*
  - *OptionsMixin*
  - *OutputMixin*

- *PathsMixin*
- *PyGenMixin*
- *RedirectableMixin*
- *StdModulesMixin*

• *Context*

8.2.1 CodeGenerator

cogeno.generator.CodeGenerator : public cogeno.context.ContextMixin , public cogeno.options

Public Functions

__init__ (*self*)

cogeno_state (*self*)
numeric cogeno state id

Public Static Attributes

cogeno_module = None

cogeno_module_states = []

The `CodeGenerator` class includes (sub-classes) several `mixin` classes:

ContextMixin

cogeno.context.ContextMixin : public object
Subclassed by *cogeno.generator.CodeGenerator*

Public Functions

context (*self*)
Get actual code generation context.

Return context

context_enter (*self*, *context*)
Switch to new code generation context.

Parameters

- `The`: new context

context_exit (*self*)
Switch back from code generation context.

Write to output file in case we are leaving the top level context.

Return the outstring of the context just left

context_out (*self*, *text*)
Output text to current context.

Parameters

- `text`: Text string

ErrorMixin

cogeno.error.ErrorMixin : public object
Subclassed by *cogeno.generator.CodeGenerator*

Public Functions

error (*self*, *msg='Error raised by cogeno generator.'*, *frame_index=0*, *lineno=0*)
Raise Error exception.

Extra information is added that maps the python snippet line seen by the Python interpreter to the line of the file that inlines the python snippet.

Parameters

- `msg`: [optional] exception message
- `frame_index`: [optional] Call frame index. The call frame offset of the function calling *error()*. Zero if directly called in a snippet. Add one for every level of function call.
- `lineno`: [optional] line number within template

ImportMixin

cogeno.importmodule.ImportMixin : public object
Subclassed by *cogeno.generator.CodeGenerator*

Public Functions

import_module (*self*, *name*)
Import a Cogeno module.

Import a module.

Module file is searched in current directory or modules directories.

Parameters

- `name`: Module to import. Specified without any path.

import_extensions (*self*, *extensions_paths*, *update=False*)
Import extension modules.

Extension paths are searched for extension modules.

Parameters

- `extensions_paths`: Directory paths for extensions
- `update`: Optional, on true the modules are reloaded if already loaded

import_extensions_from_option (*self*, *update=False*)
Import extension modules given in extension:paths option.

Extension paths are searched for extension modules. Thes modules are imported. As modules may update the option in itself the import is done recursively.

Parameters

- `update`: Optional, on true the modules are reloaded if already loaded

IncludeMixin

cogeno.include.IncludeMixin : public object
Subclassed by *cogeno.generator.CodeGenerator*

Public Functions

out_include (*self*, *include_file*)
Write the text from include_file to the output.

The `include_file` is processed by cogeno. Inline code generation in `include_file` can access the globals defined in the `including source file` before inclusion. The `including source file` can access the globals defined in the `include_file` (after inclusion).

Parameters

- `include_file`: Path of include file, either absolute path or relative to current directory or relative to templates directory (e.g. 'templates/drivers/simple_tmpl.c')

guard_include (*self*)
Prevent the current file to be included by `cogeno.out_include()` when called the next time.

InlineGenMixin

cogeno.inlinegen.InlineGenMixin : public object
Subclassed by *cogeno.generator.CodeGenerator*

Jinja2GenMixin

cogeno.jinja2gen.Jinja2GenMixin : public object
Subclassed by *cogeno.generator.CodeGenerator*

Public Functions

render (*self*, *template_spec*, *data=None*)
Render a Ninja2 template.

Parameters

- `template_spec`:
- `data`:

Jinja2SnippetLoader : public BaseLoader

Public Functions

__init__ (*self*, *template_name=None*, *template_code=None*)

get_source (*self*, *environment*, *template*)

Public Static Attributes

snippet_templates = dict()
Pool of known snippets.

LockMixin

cogeno.lock.LockMixin : public object
Subclassed by *cogeno.generator.CodeGenerator*

Public Functions

lock_file (*self*)
Lock file used for the current context.

Return lock file name

lock (*self*)
Get the global cogeno lock.

```
try:
     with cogeno.lock().acquire(timeout = 10):
         ...
except cogeno.lock_timeout():
    cogeno.error(...)
except:
    raise
```

Return Lock object

lock_timeout (*self*)
Lock timeout.

Return Lock timeout object

LogMixin

cogeno.log.LogMixin : public object
Subclassed by *cogeno.generator.CodeGenerator*

Public Functions

log(*self*, *message*, *message_type=None*, *end='\n'*, *logonly=True*)
Print message and write to log file.

Parameters

- `message`: Message
- `message_type`: If given will be prepended to the message
- `end`: Character to put at the end of the message. '\n' by default.
- `logonly`: Only write to logfile. True by default.

msg(*self*, *message*)
Print message to stdout and log with a "message: " prefix.

See *LogMixin::log()*

Parameters

- `message`: Message

warning(*self*, *message*)
Print message to stdout and log with a "warning: " prefix.

See *LogMixin::log()*

Parameters

- `message`: Message

prout(*self*, *message*, *end='\n'*)
Print message to stdout and log.

See *LogMixin::log()*

Parameters

- `message`: Message
- `end`: Character to put at the end of the message. '\n' by default.

prerr(*self*, *message*, *end='\n'*)
Print message to stderr and log with a "error: " prefix.

See *LogMixin::log()*

Parameters

- `message`: Message
- `end`: Character to put at the end of the message. '\n' by default.

OptionsMixin

cogeno.options.OptionsMixin : public object

Subclassed by *cogeno.generator.CodeGenerator*

Public Functions

option (*self*, *name*)

Get option of actual context.

Return option value

Parameters

- `name`: Name of option

options_add_argument (*self*, *args*, *kwargs*)

Add option arguments to option parser of actual context.

Cogeno modules may add arguments to the cogeno option parser. The argument variables given to cogeno are rescanned after new option arguments are provided.

```
def mymodule(cogeno):
    if not hasattr(cogeno, '_mymodule'):
        cogeno._mymodule = None

        cogeno.options_add_argument('-m', '--mymodule', metavar='FILE',
            dest='mymodule_file', action='store',
            type=lambda x: cogeno.option_is_valid_file(x),
            help='Load mymodule data from FILE.')

   if getattr(cogeno, '_mymodule') is not None:
       return cogeno._mymodule

   if cogeno.option('mymodule_file'):
       mymodule_file = cogeno.option('mymodule_file')
   else:
       cogeno.error(..., 2)

   ...
   cogeno._mymodule = ...
```

option_is_valid_file (*self*, *filepath*)

option_is_valid_directory (*self*, *directorypath*)

options_argv_append (*self*, *args*)

OutputMixin

cogeno.output.OutputMixin : public object
Subclassed by *cogeno.generator.CodeGenerator*

Public Functions

out (*self*, *args*)
Write text to the output.

The string arguments are concenated. The filters are then applied to the lines of the concenated string. The resulting string is written to the output.

OutputFilterDedent and *OutputFilterTrimBlankLines* make it easier to use multi-line strings, and they are only are useful for multi-line strings:

```
cogeno.out("""
   These are lines I
   want to write into my source file.
""", cogeno.OutputFilterDedent(), cogeno.OutputFilterTrimBlankLines())
```

Return output string

Parameters

- `*args`: Variable length argument list of strings and output filters.

outl (*self*, *args*)
Write text to the output with newline appended.

See OutputMixin::out(self, *args)

Return output string

Parameters

- `*args`: Variable length argument list of strings and output filters.

out_insert (*self*, *insert_file*, *args*)
Insert the text from the file into the output.

See OutputMixin::out(self, *args)

Return output string

Parameters

- `insert_file`: Path of file, either absolute path or relative to current directory or relative to templates directory.
- `*args`: Variable length argument list of strings and output filters.

class cogeno.**OutputFilter**
Subclassed by *cogeno.output.OutputMixin.OutputFilterDedent*, *cogeno.output.OutputMixin.OutputFilterLineNumbers*, *cogeno.output.OutputMixin.OutputFilterReplace*, *cogeno.output.OutputMixin.OutputFilterReSub*, *cogeno.output.OutputMixin.OutputFilterStartAt*, *cogeno.output.OutputMixin.OutputFilterStopAt*, *cogeno.output.OutputMixin.OutputFilterTemplateSubstitude*, *cogeno.output.OutputMixin.OutputFilterTrimBlankLines*

Public Functions

__call__(*self*, *line*, *lineno=None*, *lines=None*)
Apply filter.

Return line with filter applied. Maybe None in case the line is deleted
Parameters
- `line`: line to apply filter to
- `lineno`: line number - starts at 1
- `lines`: all lines

parselinenos(*self*, *spec*, *total*)
Parse a line number spec.

Return A list of wanted line numbers (line numbers start at 1).
Parameters
- `spec`: Line number spec (such as "1,2,4-6")
- `total`: Total number of lines

OutputFilterDedent : public cogeno.output.OutputMixin.OutputFilter
Remove common initial white space from the lines.

Parameters

- `new_indent`: Optional new indentation (after dedent)

Public Functions

__init__(*self*, *new_indent=''*)

__call__(*self*, *line*, *lineno=None*, *lines=None*)
Apply filter.

Return line with filter applied. Maybe None in case the line is deleted
Parameters
- `line`: line to apply filter to
- `lineno`: line number - starts at 1
- `lines`: all lines

Public Members

white_prefix

new_indent

OutputFilterLineNumbers : public cogeno.output.OutputMixin.OutputFilter
Filter lines by line numbers.

Filter lines that are given by line sumber specifications (such as "1,2,4-6").

Parameters

- `args`: list of arguments denoting line number specifications

Public Functions

__init__(*self*, *args*)

__call__(*self*, *line*, *lineno=None*, *lines=None*)
Apply filter.

Return line with filter applied. Maybe None in case the line is deleted
Parameters
- `line`: line to apply filter to
- `lineno`: line number - starts at 1
- `lines`: all lines

Public Members

args

line_numbers

OutputFilterReplace : public cogeno.output.OutputMixin.OutputFilter
Replace substring.

Parameters

- `old`: old substring to replace
- `new`: new substring which will replace the old substring. if new is None and the resulting line is empty it is deleted.
- `count`: (optional) the number of times to replace the old substring with the new substring

Public Functions

__init__(*self*, *old*, *new*, *count=None*)

__call__(*self*, *line*, *lineno=None*, *lines=None*)

Public Members

old

new

count

OutputFilterReSub : public cogeno.output.OutputMixin.OutputFilter
Substitude regular expression pattern.

Replace the leftmost non-overlapping occurrences of pattern in each line by the replacement repl.

Parameters

- `pattern`: Pattern for replacement. Pattern is a string that will be compiled to a pattern object.
- `repl`: Replacement. Repl can be a string or a function. If repl is a function, it is called for every non-overlapping occurrence of pattern.
- `count`: (optional) maximum number of pattern occurrences to be replaced

Public Functions

__init__(*self*, *pattern*, *repl*, *count=0*, *flags=0*)

__call__(*self*, *line*, *lineno=None*, *lines=None*)

Public Members

pattern

repl

count

flags

OutputFilterStartAt : public cogeno.output.OutputMixin.OutputFilter

Start output at pattern.

Parameters

- `start_at`: Start pattern

Public Functions

__init__(*self*, *start_at*)

__call__(*self*, *line*, *lineno=None*, *lines=None*)

Public Members

start_at

started

OutputFilterStopAt : public cogeno.output.OutputMixin.OutputFilter

Stop output at pattern.

Parameters

- `stop_at`: Stop pattern

Public Functions

__init__(*self*, *stop_at*)

__call__(*self*, *line*, *lineno=None*, *lines=None*)

Public Members

stop_at

stopped

OutputFilterTemplateSubstitude : public cogeno.output.OutputMixin.OutputFilter

Substitude template placeholders.

Template placeholder substitution supports $-based substitutions, using the following rules:

- $$ is an escape; it is replaced with a single $.
- $identifier names a substitution placeholder matching a mapping key of "identifier". By default, "identifier" is restricted to any case-insensitive ASCII alphanumeric string (including underscores) that starts with an underscore or ASCII letter. The first non-identifier character after the $ character terminates this placeholder specification.
- ${identifier} is equivalent to $identifier. It is required when valid identifier characters follow the placeholder but are not part of the placeholder, such as "${noun}ification".

At least up to 4 nesting levels of placeholders are substituded, e.g.:

- ${placeholder_level1} : mapping = 'placeholder_level1' : 'holder'
- ${place${placeholder_level1}_level2} : mapping = 'placeholder_level2' : 'placeholder'
- ${${placeholder_level2}_level3} : mapping = 'placeholder_level3' : 'placeholder_level'
- ${${placeholder_level3}4} : mapping = 'placeholder_level4' : 'success'

If more than one placeholder patterns are provided the substitution is done for each pattern sequencing through the patterns list.

Parameters

- `mapping`: Mapping is any dictionary-like object with keys that match the template placeholders.
- `patterns`: (optional) Patterns is a list of regular expressions describing the pattern for non-braced placeholders.

Public Functions

__init__(*self*, *mapping*, *patterns=['[_a-zA-Z][_a-zA-Z0-9] *']*)

__call__(*self*, *line*, *lineno=None*, *lines=None*)

Public Members

mapping

templates

Public Static Attributes

templates_classes = {}

OutputFilterTrimBlankLines : public cogeno.output.OutputMixin.OutputFilter
Remove initial and trailing blank lines from the block of lines.

Public Functions

__init__(*self*)

__call__(*self*, *line*, *lineno=None*, *lines=None*)
Apply filter.

Parameters
- `line`: line to apply filter to
- `lineno`: line number - starts at 1
- `lines`: all lines return line with filter applied. Maybe None in case the line is deleted

Public Members

trim_inital_lines

trim_trailing_lines_start

PathsMixin

cogeno.paths.PathsMixin : public object
Subclassed by *cogeno.generator.CodeGenerator*

Public Functions

modules_paths_append(*self*, *path*)

modules_paths(*self*)

templates_paths_append(*self*, *path*)

templates_paths(*self*)

template_path(*self*)

Public Static Functions

path_walk(*top*, *topdown=False*, *followlinks=False*)
Walk directory tree.

For each directory in the tree rooted at directory top (including top itself), it yields a 3-tuple (dirpath, dirnames, filenames):

- dirpath: the path to the directory.
- dirnames: list of the paths of the subdirectories in dirpath
- filenames: list of the paths of the non-directory files in dirpath

See Python docs for os.walk, exact same behavior but it yields Path() instances instead. From: http://ominian.com/2016/03/29/os-walk-for-pathlib-path/

Return yields a 3-tuple (dirpath, dirpathes, filepathes)

Parameters

- `top`: root of directory tree
- `topdown`: if topdown is True, the triple for a directory is generated before the triples for any of its subdirectories (directories are generated top-down). If topdown is False, the triple for a directory is generated after the triples for all of its subdirectories (directories are generated bottom-up).
- `followlinks`:

rmtree (*top*)
Delete an entire directory tree.

Parameters

- `top`: root of directory tree

find_template_files (*top*, *marker*, *suffix='.c'*)
Find template files.

Return List of template file pathes

Parameters

- `marker`: Marker as b'my-marker'
- `suffix`:

cogeno_path()

find_file_path (*file_name*, *paths*)

PyGenMixin

cogeno.pygen.PyGenMixin : public object
Subclassed by *cogeno.generator.CodeGenerator*

RedirectableMixin

cogeno.redirectable.RedirectableMixin : public object
Subclassed by *cogeno.generator.CodeGenerator*, cogeno.redirectable.Redirectable

Public Functions

set_standard_streams (*self*, *stdout=None*, *stderr=None*)
Redirect status and error reporting.

Assign new files for standard out and/or standard error.

Parameters

- `stdout`:
- `stderr`:

StdModulesMixin

cogeno.stdmodules.StdModulesMixin : public object
Subclassed by *cogeno.generator.CodeGenerator*

Public Functions

edts (*self*, *force_extract=False*)
Get the extended device tree database.

Return Extended device tree database.

cmake (*self*)
Get the cmake variables database.

Return CMake variables database.

cmake_variable (*self*, *variable_name*, *default='<unset>'*)
Get the value of a CMake variable.

If variable_name is not provided to cogeno by CMake the default value is returned.

A typical set of CMake variables that are not available in the `CMakeCache.txt` file and have to be provided as defines to cogeno if needed:

- "PROJECT_NAME"
- "PROJECT_SOURCE_DIR"
- "PROJECT_BINARY_DIR"
- "CMAKE_SOURCE_DIR"
- "CMAKE_BINARY_DIR"
- "CMAKE_CURRENT_SOURCE_DIR"
- "CMAKE_CURRENT_BINARY_DIR"
- "CMAKE_CURRENT_LIST_DIR"
- "CMAKE_FILES_DIRECTORY"
- "CMAKE_PROJECT_NAME"
- "CMAKE_SYSTEM"

- "CMAKE_SYSTEM_NAME"
- "CMAKE_SYSTEM_VERSION"
- "CMAKE_SYSTEM_PROCESSOR"
- "CMAKE_C_COMPILER"
- "CMAKE_CXX_COMPILER"
- "CMAKE_COMPILER_IS_GNUCC"
- "CMAKE_COMPILER_IS_GNUCXX"

Return value

Parameters

- `variable_name`: Name of the CMake variable
- `default`: Default value

cmake_cache_variable(*self*, *variable_name*, *default='<unset>'*)
Get the value of a CMake variable from CMakeCache.txt.

If variable_name is not given in `CMakeCache.txt` the default value is returned.

Return value

Parameters

- `variable_name`: Name of the CMake variable
- `default`: Default value

configs(*self*, *force_extract=False*)
Get the configuration variables database.

Return Configuration variables database.

config_properties(*self*)
Get all config properties.

The property names are the ones config file.

Return A dictionary of config properties.

config_property(*self*, *property_name*, *default='<unset>'*)
Get the value of a configuration property fromthe config file.

If property_name is not given in .config the default value is returned.

Return property value

Parameters

- `property_name`: Name of the property
- `default`: Property value to return per default.

8.2.2 Context

cogeno.context.Context : public object
Context for code generation.

Public Functions

__init__(*self*, *generator*, *parent_context=None*, *generation_globals=None*, *options=None*, *eval_begin=None*, *eval_end=None*, *eval_adjust=None*, *delete_code=None*, *template_file=None*, *template=None*, *template_source_type=None*, *script_type=None*, *template_tabsize=None*, *templates_paths=None*, *modules_paths=None*, *jinja2_environment=None*, *output_file=None*, *log_file=None*, *lock_file=None*)
Initialise context object.

__str__(*self*)

__repr__(*self*)

parent(*self*)

generation_globals(*self*)

script_is_inline(*self*)

script_is_python(*self*)

script_is_jinja2(*self*)

script_type(*self*)

template_is_snippet(*self*)
Template is a snippet.

Snippets are parts of the template of the parent context.

Return True in case the template is a snippet, False otherwise.

template_is_file(*self*)

template_is_string(*self*)

options(*self*)
Options.

out(*self*, *line*)
Add line.

outl(*self*, *line*)
Add line with newline.

CHAPTER

NINE

FREQUENTLY ASKED QUESTIONS

TBD

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

W

Z