



nRF Connect SDK and tools

Next generation SDK for Nordic wireless solutions,
and its tools and development environment

H2, 2022

Nordic Tech Tour 2022

nRF Connect



- nRF Connect is our umbrella of tools to help developers build and debug their applications quickly
- Consist of
 - nRF Connect SDK
 - nRF Connect for Desktop
 - nRF Connect for VS Code
 - nRF Cloud
 - Mobile applications

nRF Connect SDK

Introduction



nRF Connect SDK



- One code base and toolchain for nRF91, nRF53, nRF52 and nRF21 Series
 - Optional for nRF52 Series (>= v1.3.0)
- Includes LTE-M/NB-IoT/GPS, Bluetooth Low Energy, Bluetooth mesh, Thread/Zigbee, Matter, ESB, Gazell, NFC
- Bluetooth v5.3 PS1 Controller stack since v2.0.0

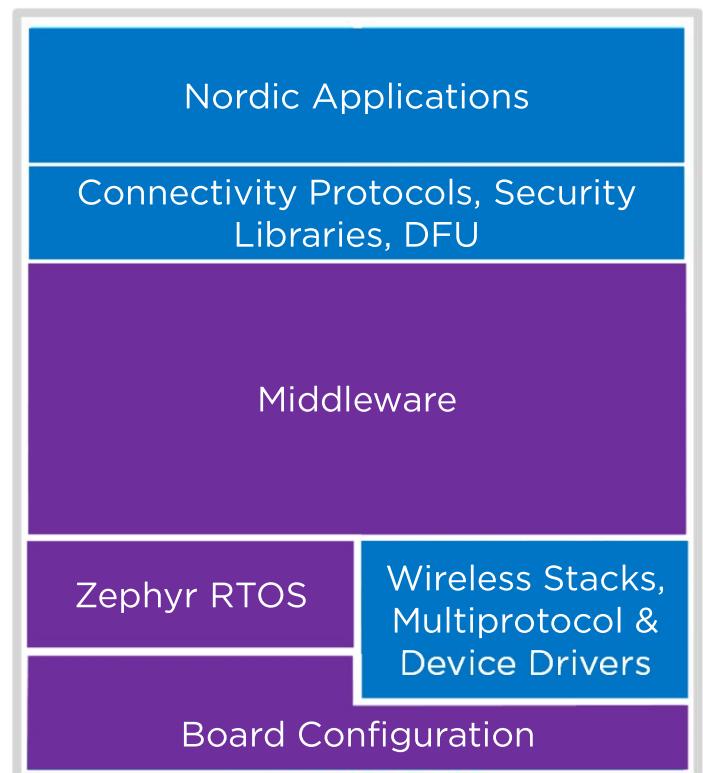


Slide 4

- PS0** Add nRF70 series?
Presley, Sam, 2022-08-29T14:46:26.036
- KP0 0** not available today
Kastnes, Paal, 2022-09-01T13:54:41.587
- PS1** Should this now read "Bluetooth 5.3"?
Presley, Sam, 2022-08-29T14:49:00.753

Code base

- Contains app code, connectivity protocols, wireless stacks and peripheral drivers
- Code is organized into several repositories (Nordic - blue and Open Source (OS) code - purple)

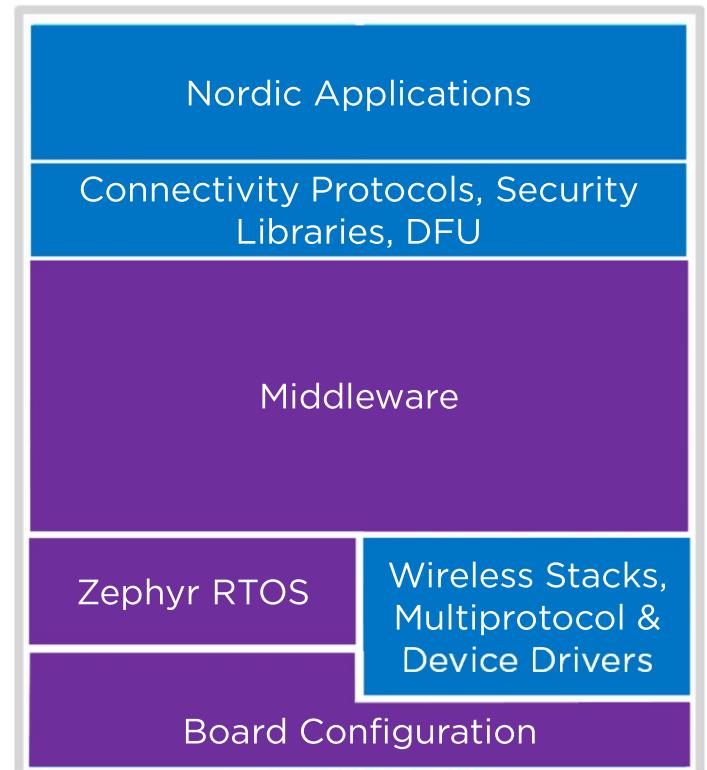


Slide 5

- PS0** No mention of what the colours mean, should we add a legend to explain that blue is Nordic, purple is open source?
Presley, Sam, 2022-08-29T14:50:14.495
- KP1** boxes don't line up in the diagram. Same issue on following slides as well. Becomes much more visible with the projectors.
Kastnes, Paal, 2022-09-01T13:56:17.132

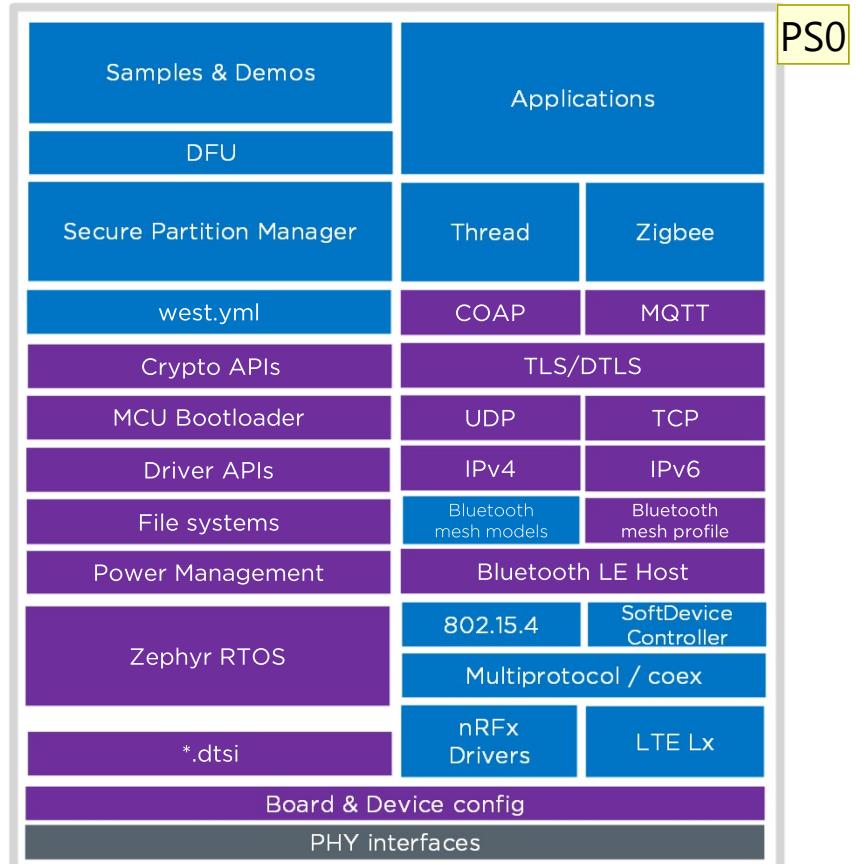
nRF Connect SDK repositories

- nRF: Application & connectivity protocols
- nrfxlib: compiled libraries where Nordic cannot distribute source code
- nrfx: peripheral drivers
- Zephyr: RTOS & board configuration (OS)
- MCUboot: Secure Bootloader (OS)
- Other repositories
 - Trusted Firmware-M, Matter, etc.



Code base in detail

- A wide range of wireless technologies and applications is supported by one integrated code base
- Code management, build and configuration tools allow developers to focus on the components required for their specific designs while having a powerful solution toolbox available



Slide 7

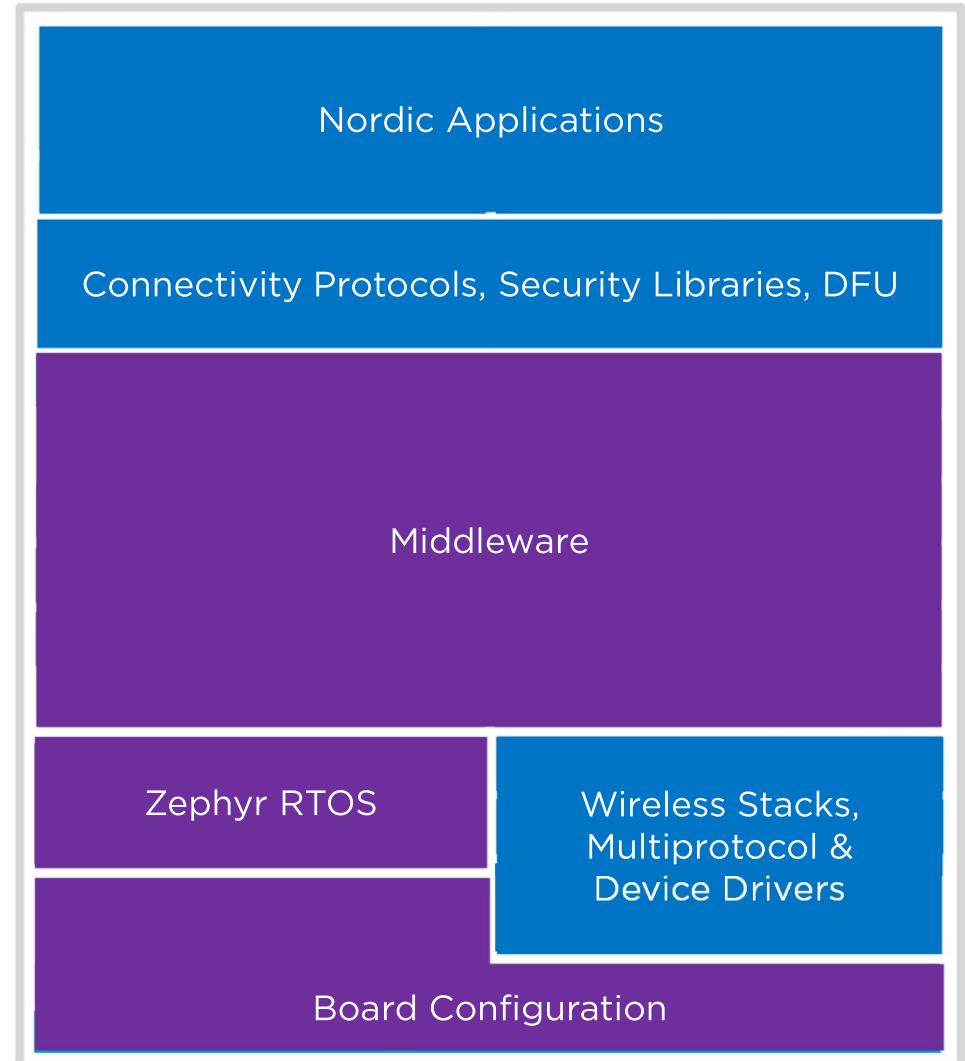
PS0

This block diagram was hard to read when projected

Presley, Sam, 2022-08-29T14:50:39.686

Nordic SW

- The main SDK repositories with samples, applications and connectivity stacks
- Source code exclusively written by Nordic Semiconductor
- Licensed with permissive Nordic 3-clause or 5-clause BSD license



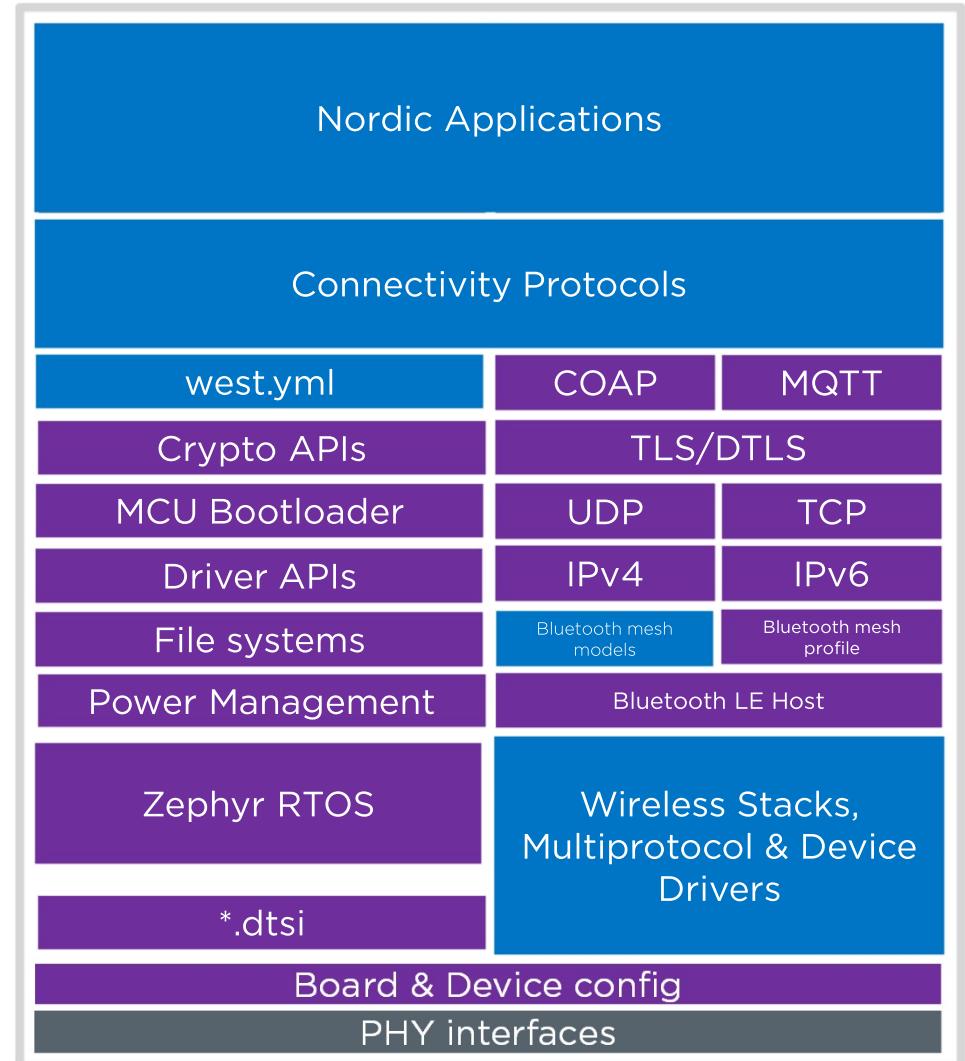
Slide 8

BA0

Must be de
animation before PDF
Bergerud, Aina, 2022-08-12T09:34:09.996

Open-source SW

- nRF Connect SDK re-distributes OS for standard platform components
- Nordic collaborates with communities of industry experts to deliver these components as part of the nRF Connect SDK
- OS repositories are licensed with permissive license (e.g. Apache 2.0)



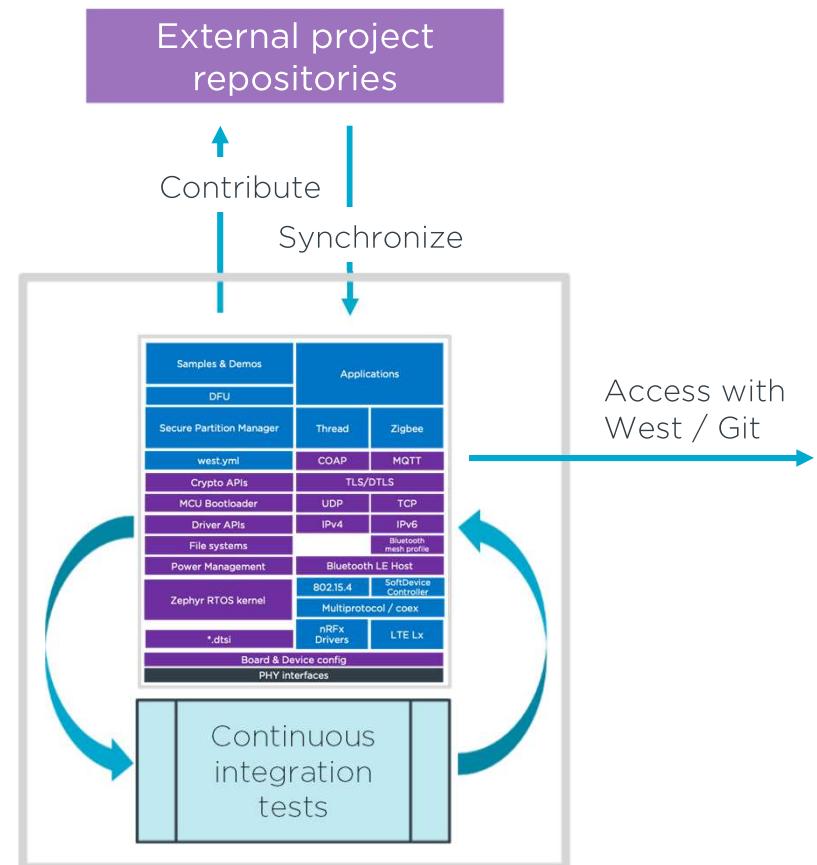
Slide 9

BA0

Must be de
animation before PDF
Bergerud, Aina, 2022-08-12T09:34:21.091

Nordic synchronizes with external repos

- nRF Connect SDK is a single platform
- All source code is distributed by Nordic
- Includes open-source code from external projects
- Nordic contributes to, and synchronizes with external projects
- Nordic runs integration test on all source code and manages configuration
- Customers clone a tag using git and west

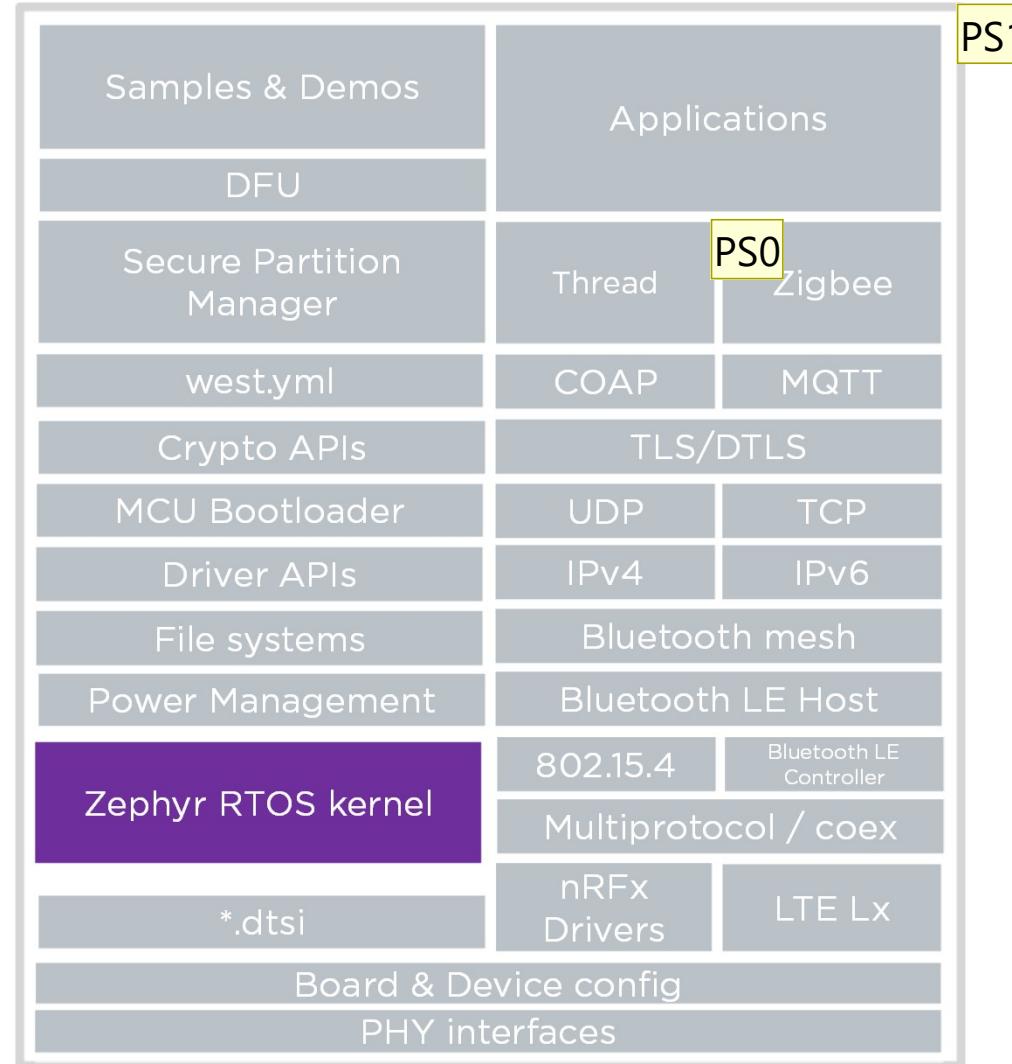


RTOS

Introduction

What is an RTOS?

- Real Time Operating System
 - Goal is to ensure predictable/deterministic execution pattern
 - Embedded systems often have strict timing requirements
 - Scheduler decides which task to execute at which time
 - Achievable by setting a priority for each execution thread

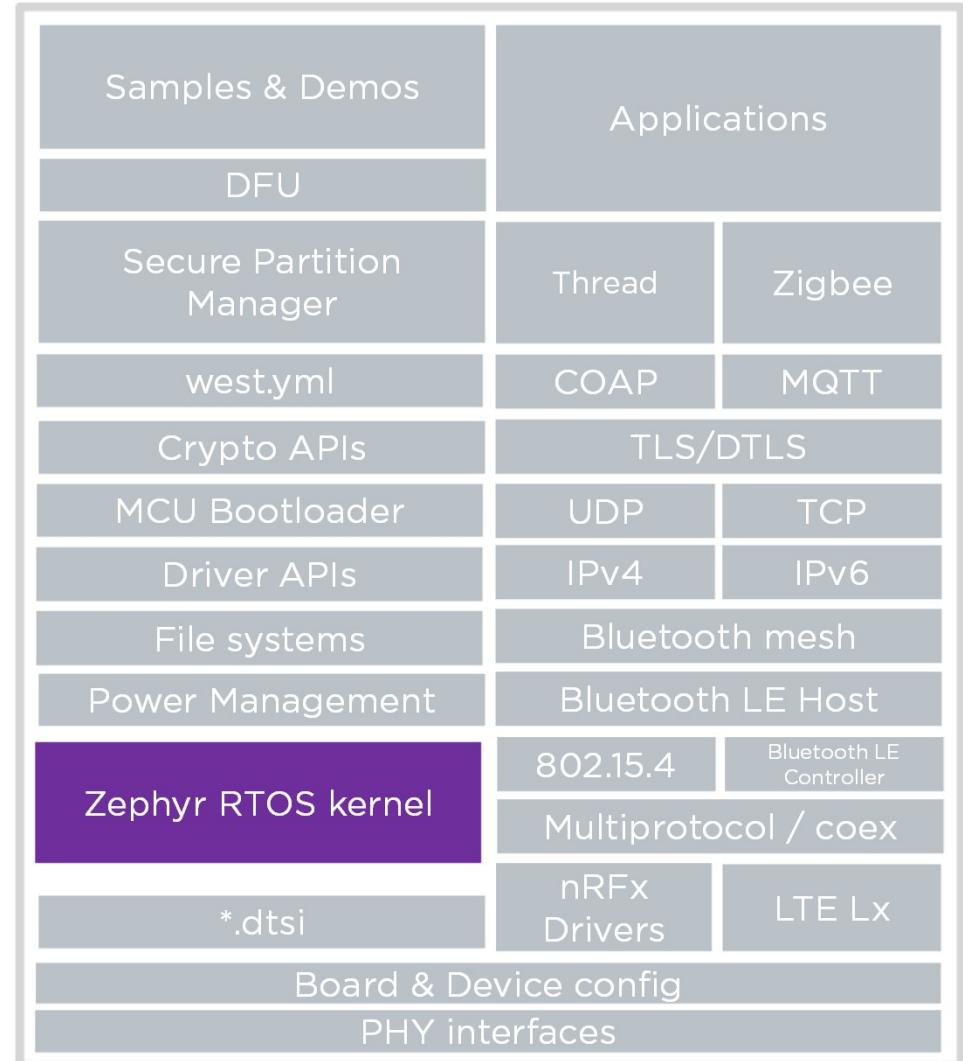


Slide 12

- PS0** The "Thread" box changes from this slide to the next
Presley, Sam, 2022-08-29T14:51:38.099
- PS1** This is a better way of highlighting what is being discussed on the block diagram rather than lots of colours
Presley, Sam, 2022-08-29T14:52:29.351

Why use an RTOS?

- How does an RTOS help a product developer?
 - Separation of Concern
 - More portable & re-usable applications
 - Controls application complexity in large memory devices
- Higher-level programming model
 - -> faster time to market





nRF Connect SDK

Details

Tools

- Use the same tools to manage your code
- Your code can be in separate repositories from nRF Connect SDK and open source
- Simplify incorporating new versions of SDK or patches
- Facilitate scalable development for multiple products and multiple teams in your organization



Tagged vs main branch

- Tagged versions of nRF Connect SDK are stable releases
 - <https://github.com/nrfconnect/sdk-nrf/tags>
 - V2.1.1 is latest tagged revision
 - Technical support is available
 - Use as starting point for development
- Main branch is most current development status
 - Newest version of nrf repository, not a stable release
 - No technical support available
 - Use if you need to test latest features earlier

nRF Connect SDK documentation

- [Documentation link](#)
- Click on arrow in top right to choose documentation tag
 - 2.x.99 refers to main branch
 - Latest tag is 2.1.1
- Click on the tabs on the top banner to switch to nrfxlib, Zephyr, MCUboot or other documentation pages



Manage source code and configurations

West
Multi-repository
management tool

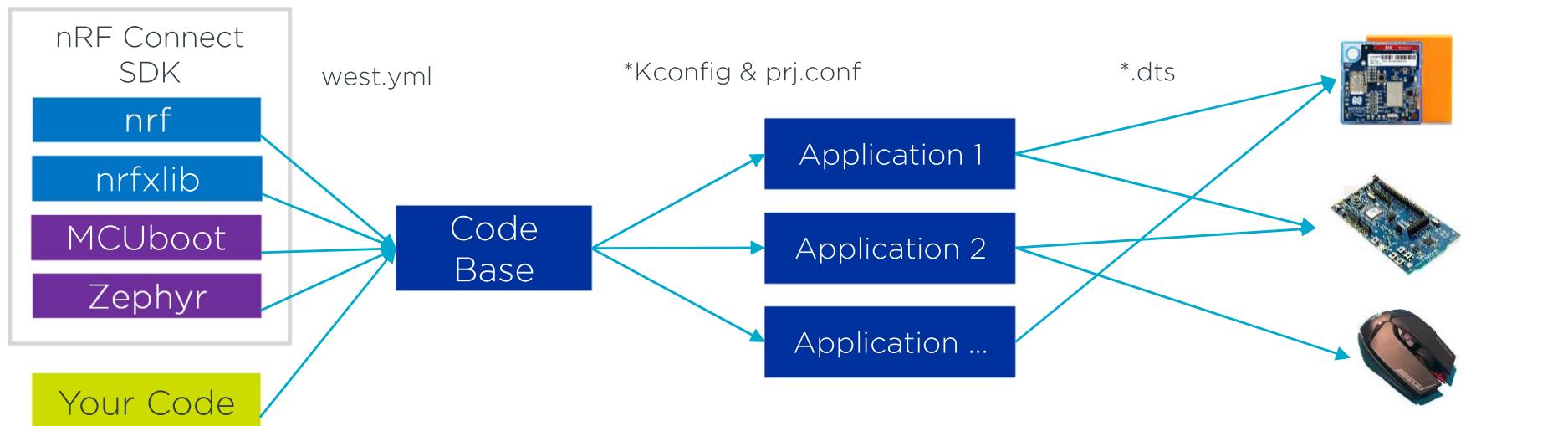
Kconfig
Source module / feature
configuration for compile

DeviceTree
Target Board / Device
description

Clone / update

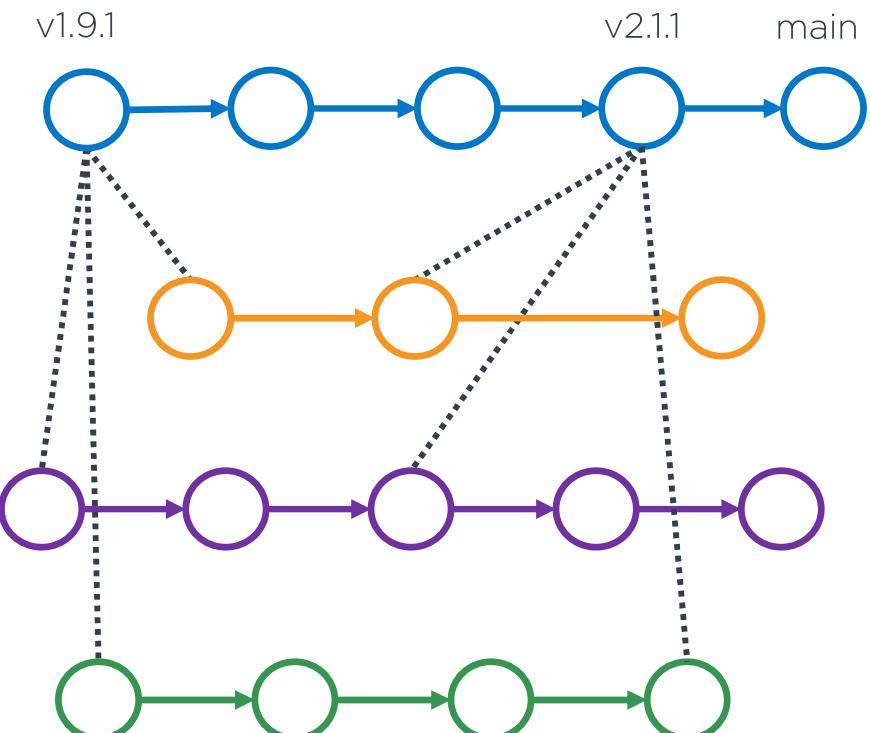
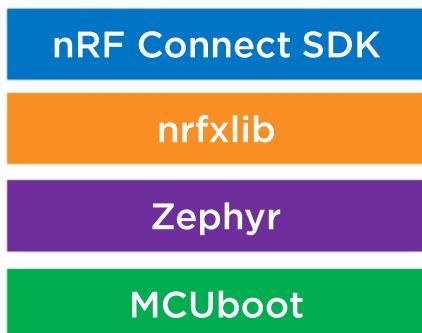
Configure features

Configure target



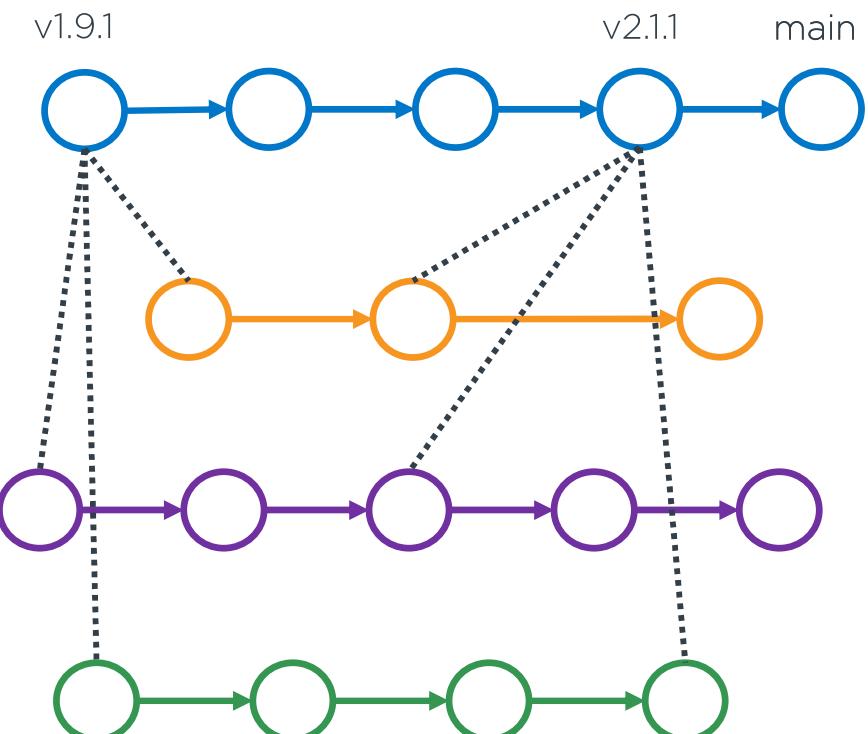
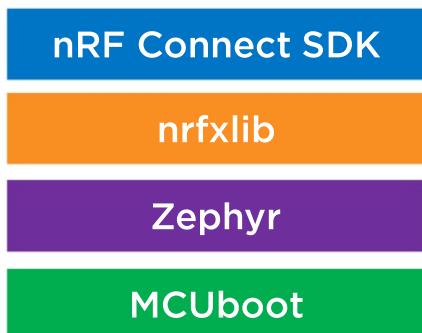
West tool

- West init
 - Initialize a new west installation
- git checkout <tag>
 - e.g. git checkout v1.8.0
- West update
 - Used to update the four repositories



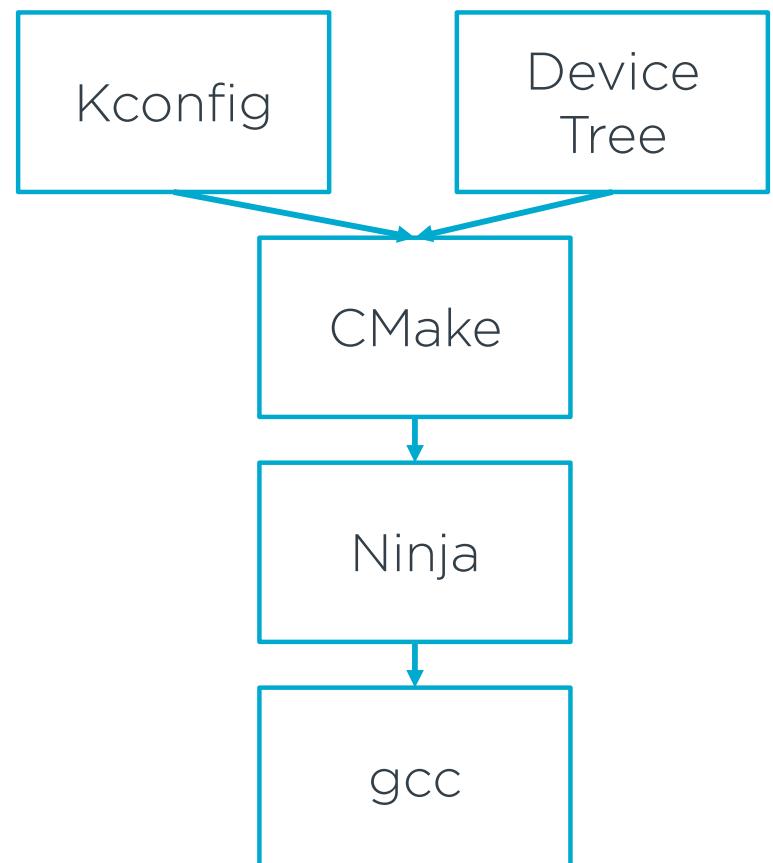
West tool

- Using git and west simplifies multi repository management by making it seem you are only updating one repo
- Simple to migrate between different tags



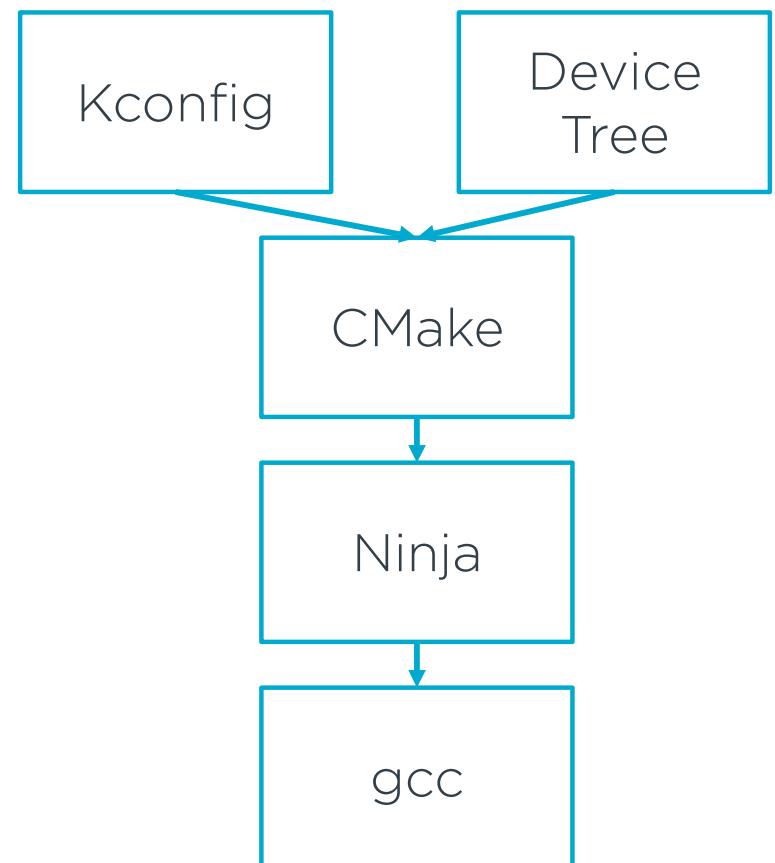
nRF Connect SDK Toolchain

- Kconfig
 - Generates definitions to configure the system (e.g. GPS, MQTT settings)
 - Generally located in an example folder
 - Software features defined in Kconfig, enabled in prj.conf file
 - Kconfig and prj.conf files merged into one .config file for CMake
 - → Configure software features without changing source code



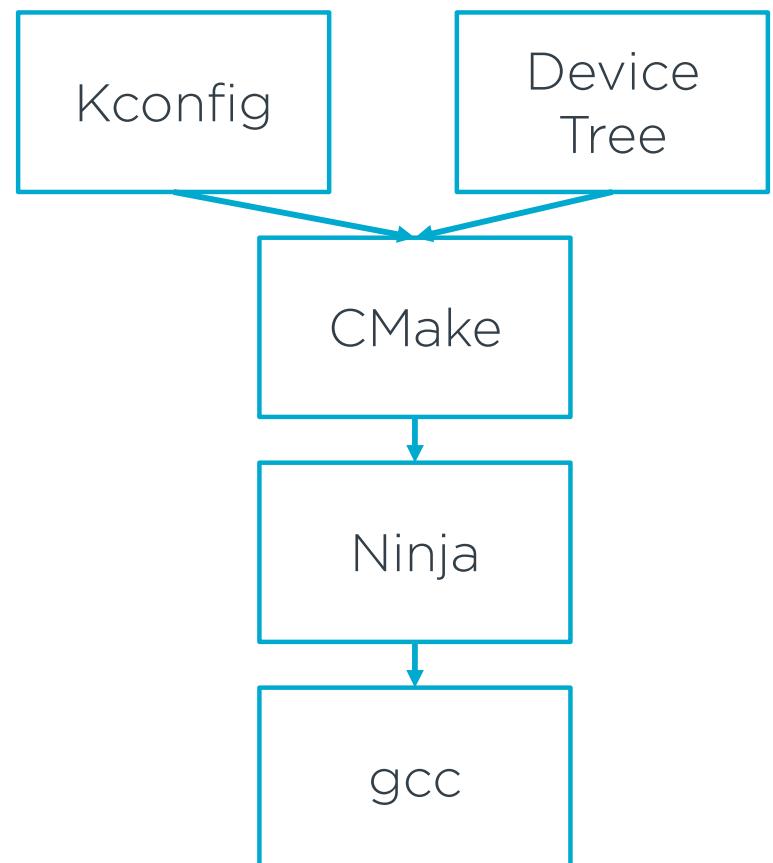
nRF Connect SDK Toolchain

- DeviceTree (dts,dtsi)
 - Describes HW, pin layout
 - Allows for flexible HW modification via an overlay file
 - → Build for different PCB designs and SoCs without changing source code



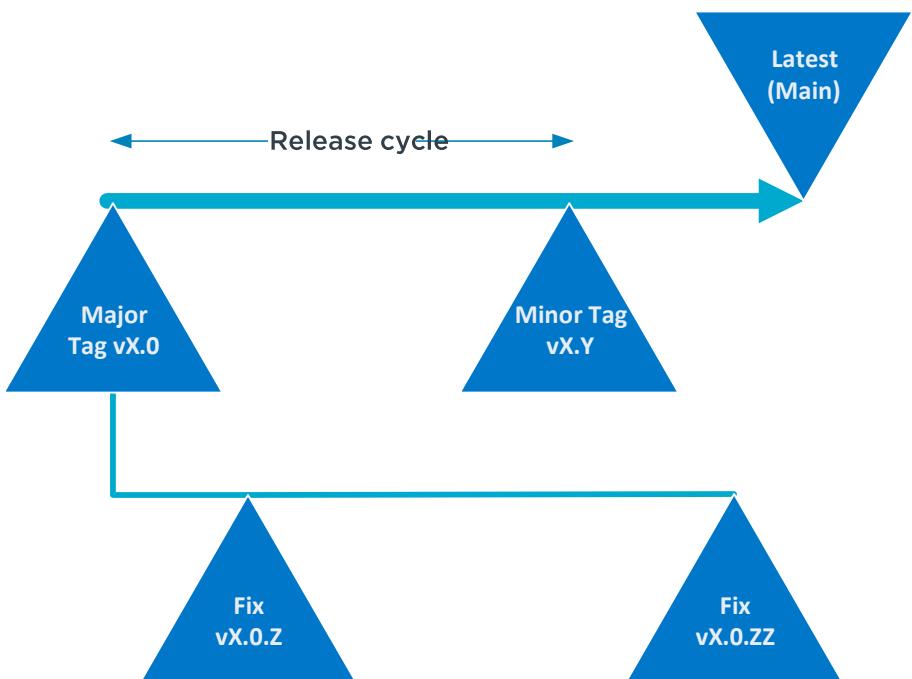
nRF Connect SDK Toolchain

- CMake
 - Generates build files
- Ninja
 - Similar to make
 - Faster than make when performing incremental builds
 - Requires CMake in order to generate build files
- gcc
 - Creates executables (hex file)



Release cycles

- Regular releases (e.g. quarterly)
- Publicly hosted on [GitHub](#)
- Fixes released as needed
 - Long term supported releases can have fixes applied and delivered after new releases
- Latest development version available
- Version control management with Git:
 - manage new version and fix adoption
 - tool supported merging



IDE support



- nRF Connect for Visual Studio Code
 - Built from the ground up for nRF Connect SDK
 - Highly extendable and configurable
 - CLI and GUI Interfaces
 - Cross-platform support
 - Windows, macOS, Linux
 - Create new board wizard
 - Rich set of [tutorial videos](#)

The screenshot shows the nRF Connect for Visual Studio Code interface. The main window has a dark theme. On the left, there's a sidebar with sections for 'WELCOME', 'APPLICATIONS' (AssetTracker), 'ASSETTRACKER' (main.c), 'ACTIONS' (Build, Pristine build, Gconfig, Flash, Debug, Debug with Ozone), and 'CONNECTED DEVICES' (NRF9160, VCOM0, VCOM1, VCOM2). The central area is a code editor with the file 'main.c' open, displaying C code for the AssetTracker application. The right side of the interface includes a terminal window showing build logs and a debug console. The bottom status bar indicates the terminal is disconnected.

```

main.c - Untitled (Workspace) - Visual Studio Code

File Edit Selection View Go Run Terminal Help
File Edit Selection View Go Run Terminal Help
WELCOME
APPLICATIONS
  AssetTracker (1)
    Build nRF9160 WiFi/BT
    Intended Product (1)
ASSETTRACKER (main.c)
  Inputs
  Output files
  Devices
    Overview
    Board
    API
    Apps
      App 1 (1 MB)
      macbook (1 KB)
      Image # 216 kB
      Import-Simonscare 192 kB
  Actions
    Build
    Pristine build
    Gconfig
    Flash
    Debug
    Debug with Ozone
  Connected devices
    NRF9160 (PCA10050)
      NRF9160.xsd, REV1
      VCOM0 COM0
      VCOM1 COM1
      VCOM2 COM1
  PREVIEW OUTPUT TERMINAL APP TERMINAL DEBUG CONSOLE
  [284/290] Building C object zephyr/Outfiles/zephyr_final.dir/dev_handles.c.o
  [285/290] Building C executable zephyr/zephyr.elf
  [285/290] Device capture
  Total Size: 297284 B 55.86%
  FLASH: 258644 B 592784 B 55.86%
  SRAM: 101022 B 179956 B 56.45%
  DFI L1SRAM: 0 B 2 KB 0.00%
  [286/290] Generating zephyr/macbook_primary.hex
  powershell
  nRF Connect Build...
  gdb-server-2
  nRF Terminal Disconnected
  Line 1 Col 1 Tab Stop 4 UTF-8 CR/LF C AssetTracker/Build
  
```

nRF Connect SDK fundamentals course

- Self-paced hands-on online course focusing on learning the essentials of nRF Connect SDK.
- Lots of hands-on exercises.
- Centralized up-to-date content.
- Protocol agnostic (cellular IoT, Bluetooth LE, Bluetooth mesh, etc..).
- Supports all our DKs and the Thingy:91.
- Ideal for new users of nRF Connect SDK or users switching from nRF5 SDK to nRF Connect SDK.
- Test your knowledge through interactive quizzes.

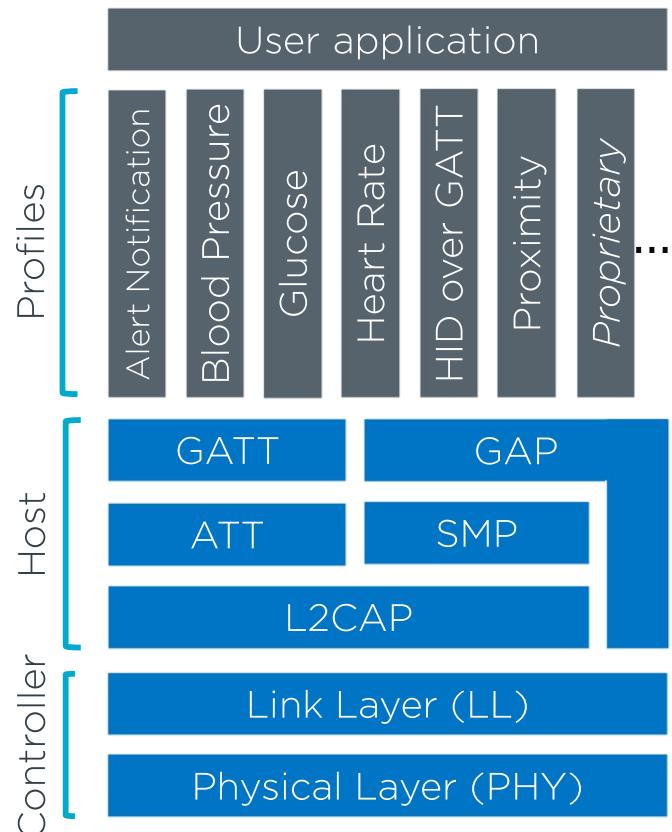


nRF Connect SDK

Evolution

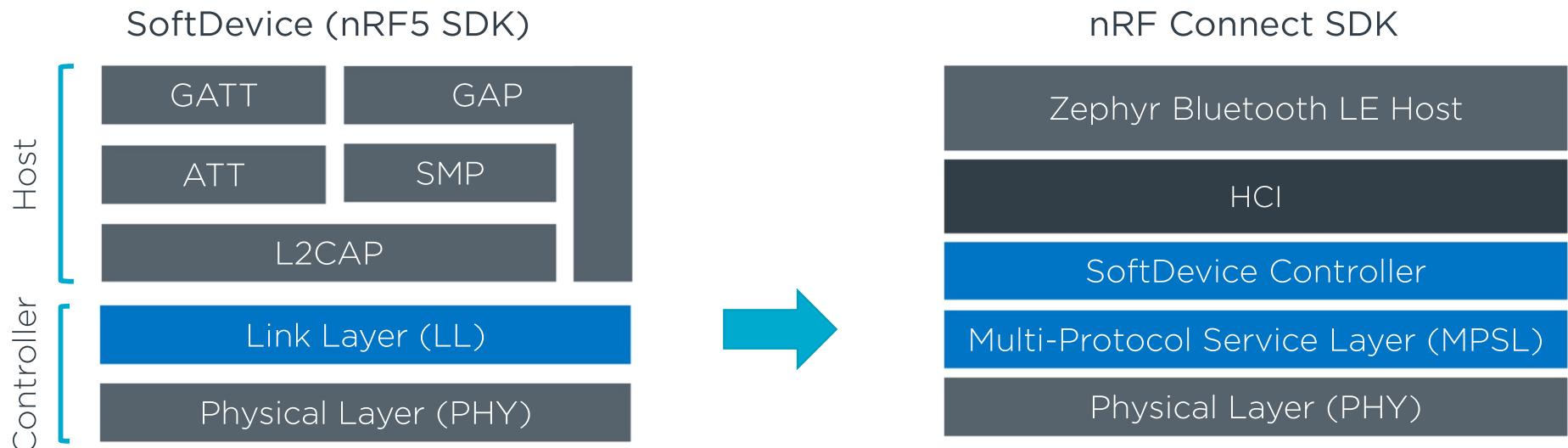
Ported SoftDevice Controller

- Market leaders in Bluetooth LE in large part to our nRF5 SDK + SoftDevice
- Link Layer (LL) enables best in class interoperability between SoC and smartphones and is difficult to get right
- Ported the SoftDevice LL to nRF Connect SDK
 - Ensures best in class smartphone interoperability in nRF Connect SDK vs nRF5 SDK + SoftDevice



SoftDevice Controller

- The standalone SoftDevice Controller library is now the default Bluetooth LE Controller for Bluetooth samples





nRF Connect SDK Tools

Tools and development environment for nRF Connect SDK

H2, 2022

Nordic Tech Tour 2022

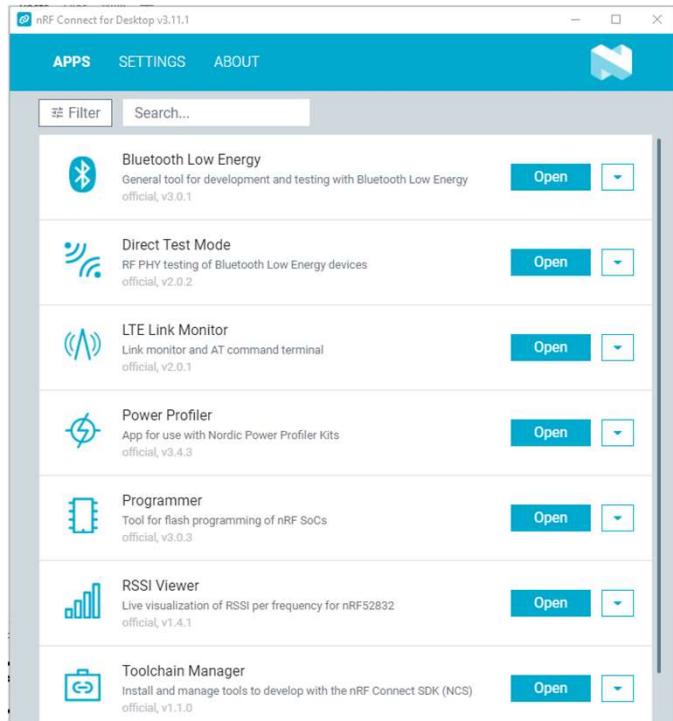
nRF Connect Tools

- Consist of
 - nRF Connect for Desktop
 - nRF Connect for VS Code
 - nRF Command line tools
 - Zephyr project tools

nRF Connect Tools supported platforms

- nRF Connect Tools has been developed and tested on the following operating systems:
 - Windows 10 running x86 CPUs
 - macOS 11 running x86 and Apple Silicon MCUs
 - Ubuntu 20.04LTS running x86 CPUs
- Older versions of the operating systems will typically not work
- Newer versions may work but this is not guaranteed

nRF Connect for Desktop

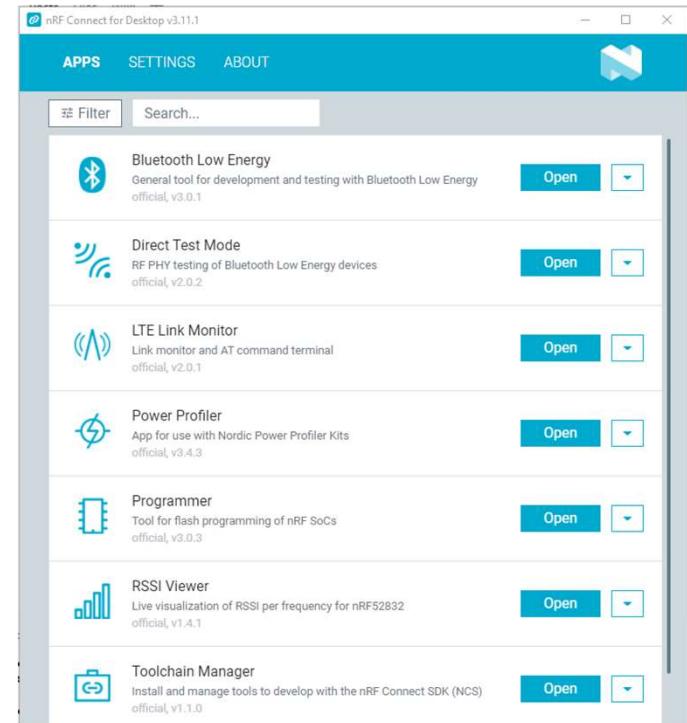


- Tools aimed at SW development and general use:
 - Toolchain Manager installs and manages tools related to nRF Connect SDK
 - Programmer is a great visual tool for flashing Nordic devices, updating modem FW on nRF91 Series
 - Power profiler used to optimize power consumption

Short range tools

In nRF Connect for Desktop

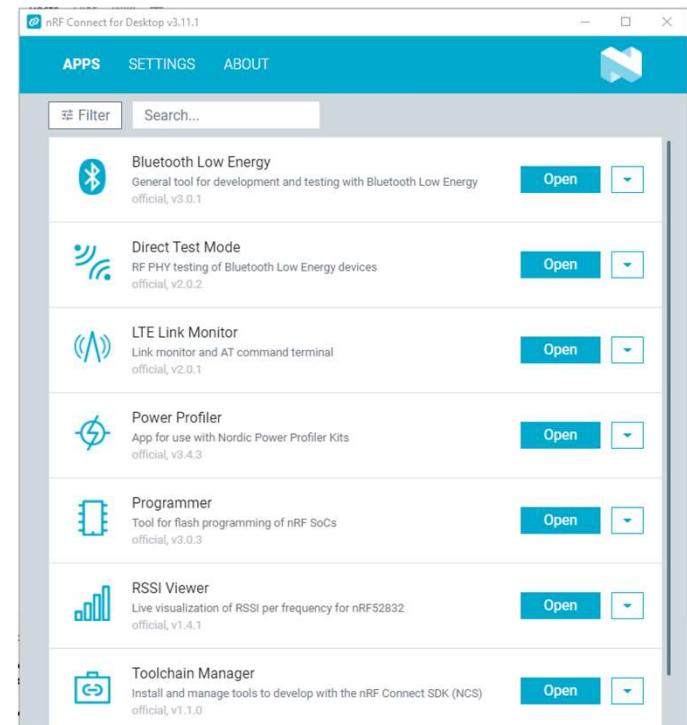
- Bluetooth Low Energy
 - Connect it to advertising devices
 - Discover their services
 - Maintain connection and parameters
- RSSI Viewer
 - Visualizes energy in all channels
- Direct Test Mode
 - Test TX and RX
 - Changes channel, transmit power, packet size



Cellular tools In nRF Connect for Desktop

- Trace Collector v2 preview
 - collect modem traces on nRF91 Series
 - Interface to Wireshark
- LTE Link Monitor
 - monitors nRF91 Series connection status
 - sends AT commands
 - updates certificates

KP2



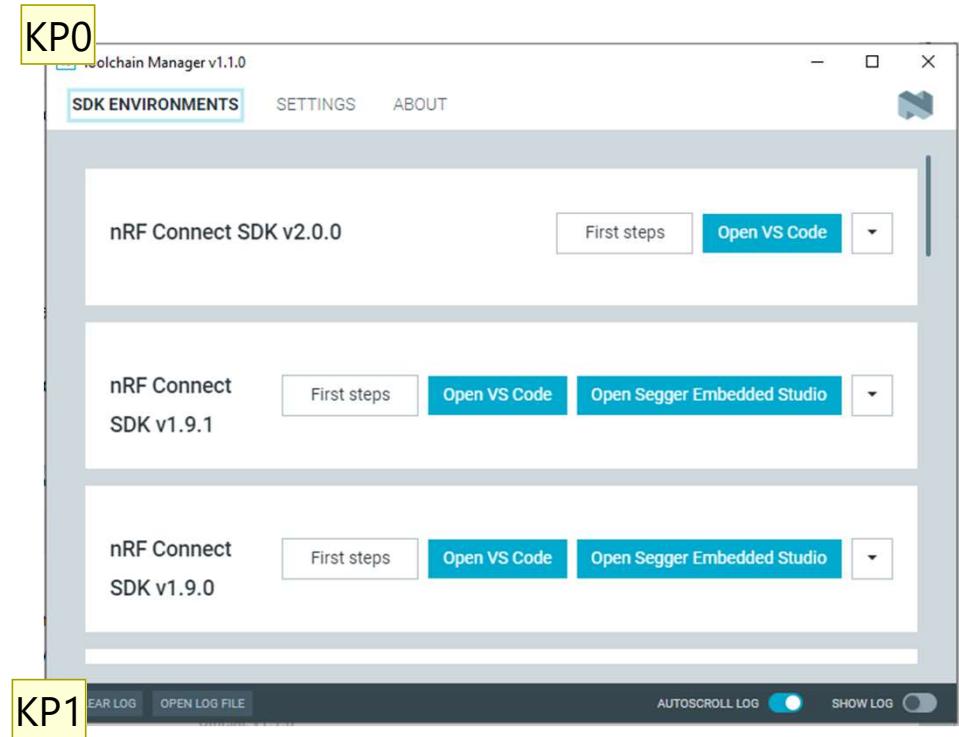
Slide 35

- KP0** Too long, make shorter and more concise
Kastnes, Paal, 2022-08-02T12:14:47.837
- KP1** Too long, make shorter and more concise
Kastnes, Paal, 2022-08-02T12:15:02.144
- KP2** No mention of developing for Short Range, don't we have any tools for that?
Kastnes, Paal, 2022-08-02T12:15:29.639
- HI2 0** Separate slides for short range and LTE
Hanssen, Ingar, 2022-08-02T15:17:20.960

Toolchain Manager



- Provides single click download of
 - Entire code base
 - Toolchain and supporting tools
- Available for
 - Windows, macOS for all nRF Connect SDKs
 - Linux for SDK 2.0 onwards
- Makes a copy of each version (tag)
 - allows working on multiple versions
- Helps on installing required tools



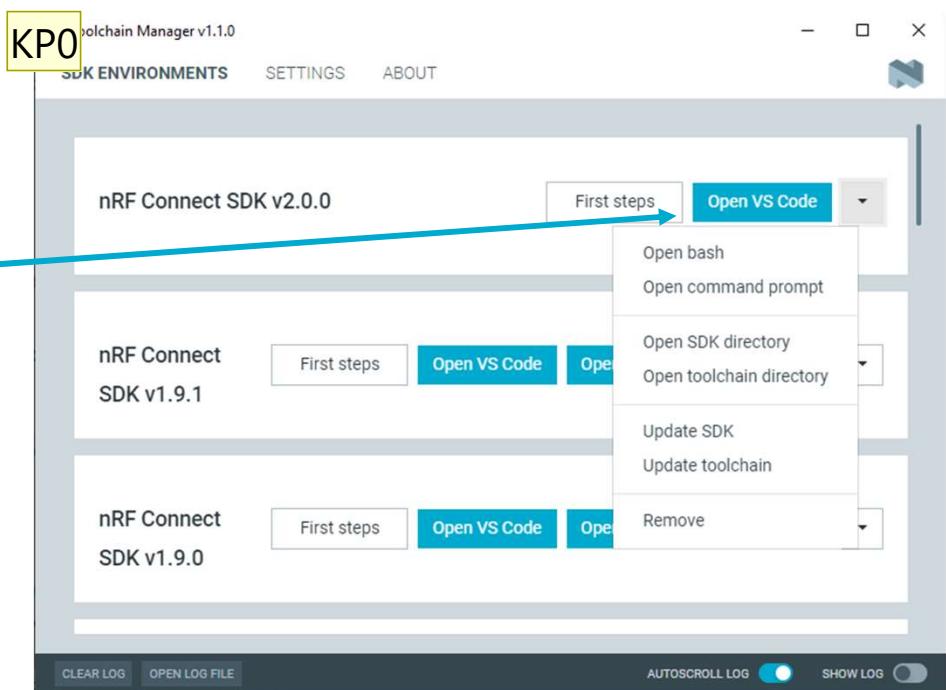
Slide 36

- KP0** If we can't run everything as non-admin then this is a moot point and we shouldn't have it in. And make text more "bullety": No Admin rights required"
Kastnes, Paal, 2022-08-02T12:18:32.441
- H10 0** Admin moved to readers notes. Bullified text
Hanssen, Ingar, 2022-08-02T15:16:26.892
- KP1** Remove this one as we don't want to talk about a deprecated tool. I take for granted it has been removed so it can't be downloaded any longer.....
Kastnes, Paal, 2022-08-02T12:19:14.630

Toolchain Manager



- VS Code is the recommended IDE
- VS Code may launch directly from your system/desktop
- Use the Open bash/command prompt/terminal to use this SDK version from CLI
- Segger Embedded Studio must be launched from the Toolchain Manager (not recommended for new designs)



Slide 37

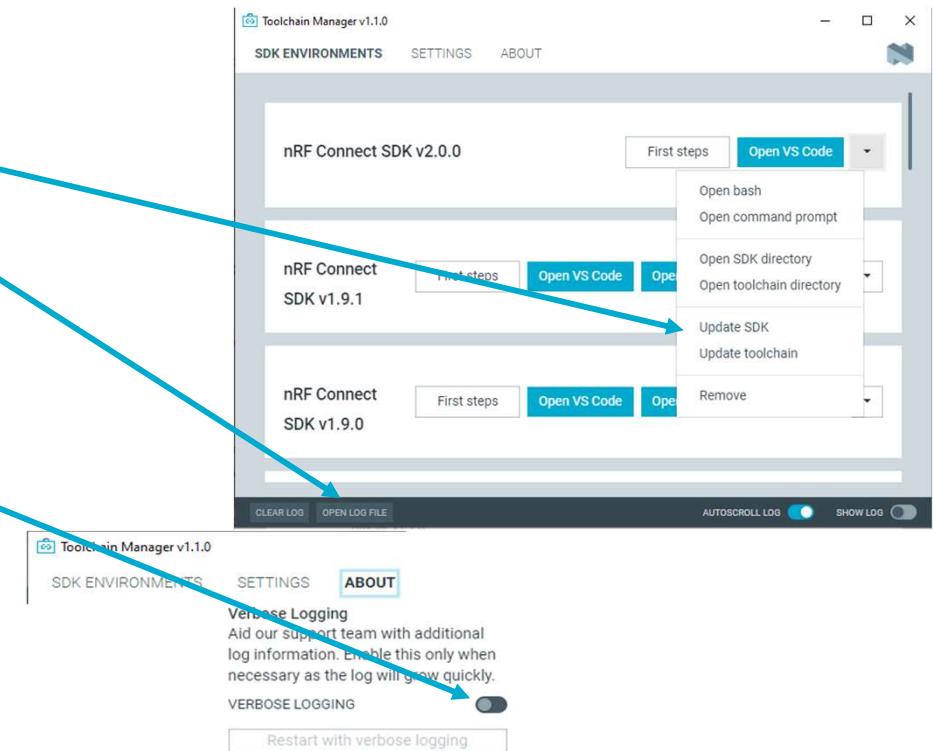
KP0

We shouldn't have the tool we don't recommend to use at the top! If we keep the sentence then it should be at the bottom with a "for the not recommended for new users" bladi bladi bladi

Kastnes, Paal, 2022-08-02T12:20:50.478

Toolchain Manager, continue

- If problems during installation:
 - Try the update functions
 - View the logfile for full details
 - Turn on Verbose Logging in the “About” menu



Slide 38

KP0

Maybe take these in different order as the verbose logging is needed to see why it fails.

Sidenote: the way it fails today where you get 0 warnings that things didn't work out, it just doesn't work afterwards is really bad.

Kastnes, Paal, 2022-08-02T12:22:31.511

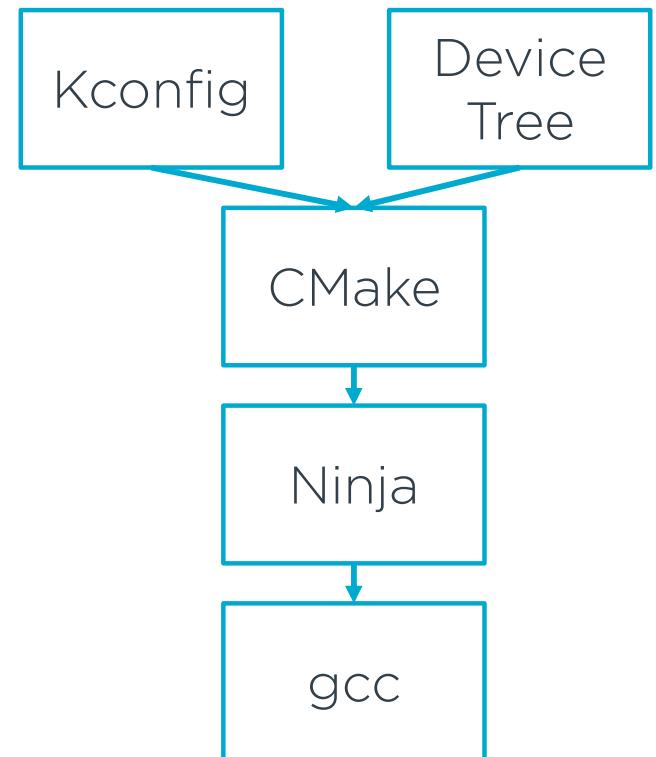
HIO 0

Verbose logging will flood with data and should be the last resort.. Suggest keeping the order as is, that will avoid the crossing arrows as well

Hanssen, Ingar, 2022-08-02T13:12:03.159

Tools for building nRF Connect SDK

- Kconfig:
 - Generates definitions to configure the system
 - Software features defined in Kconfig, enabled in prj.conf file
- DeviceTree:
 - Describes HW, pin layout
 - Allows for flexible HW mods via an overlay file
- Cmake: generates build files
- Ninja: Like make, but faster..
- Gcc: Creates executables



IDE support

- nRF Connect for Visual Studio Code
 - Installed via the Toolchain Manager
 - Built from the ground up for nRF Connect SDK
 - Highly extendable and configurable
 - CLI and GUI Interfaces
 - Create new board wizard
 - Built-in terminal
 - Rich set of [tutorial videos](#)



The screenshot shows the nRF Connect for Visual Studio Code interface. On the left, there's a sidebar with sections like 'WELCOME', 'APPLICATIONS' (listing 'AssetTracker (1)', 'Build service services', and 'BluetoothProduct (1)'), 'ASSETTRACKER' (with a tree view of files like 'Input files', 'Output files', 'SR Devices', 'Overview', 'Board', 'GPIO', 'Registers', 'Flash 1 MB', 'mcuboot 64 kB', and 'image 0 256 kB'), 'ACTIONS' (with options like 'Run', 'Priming build', 'Gathering', 'Flash', 'Debug', and 'Debug with Ozone'), and 'CONNECTED DEVICES' (listing 'NRF9160 9160 DE (NCA10009)' and its sub-devices: 'VCOM0 COMP', 'VCOM1 COMP', and 'VCOM2 COMP1'). The main area is a code editor with a C file named 'main.c' containing code related to event manager initialization and mcuboot loading. Below the code editor is a terminal window showing build logs for 'zephyr' and 'mcuboot'. At the bottom, there are tabs for 'PROBLEMS', 'OUTPUT', 'TERMINAL', 'NRF TERMINAL', and 'DEBUG CONSOLE'. A status bar at the bottom right shows 'Line 1, Col 1' and other UI elements.

```

main.c - Untitled (Workspace) - Visual Studio Code

File Edit Selection View Dev Run Terminal Help

nRF-CONNECT

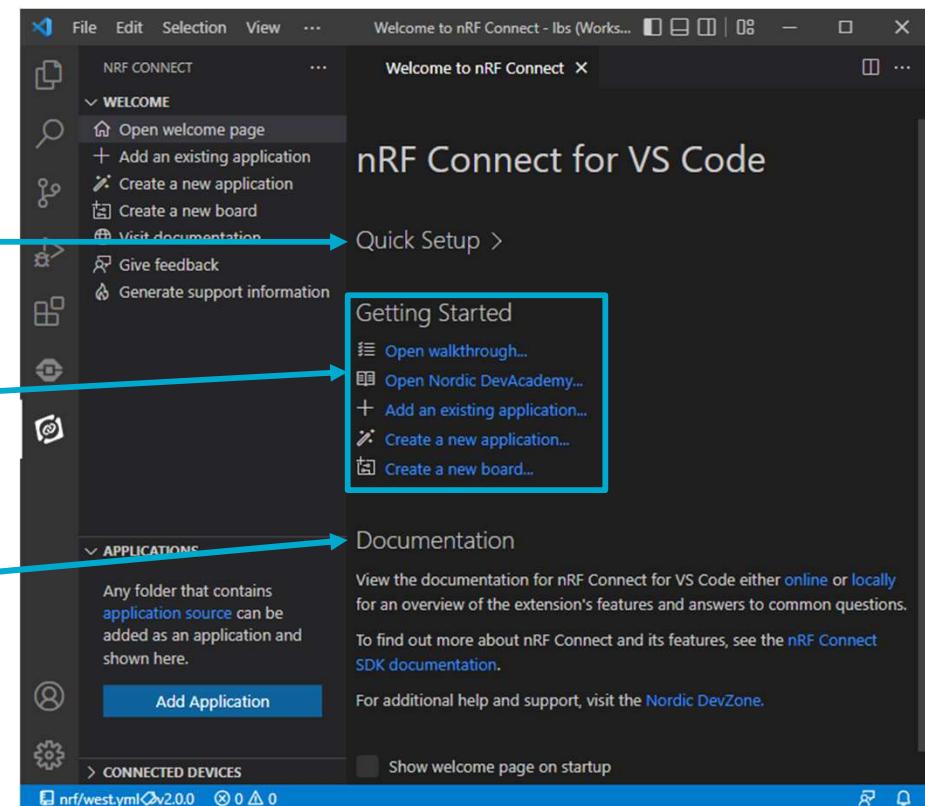
WELCOME
APPLICATIONS
  AssetTracker (1)
    Build service services
  BluetoothProduct (1)
ASSETTRACKER
  Board
  GPIO
  Registers
  Flash 1 MB
  mcuboot 64 kB
  image 0 256 kB
ACTIONS
  Run
  Priming build
  Gathering
  Flash
  Debug
  Debug with Ozone
CONNECTED DEVICES
  NRF9160 9160 DE (NCA10009)
    VCOM0 COMP
    VCOM1 COMP
    VCOM2 COMP1

main.c
main.c (1 file)
1 // 
2 // 
3 // 
4 // 
5 // 
6 // 
7 // 
8 // 
9 // 
10 // 
11 // 
12 // 
13 // 
14 // 
15 // 
16 // 
17 // 
18 // 
19 // 
20 // 
21 // 
22 // 
23 // 
24 // 
25 // 
26 // 
27 // 
28 // 
29 // 
30 // 
31 // 
32 // 
33 // 
34 // 
35 // 
36 // 
37 // 
38 // 
39 // 
40 // 
41 // 
42 // 
43 // 
44 // 
45 // 
46 // 
47 void main(void)
48 {
49     int err;
50     struct app_msg_data msg;
51 
52     handle_nrf_mcu_lib_init(&ret);
53 
54     if (event_manager_init() != 0)
55     {
56         /* Without the event manager, the application will not work
57          * as intended. A reboot is required in an attempt to recover.
58         */
59         LOG_E("Event manager could not be initialized, rebooting...");
60         k_sleep(K_SECONDS(5));
61         sys_reboot(SYS_REBOOT_COLD);
62     }
63     else
64         SHDO_PVMH((app, APP_EVT_START));
65 
66     self.thread_id = k_current_get();
67 
68     err = module_start(&self);
69 
70     if (err)
71     {
72         LOG_E("Failed starting module, error: %d", err);
73         SEND_ERROR(app, APP_EVT_ERROR, err);
74     }
75 }

PROBLEMS OUTPUT TERMINAL NRF TERMINAL DEBUG CONSOLE
[284/290] Building C object zephyr\Out\files\zephyr\final\dir\dev_handles.c.o
[285/290] Linking C executable zephyr\zephyr.elf
Memory report for zephyr\zephyr.elf:
  FLASH: 72864 B  2224 B  0.48%
  SRAM:  10102 B  17998 B  56.45%
  DRAM:   0 B   2 KB  0.00%
[286/290] Generating zephyr\mcuboot_primary.hex
[287/290] Generating zephyr\mcuboot_mcuboot_hex
[288/290] Generating zephyr\mcuboot_mcuboot_hex
[289/290] Generating zephyr\mcuboot_mcuboot_hex
[290/290] Generating zephyr\mcuboot_mcuboot_hex
powershell
nRF Connect Build...
gdb-server-2
Line 1, Col 1 Tab Size 4 Unit 8 CR/LF C AssetTrackerBuild JV Q
  
```

Setting up nRF Connect for VS Code

- The welcome page
 - Quick Setup:
 - Sets the Toolchain (and SDK) versions
 - This is valid for the whole VS Code workspace
 - Can be used to change toolchain version in existing applications
 - Getting Started:
 - Open walkthrough: A set of short videos
 - Nordic DevAcademy : More deep diving tutorials
 - Links to documentation
 - New features can be found locally under [Changelog](#)
 - Please visit the documentation site:
[nRF Connect for VS Code](#)



Slide 41

- KP0** This looks like it has already been set up as it has all the Nordic extensions. The picture and the text look a bit funk due to this
Kastnes, Paal, 2022-08-02T12:25:49.697
- HIO 0** Yes it's set up because this happened the first time you run the Toolchain Manager which sets up all the extensions. Setting up basically here mean choosing SDK version
Hanssen, Ingar, 2022-08-02T15:28:36.441
- KP0 1** And this is not the welcome page I got, this is what you get in the Nordic extension isn't it? The one that you don't find if you don't know about it....
Kastnes, Paal, 2022-08-03T08:43:09.176
- KP1** Inconsistent use of arrows, group things and one arrow per subject. One arrow from Getting started, one arrow to documentation etc.
Kastnes, Paal, 2022-08-02T12:27:04.995
- KP1 0** This hasn't been fixed
Kastnes, Paal, 2022-08-03T08:41:39.320

Setting up a new application

- 3 different ways of setting up a new application:

- + Add an existing application:

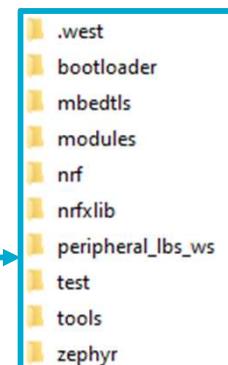
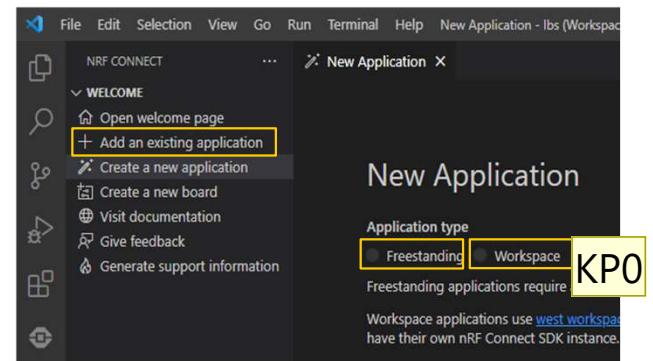
- This opens an existing project which can be a sample in the repo or a project from Segger Embedded Studio or command line
 - To use select the folder containing the file cmakelist.txt

- + Create a new application → Freestanding:

- Copies the application files (but not the SDK files) to a new folder.
 - Suitable for evaluation or multiple projects using the same SDK

- + Create a new application → Workspace

- Workspace here means a west workspace.
 - Copy the sample files and clones the SDK into the **VS Code workspace folder**.
 - Suitable for project where it is important to have a full copy of all the files in the project.

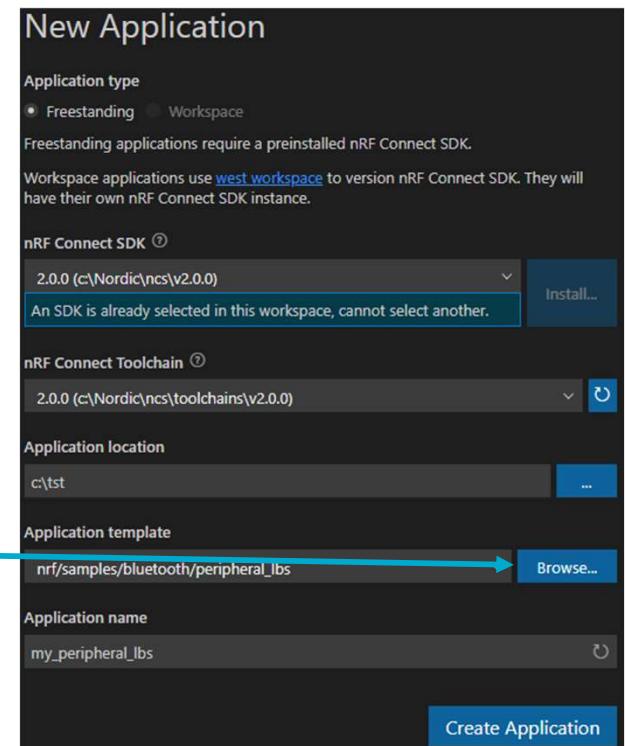


Slide 42

- KP0** This is a bit convoluted for me. For samples that are in the SDK, do we copy files and then clone the entire SDK? And "into the project" is a bit vague as we haven't introduced the concept. Isn't what happens here that we generate a new folder structure and clone the entire SDK and into this?
Kastnes, Paal, 2022-08-02T12:30:34.842
- HIO 0** "do we copy files and then clone the entire SDK": Yes
"Isn't what happens here that we generate a new folder structure and clone the entire SDK and into this?" Not quite: We copy the sample folder and close the SDK as library to the app. Description is reviewed by VS Code team
Hanssen, Ingar, 2022-08-02T15:38:31.104
- KP0 1** This still doesn't say where the cloned data is stored. I still think this is unclear....
Kastnes, Paal, 2022-08-03T08:44:55.111
- HIO 2** Structure is like this
(my project is called peripheral_lbs_ws):
VS Code workspace_folder
>.west
>bootloader
>mbedtls
>modules
>nrf
>nrfxlib
>peripheral_lbs_ws
>test
>tools
>zephyr
Hanssen, Ingar, 2022-08-11T06:49:59.187

Setting up a new application, continue

- Select the SDK and Toolchain versions
- Set App location
 - The app will be created in a subdirectory in this directory
- Select the template (aka “sample”)
 - Just start typing the sample name and it will present you a list matching what you have typed
 - Or use the sample browser
- Finally click “Create Application”



Slide 43

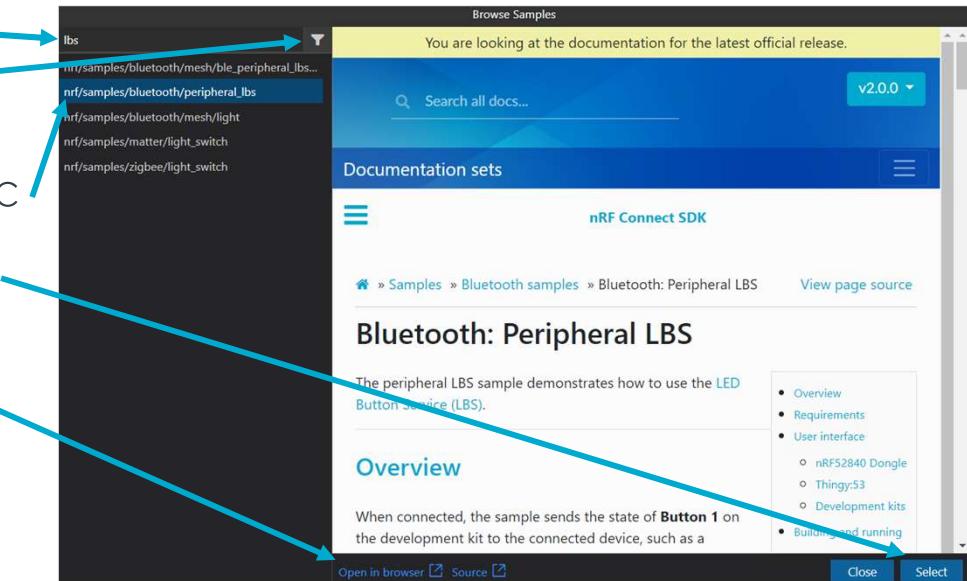
KP0

And what does this mean?

Kastnes, Paal, 2022-08-03T08:45:33.486

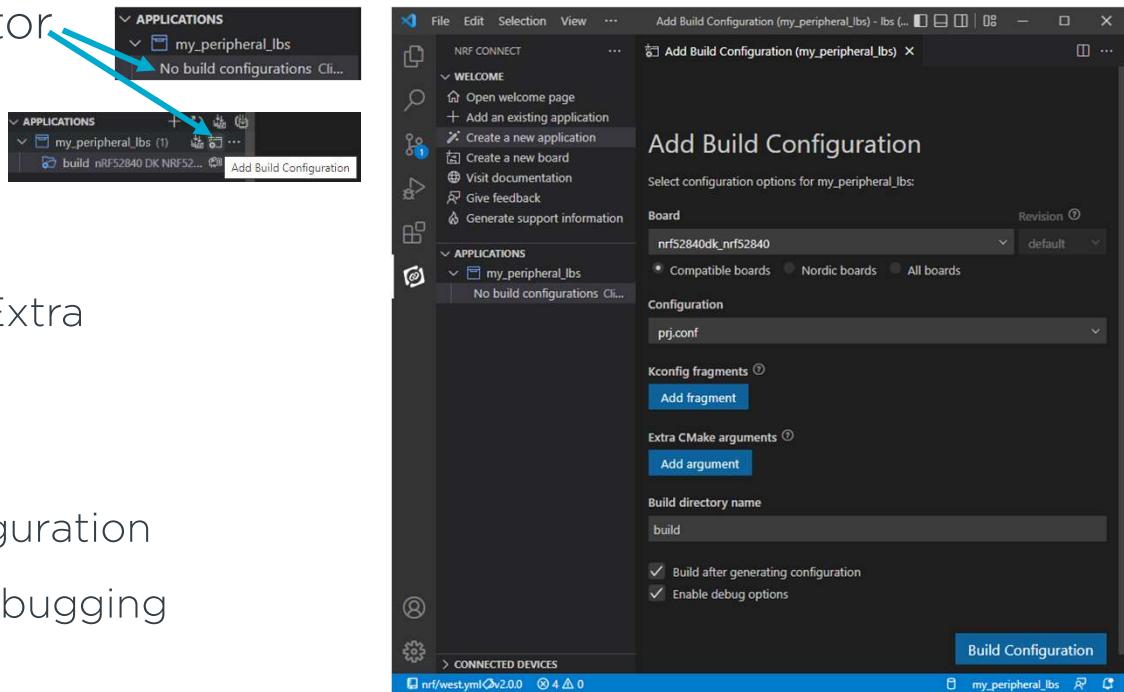
Using the sample browser

- Start typing to filter on name
- More search filters
- Click the app template to view doc
- Click “Select” to use this template
- Click to open this doc in your systems web-browser



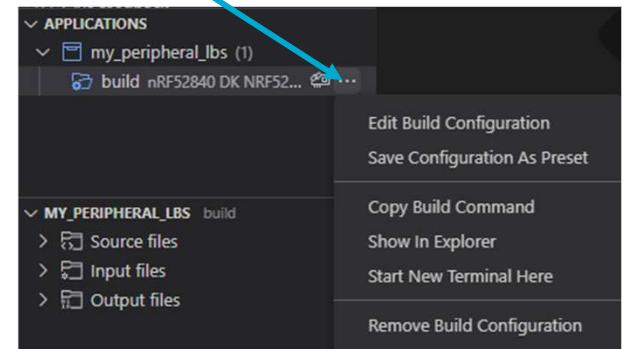
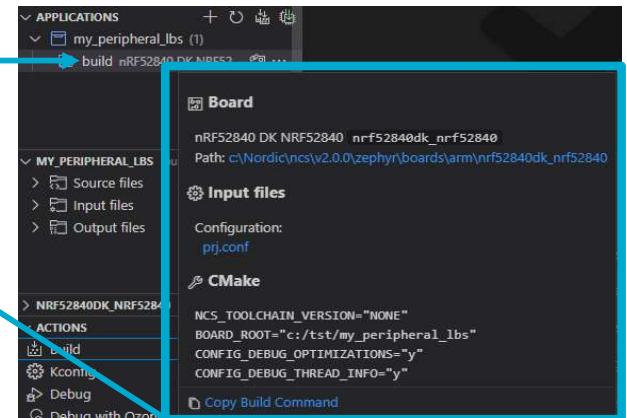
Setting up build configurations

- Click to start the configurator
 - Select your board
 - Select you conf-file from the pull-down menu
 - Add Kconfig fragments and Extra Cmake arguments if needed
 - Click  for more info
 - Select the name of this configuration
 - “Enable Debug Options” if debugging is to be used
 - And finally click 



A few nice features in build configs

- Mouse-over will show build settings
- Click the  to activate the sub-menu
 - Edit build configuration
 - Changes the build setting of this build
 - Save Configuration As Preset
 - See next slide for info on preset file
 - Copy build command:
 - Will copy the full west build command to the clipboard
 - Show in Explorer
 - Goes to the build directory in the VS Code explorer
 - Start New Terminal
 - Will open a VS Code terminal in build directory

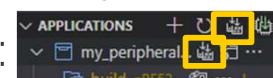


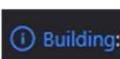
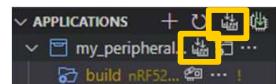
The Cmake preset file

- Contains all configuration settings
- Can be shared between machines
- Allows multiple build targets
- Using preset files with west:
 - Generic command:
 - west build -d \${targetDir} -- --preset \${presetName}
 - For our demo case where the environment is set up simply:
 - west build -- --preset build

```
1  CMakePresets.json X
2  my_peripheral_lbs > { } CMakePresets.json > [ ] configurePresets > {} 0 > {} cacheVariables
3
4  {
5      "version": 2,
6      "cmakeMinimumRequired": {
7          "major": 3,
8          "minor": 20
9      },
10     "configurePresets": [
11         {
12             "name": "build",
13             "displayName": "Build for nRF52840 DK NRF52840",
14             "generator": "Ninja",
15             "binaryDir": "${sourceDir}/build",
16             "cacheVariables": [
17                 "NCS_TOOLCHAIN_VERSION": "NONE",
18                 "BOARD": "nrf52840dk_nrf52840",
19                 "BOARD_ROOT": "${sourceDir}/",
20                 "CONFIG_DEBUG_OPTIMIZATIONS": "y",
21                 "CONFIG_DEBUG_THREAD_INFO": "y",
22                 "CONF_FILE": "${sourceDir}/prj.conf",
23                 "OVERLAY_CONFIG": "",
24                 "DTC_OVERLAY_FILE": "",
25                 "SHIELD": ""
26             ]
27         }
28     ]
29 }
```

Building your first app

- To (re)build your app click “build” in the Actions view
 - Click here to force a pristine build (a forced full build)
 - If you have several applications or build configurations, you can build all with one click:
 - Building progress can be viewed by clicking



The screenshot shows the Visual Studio Code interface with the nRF Connect extension installed. The left sidebar displays a tree view of projects and files:

- NRF CONNECT**
- WELCOME**
 - + Add an existing application
 - ✗ Create a new application
 - >Create a new board
 - Visit documentation
 - Give feedback
- APPLICATIONS**
 - my_peripheral_lbs (1)
 - build nRF52840 DK NRF52840
- MY_PERIPHERAL_LBS build**
 - > Source files
 - > Input files
 - > Output files
- > NRF52840DK_NRF52840
- ACTIONS**
 - Build
 - Kconfig
 - Debug
 - Debug with Ozone
 - Flash
- CONNECTED DEVICES**
 - > 683943804

A blue arrow points from the "Build" action in the ACTIONS section to the "Building" status in the terminal output.

The main editor area shows the `main.c` file content:

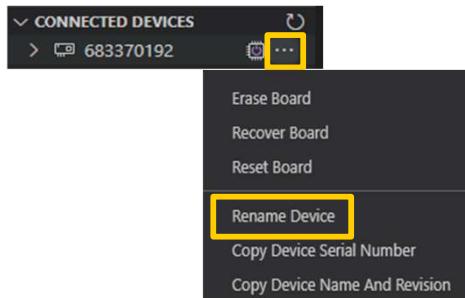
```
my_peripheral_lbs > src > C main.c > main(void)
231
232
233
234     err = bt_le_adv_start(BT_LE_ADV_CONN, ad, ARRAY_SIZE(sd));
235
236     if (err) {
237         printk("Advertising failed to start (err %d)\n", err);
238         return;
239     }
240
241     printk("Advertising successfully started\n");
242
243     for (;;) {
244         dk_set_led(RUN_STATUS_LED, (++blink_status) & K_MSEC(RUN_LED_BLINK_INTERVAL));
245     }
246 }
```

The bottom right corner shows the terminal output:

```
1.c.obj [98/232] Building C object zephyr/lib/libc/minimal/CMakeFiles/lib__libc__minimal.dir/source/stdlib/strtoll.c.obj
[99/232] Building C object zephyr/lib/libc/minimal/CMakeFiles/lib__libc__minimal.dir/source/stdlib/malloc.c.obj
[100/232] Building C object zephyr/lib/libc/minimal/CMakeFiles/lib__libc__minimal.dir/source/string/strncasecmp.c.obj
[101/232] Building C object zephyr/lib/libc/minimal/CMakeFiles/lib__libc__minimal.dir/source/string/strncpy.c.obj
Building: qsort.c.obj
```

Flashing connected boards

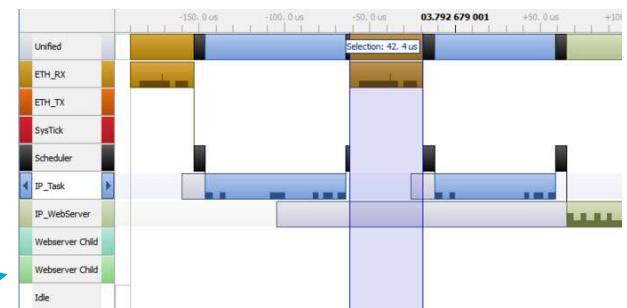
- Link your build config to a connected board
- And flash all linked boards with one click
- Flash your active build
- Give your board a name:



A screenshot of a Visual Studio code editor window titled 'main.c - Workspace'. The code is for a Zephyr application named 'my_peripheral_lbs'. The 'File Explorer' sidebar shows a project structure with 'Source files', 'Input files', and 'Output files'. The 'NRF CONNECT' sidebar shows a list of connected devices, including 'NRF52840DK_NRF52840'. In the bottom-left corner of the IDE, there is a toolbar with several icons, one of which is a blue arrow pointing to the 'Flash' button. The status bar at the bottom indicates 'Serial Port Connected (COM13)' and 'Building...'. The output window shows the build process with messages like 'Building C object zephyr/lib/libc/minimal/CMakeFiles/lib__libc_minimal.dir/source/stdlib/strtol.c.obj' and 'Building C object zephyr/lib/libc/minimal/CMakeFiles/lib__libc_minimal.dir/source/string/strncasecmp.c.obj'.

Debugging time critical code

- Debugging while running time critical apps like comms stacks:
 - Use the logging feature or “printf”:
 - [Logging – nRF Connect SDK Documentation](#)
 - NB: “printf” can be intrusive
 - Use [Segger SystemViewer](#)
 - Will visualize the program flow
 - Segger SystemViewer must be enabled in Kconfig
- Using stop-mode debugging (breakpoints) *will* break the comms link



Slide 50

KP0 Changed to point to the nRF Connect SDK, we do not link to the Zephyr project unless we are talking about Zephyr (and we are in the nRF Connect SDK presentation....)
Kastnes, Paal, 2022-08-03T08:55:09.087

RTOS aware debugging

- Start your debug session in the Actions pane
- Controls to operate the debugger
- Toggle breakpoints by clicking
 - Hint: Right-click on breakpoint for “smart” breakpoints
 - Use GDB console directly here
- NB: Remember to build for debugging...

The screenshot shows the Visual Studio Code interface during a debug session. The top navigation bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar indicates "main.c - lbs (Workspace) - Visual Studio Code".

The left sidebar features an "ACTIONS" pane with options: Build, Kconfig, Debug (which is highlighted with a blue box), and Debug with Ozone. Below it is a "VARIABLES" pane showing Local variables: blink_status (value 1) and err (<optimized out>). There's also a WATCH section with a breakpoint at line 244. The "CALL STACK" pane lists threads: main RUNNING PRIO 15, bg_thread_main PENDING PRIO 65526, z_thread_entry PENDING PRIO 65525, arch_switch_to_main_thread PENDING PRIO 10, and idle UNKNOWN PRIO 15. The "BREAKPOINTS" section shows a red dot next to main.c:244, which is highlighted with a blue box. The "COREX PERIPHERALS" and "COREX REGISTERS" sections are also visible.

The main editor area displays the "main.c" file with code related to Bluetooth advertising and LED control. The DEBUG CONSOLE at the bottom shows the output of the debugger, including the hit breakpoint message:

```
Thread 2 hit Breakpoint 4, main () at ..../src/main.c:244
244 dk_set_led(RUN_STATUS_LED, (++blink_status)
% 2);
```

The status bar at the bottom shows "nrf/westyml v2.0.0" and "Serial Port Connected (COM13)".

RTOS aware debugging, continue

- Variables:
 - Add watch by right-click variable
 - When paused view value by mouseover
- RTOS in Call Stack View:
 - Thread name and status
 - Thread call stack when paused
- MCU peripheral's and register views

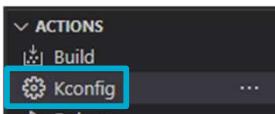
```

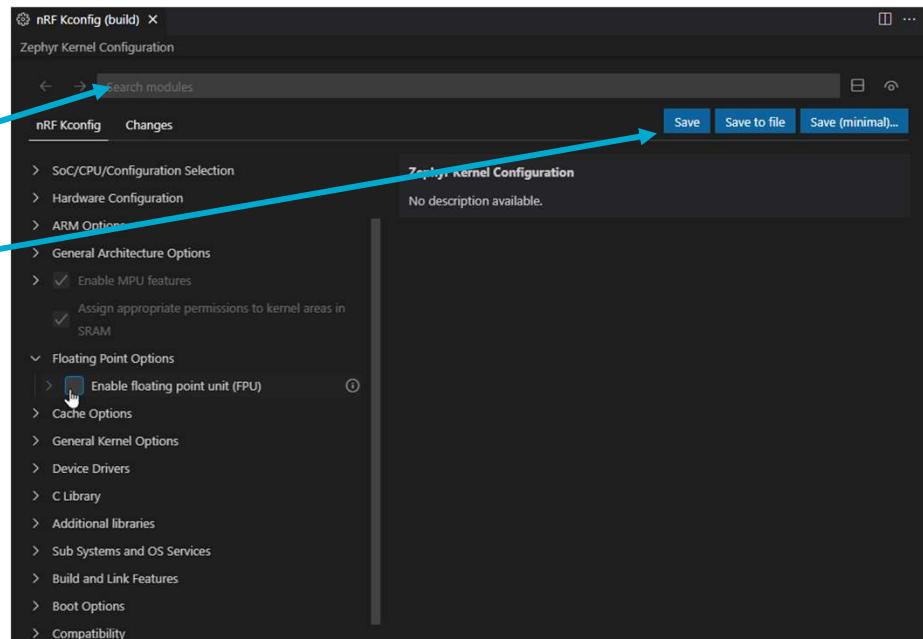
my_peripheral_lbs > src > C main.c > main(void)
233
234 err = bt_le_adv_start(BT_LE_ADV_CONN, ad, ARRAY_SIZE(
235 sd, ARRAY_SIZE(sd));
236 if (err) {
237   printk("Advertising failed to start (err %d)\n"
238   return;
239 }
240
241 printk("Advertising successfully started\n");
242
243 for (;;) {
244   dk_set_led(RUN_STATUS_LED, (++blink_status) % 2
245   k_sleep(K_MSEC(RUN_LED_BLINK_INTERVAL));
246 }
247
248

main.c - lbs (Workspace) - Visual Studio Code | File Edit Selection View Go Run Terminal Help
File Edit Selection View Go Run Terminal Help
RUN AND DE... No Configuration ...
CALL STACK
PAUSED ON BREAKPOINT
main@0x000108ac ./src/main.c:244
bg_thread_main@0x0002237c C:/No...
z_thread_entry@0x000294ea C:/No...
arch_switch_to_main_thread@0x000138f
??@0x5182f030 Unknown Source 0
idle UNKNOWN PRIO 15 PAUSED
MPSL Work PENDING PRIO 65526 PAUSED
syswqk PENDING PRIO 65535 PAUSED
BT TX PENDING PRIO 65527 PAUSED
BT RX PENDING PRIO 65528 PAUSED
BREAKPOINTS
main.c SRC 244
Cortex PERIPHERALS
Cortex REGISTERS
DEBUG CONSOLE ... Filter (e.g. text, /exclude)
% 2;
Thread 2 hit Breakpoint 4, main () at ..../src/main.c:244
244          dk_set_led(RUN_STATUS_LED, (++blink_status)
% 2);

```

Configuring using nRF Kconfig

- Start nRF Kconfig from Actions
- nRF Kconfig features:
 - Fuzzy search box
 - 3 save options:
 - Save saves temporary changes to the <build>/zephyr/.conf file (a pristine build will revert these changes).
 - Save to file saves changes to an existing .conf file.
 - Save (minimal) saves the smallest subset of settings to a new .conf file which can be used as an Kconfig fragment file



Using nRF Kconfig, continue

← → Navigation buttons

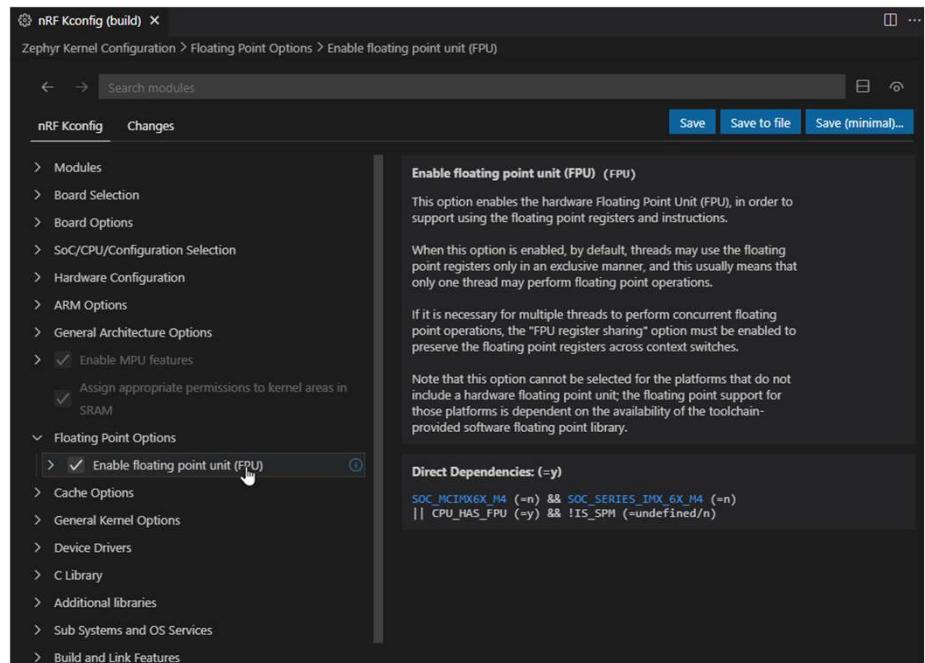
Jump to item in the tree

ⓘ View information

- Shows dependencies with clickable links:

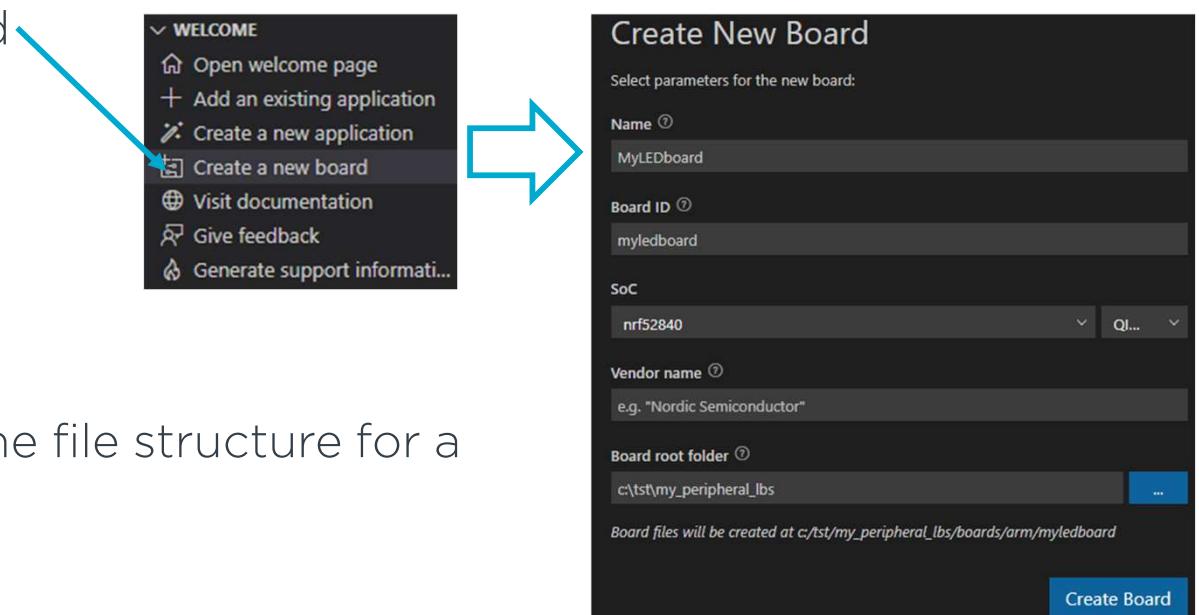
```
Direct Dependencies: (=y)
SOC_MCIMX6X_M4 (=n) && SOC_SERIES_IMX_6X_M4 (=n)
|| CPU_HAS_FPU (=y) && !IS_SPM (=n)
```

- nRF Kconfig help (online)
- Kconfig language help (online)



Defining your own board

- Click to create the board
- Follow the guide and mouseover ⓘ for help
- Complete by clicking **Create Board**
- You have now created the file structure for a new board



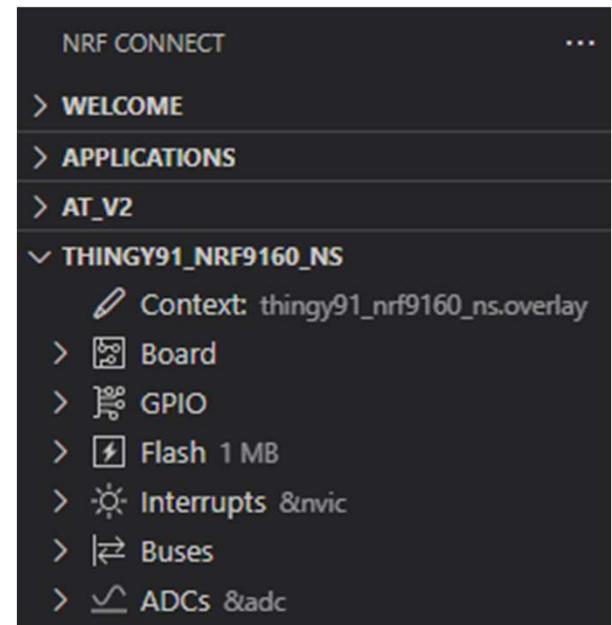
DeviceTree

- DeviceTree is a system and language for defining the hardware layout of a custom board in a hierarchical tree structure. In a typical scenario, a developer will have several different devices, such as LEDs, a display, and sensors, all of which need to be configured. This configuration is done in the DeviceTree language
- [Introduction to DeviceTree \(online\)](#)
- Video on Device drivers, covers DeviceTree as well: [watch on YouTube](#)

nRF DeviceTree

■ DeviceTree contexts:

- DeviceTree contexts consist of a board file and a list of overlay files. Each context corresponds to a single compiled DeviceTree file that goes into a build.
- If you work with more than one application or board, you will have multiple sets of DeviceTree contexts - one for each of your builds. Every time you open a new DeviceTree file, the extension will add a DeviceTree context unless this file is already part of an existing context
- The DeviceTree contexts show up in the DeviceTree View in the Sidebar (this view will be named after your board once this is set up)
- Each DeviceTree context presents an overview of common resources and their users. Each entry in the overview is linked with a specific node or property in the DeviceTree. Clicking one of these will go to the primary definition to the linked node or property.



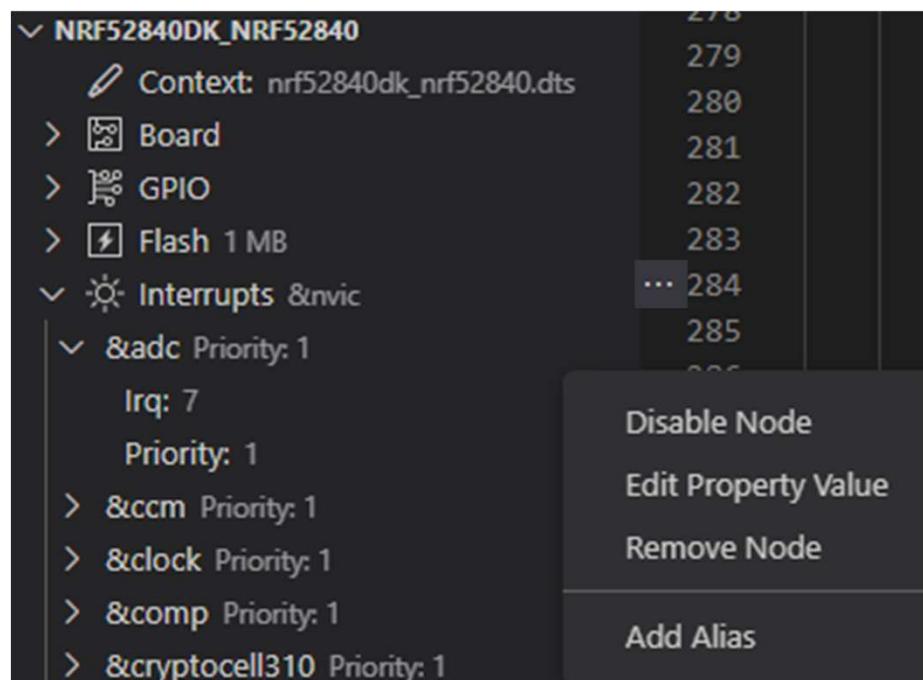
Slide 57

- KP0** And what is a context????? May be clear to a DTS or VSC expert but makes no sense to me.
Kastnes, Paal, 2022-08-03T09:00:48.145
- HIO 0** "DeviceTree contexts consist of a board file and a list of overlay files. Each context corresponds to a single compiled DeviceTree file that goes into a build."
Hanssen, Ingar, 2022-08-10T11:27:01.318
- KP0 1** Still not clear at all, where is the link to the picture on the right? What contexts do you have here? Are each of the items in the picture a context?
Again, not clear and pointing to the text doesn't make it any clearer
Kastnes, Paal, 2022-08-10T11:36:59.325
- HIO 2** Changed the pic to include the context info for the thingy91 non-secure. OK now?
Hanssen, Ingar, 2022-08-12T07:47:44.643
- BA1** [@Hanssen, Ingar] is this the name of the VS Code extension?
Bergerud, Aina, 2022-08-12T07:30:32.799
- HI1 0** *This* extension is called "nRF DeviceTree", its a part of the extension bundle consisting of 4 extensions made by us. But it can also be used standalone with the zephyr project (we don't say anything about that)
Hanssen, Ingar, 2022-08-12T07:41:35.857
- BA1 1** Ok
Bergerud, Aina, 2022-08-12T08:02:30.215

Using the nRF DeviceTree view

■ Aliases:

- Within the contexts, you can create a DeviceTree alias on a selected node if applicable. To do this:
 - Select the node you want to create an alias for.
 - Click on the More actions... menu icon.
 - Click on Add Alias.



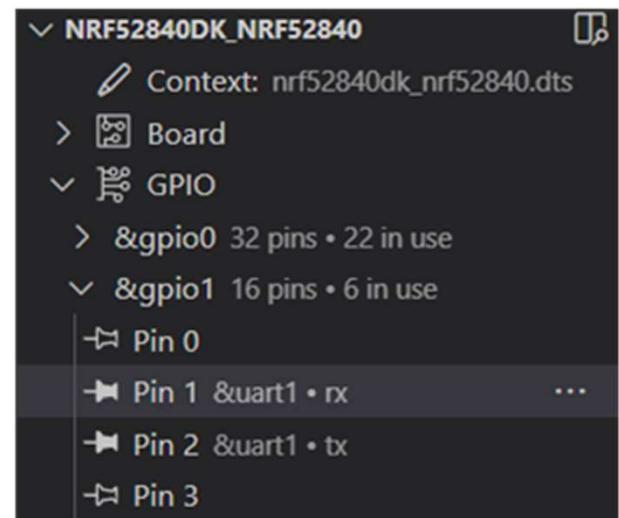
The nRF DeviceTree view, continue

■ Board

- This is a list of information related to your connected board. It includes the name, architecture, supported features, and supported toolchains

■ GPIO

- This is a list of all known GPIO controllers, which are determined by the gpio-controller property. Each GPIO controller presents a list of the allocated pins and their owners inferred from gpios properties, -pins properties, and pinctrl properties.
- The pin icon to the left of each GPIO pin indicates whether a pin is connected. A filled-in icon indicates a connected pin, and an unfilled-in icon indicates an unconnected pin.
- The GPIO context also lets you copy the associated C identifier of a given node using the action Copy C Identifier of Node.

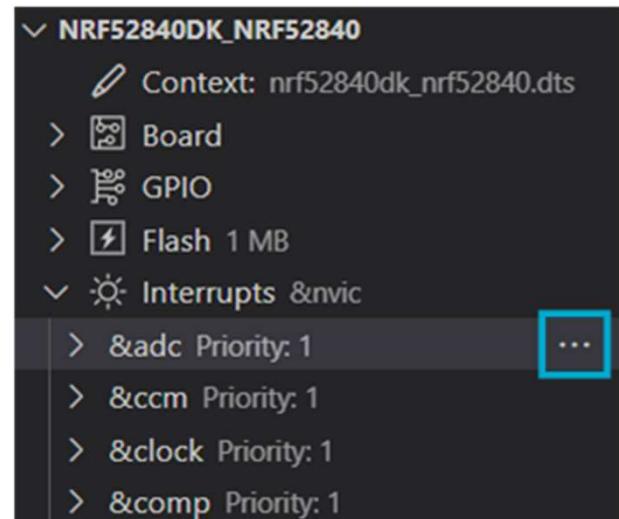


Slide 59

- KP0** What is a GPIO Controller?
Kastnes, Paal, 2022-08-03T09:02:17.042
- H10 0** Equivalent to a "port"; in this case 32 pins on gpio0 and 16 pins on gpio1
Hanssen, Ingar, 2022-08-10T11:28:29.269
- KP1** Connected to what? Or does this imply it is being assigned a function by the code? Note there will be HW people seeing this presentation and for them connected has a HW implication.
Kastnes, Paal, 2022-08-03T09:03:28.035
- HI1 0** If it's connected to a "function" like (uart1 rx) it will be indicated here. If available (not connected) connected the the pin icon will not be filled and there will be no text next to the "Pin <number>"
Hanssen, Ingar, 2022-08-10T11:31:18.546

The nRF DeviceTree view, continue

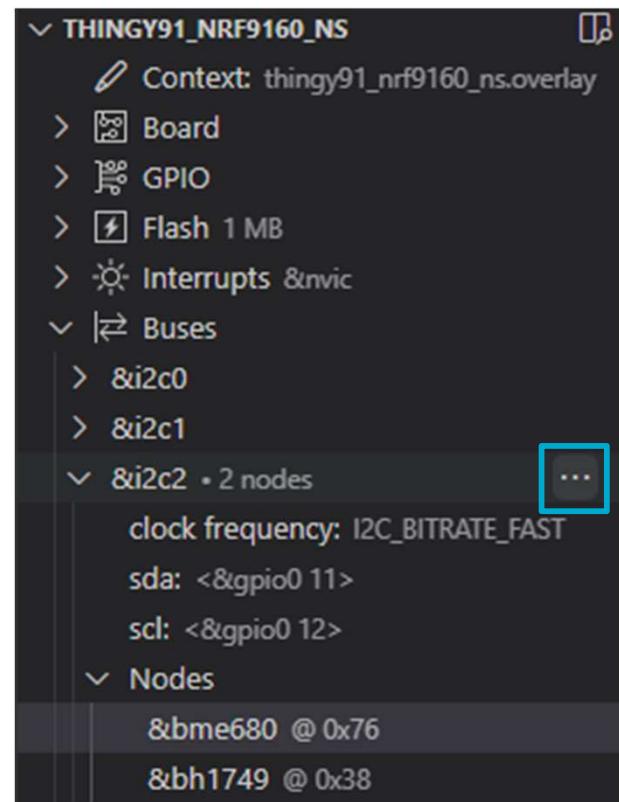
- Flash
 - This is a list of all flash controllers, or nodes, based on the soc-nv-flash type binding.
- Interrupts
 - This is a list of all interrupt controllers determined by the interrupt-controller property. It lists the allocated interrupts on the controller and their users, as well as any other available information, such as their priority and index.
 - When hovering over a node, the More actions... menu icon appears to the right. Click on it to access the following actions:
 - Edit Property Value - edit the value of the selected node in the target DeviceTree file.
 - 1. Disable Node or Enable Node - disable or enable the selected node.
 - 2. Remove node - remove the selected node from the target DeviceTree file.



The nRF DeviceTree view, continue

■ Buses

- This is a list of all known buses on the device determined by the bus entry in the node's type binding. It lists important properties of the bus, as well each node on the bus and their address if the bus has an address space. If the bus is an SPI bus, the chip select configuration of each node is also listed if it is known.
 - When hovering over a node, you can see a description of the node's binding. You can also see the More actions... menu icon to the right:
 - Enable Node or Disable Node - opens the .overlay file in the editor and sets status = "disabled" or "enabled".
1. Add Node to Bus - opens a drop-down list of available nodes to add to your board. After choosing a node, the .overlay file opens in the editor and the node information is added.
 2. Remove Node - removes the selected node from the .overlay file.
- The Buses context also lets you copy the associated C identifier of a given node using the action Copy C Identifier of Node.



Slide 61

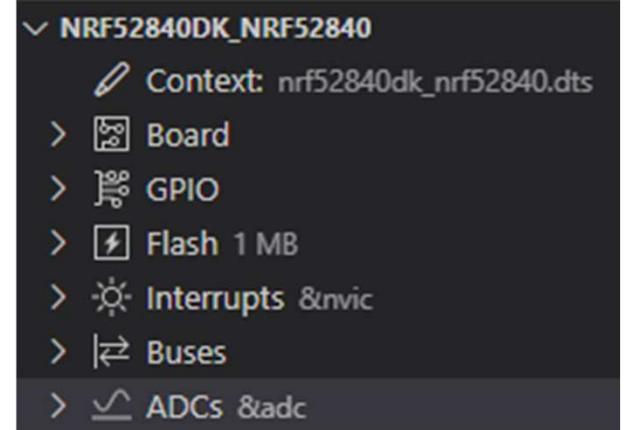
KPO Maybe have an example of a node present on DK here, now it is a but non-connected
Kastnes, Paal, 2022-08-03T09:05:32.763

HIO 0 Changed pic to show thingy91 which has 2 nodes connected to i2c #2
Hanssen, Ingar, 2022-08-11T06:29:14.803

The nRF DeviceTree view, continue

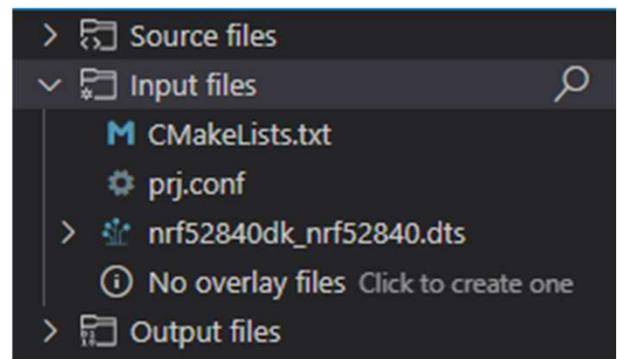
■ ADC

- This is a list of all ADC controllers on the device, or nodes, based on the adc-controller type binding. Each ADC controller contains a list of all allocated channels based on references made to the ADC instances using the io-channels property.



Using nRF DeviceTree, overlay files

- DeviceTree overlay files:
 - Use to provide additional board functionality:
 - Enabling a peripheral that is disabled by default
 - Selecting a sensor on the board for an application specific purpose.
 - Adds application-specific modifications to the <board>.dts file
 - The build system concatenates the changes from the overlay files with the base DeviceTree file.



Slide 63

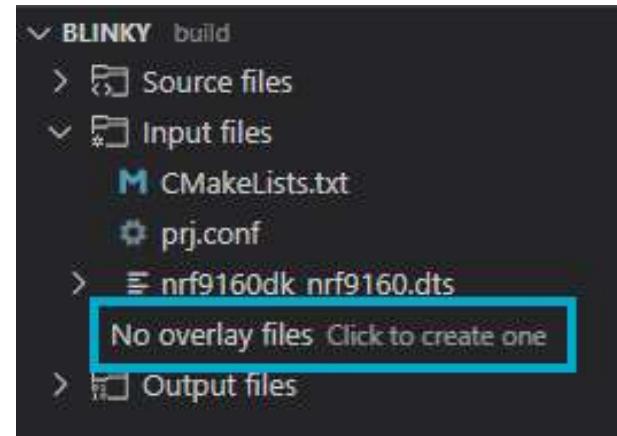
KP0

A lot of text, can we bulletify this?

Kastnes, Paal, 2022-08-03T09:06:18.154

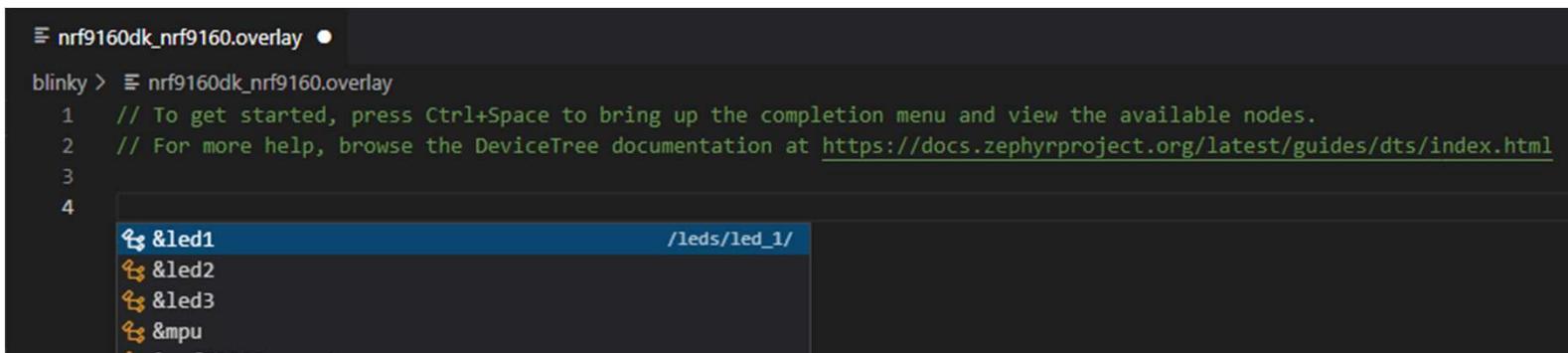
nRF DeviceTree, creating overlays

- Click on the Input files. It will say "No overlay files. Click to create one". Click this on.
- The generated .overlay file appears under Input files.



nRF DeviceTree, creating overlays

- Adding a node to the overlay file
 - With the overlay file created, we can start adding nodes to the board:
 - Press Ctrl+Space to start the auto-completion feature and see a list of available nodes.
For detailed information about writing property values, see the [nRF Connect SDK documentation](#).



```
nrf9160dk_nrf9160.overlay
blinky > nrf9160dk_nrf9160.overlay
1 // To get started, press Ctrl+Space to bring up the completion menu and view the available nodes.
2 // For more help, browse the DeviceTree documentation at https://docs.zephyrproject.org/latest/guides/dts/index.html
3
4
&led1
&led2
&led3
&mpu
&uart
```

- For an implementation example see: [Creating an overlay file - nRF Connect for VS Code](#)

Slide 65

KPO Do we ever describe how to add nodes? Not necessarily something to cover in the presentation but the trainer should know how this is done
Kastnes, Paal, 2022-08-03T09:08:20.281

HIO 0 Not sure.. This is NCS doc issue I think
Hanssen, Ingar, 2022-08-04T11:13:37.541

nRF DeviceTree language support

- Code completion:
 - Code completion will suggest as you type
 - Or press Ctrl+Space even before typing anything
- Mouse-over will display info

```

182 arduino_i2c: i2c@40003000 {
183   #address-cells = < 0x01 >;
184   #size-cells = < 0x01 >; #size-cells
185   reg = < 0x40003000 0x1000 >; reg
186   clock-frequency = < 0x186a0 >;
187   interrupts = < 0x03 0x01 >;
188   status = "okay";
189   label = "I2C_0";
190   compatible = "nordic,nrf-twi";
191   sda-pin = < 0x1a >;
192   scl-pin = < 0x1b >;
193 };
194
195 i2c1: i2c@40004000 {
196   #address-cells = < 0x01 >;
197   #size-cells = < 0x00 >; #size-cells
198   reg = < 0x40004000 0x1000 >; reg
199   clock-frequency = < 0x186a0 >;
200   interrupts = < 0x04 0x01 >;
201   status = "disabled";
202   label = "I2C_1";
203   compatible = "nordic,nrf-twi";
204   sda-pin = < 0x1e >;
205   scl-pin = < 0x1f >;
206 };

```

The #size-cells property defines the number of u32 cells used to encode the size field in a child node's reg property.

The #address-cells and #size-cells properties are not inherited from ancestors in the devicetree. They shall be explicitly defined.

A DTSpec-compliant boot program shall supply #address-cells and #size-cells on all nodes that

nRF DeviceTree language, continue

- Add missing required properties:
 - A button with a light bulb icon appears next to the selected node in the Editor.
 - To add the missing required property, click on this icon + Add required properties
 - Fill inn the missing data

```
KP0
9    pinctrl-0 = <&spi0_default>;
10   pinctrl-1 = <&spi0_sleep>;
11   pinctrl-names = "default", "sleep";
12 };
13
14 &spi1 {
15     compatible = "nordic,nrf-spi";
16     status = "okay";
17     pinctrl-0 = <&spi1_default>;
18     pinctrl-1 = <&spi1_sleep>;
19     pinctrl-names = "default", "sleep";
20 };
21
22 bme280@0 {
23     compatible = "bosch,bme280";
24     reg = <0x0 >;
25 };
26
27 arduino_spi: &spi2 {
```

Slide 67

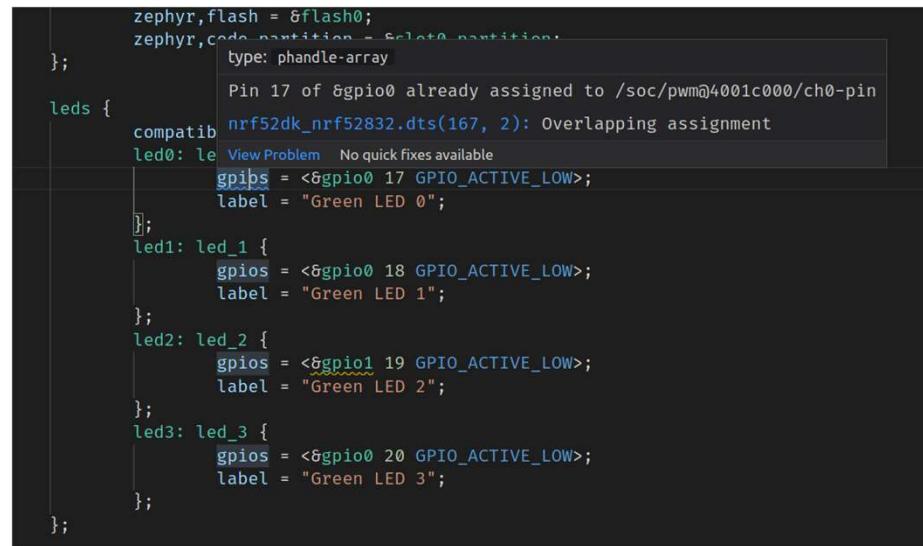
KP0

Bulletify

Kastnes, Paal, 2022-08-03T09:10:42.008

nRF DeviceTree language, continue

- Syntax validation and highlighting
 - Built-in syntax checking
 - Validates the tree structure
 - points out potential issues in the tree
(such as overlapping assignments or missing references)



The screenshot shows a code editor displaying DeviceTree source code. The code defines a 'leds' node with three child nodes: 'led0', 'led1', and 'led2'. Each child node has a 'gpios' property and a 'label' property. The 'gpios' property for 'led0' is set to '&gpio0 17 GPIO_ACTIVE_LOW'. The 'gpios' property for 'led1' is set to '&gpio0 18 GPIO_ACTIVE_LOW'. The 'gpios' property for 'led2' is set to '&gpio1 19 GPIO_ACTIVE_LOW'. The 'label' property for 'led0' is 'Green LED 0'. The 'label' property for 'led1' is 'Green LED 1'. The 'label' property for 'led2' is 'Green LED 2'. The code editor uses color coding for different parts of the code: green for comments, blue for keywords like 'zephyr', 'code', 'partition', 'type', 'compatib', 'led', 'led0', 'led1', 'led2', 'led3', 'gpios', and 'label', and grey for the rest of the text.

```
zephyr,flash = &flash0;
zephyr,code_partition -> fslc0_partition;
    type: phandle-array
Pin 17 of &gpio0 already assigned to /soc/pwm@4001c000/ch0-pin
compatibl nrf52dk_nrf52832.dts(167, 2): Overlapping assignment
led0: le View Problem No quick fixes available
    gpios = <&gpio0 17 GPIO_ACTIVE_LOW>;
    label = "Green LED 0";
};
led1: led_1 {
    gpios = <&gpio0 18 GPIO_ACTIVE_LOW>;
    label = "Green LED 1";
};
led2: led_2 {
    gpios = <&gpio1 19 GPIO_ACTIVE_LOW>;
    label = "Green LED 2";
};
led3: led_3 {
    gpios = <&gpio0 20 GPIO_ACTIVE_LOW>;
    label = "Green LED 3";
};
```

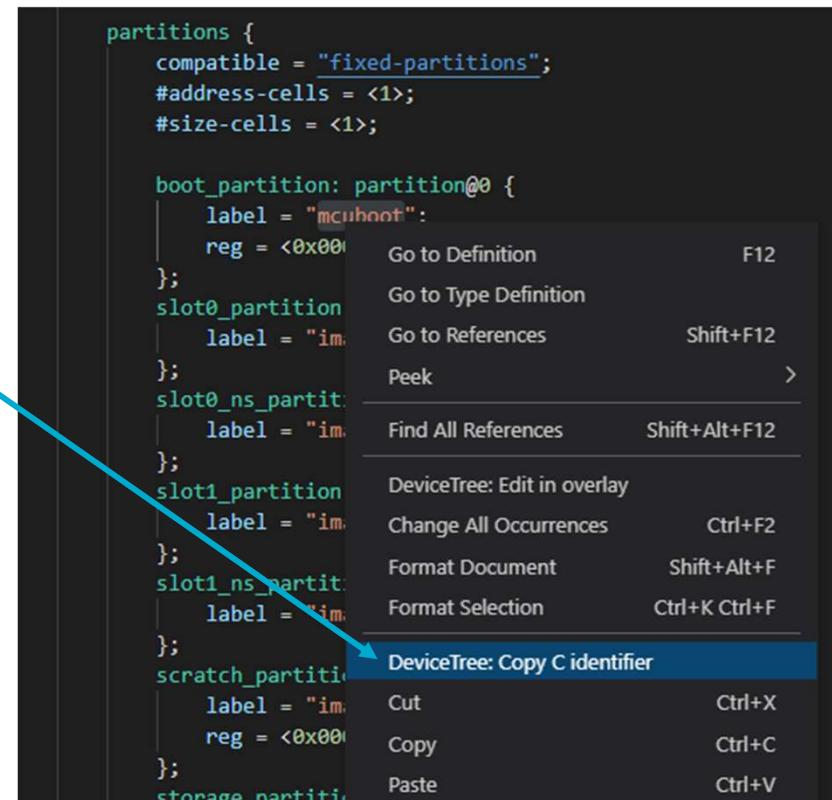
nRF DeviceTree language, continue

- Copy C identifiers
 - References to a can be copied
 - And pasted into C code
 - Right-click and choose: Copy C identifier
 - If the selected symbol has a corresponding C macro, like DT_PROP(DT_NODELABELadc), label), it will be copied to the clipboard.

```
    prescaler = <0>;
    label = "TIMER_3";
};
```

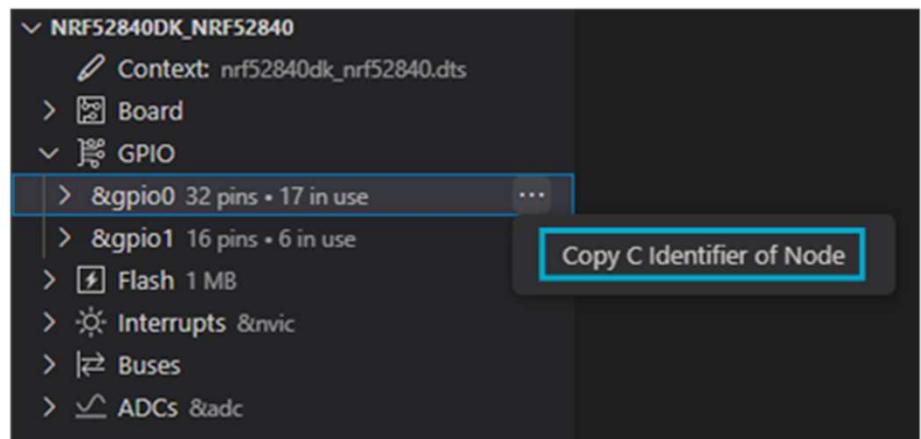
⑧ 3 △ 3 ① 5 Copied "DT_PROP(DT_NODELABEL(rtcl), clock_frequency)" to clipboard

- A message shows up in the Status Bar with the copied string:



nRF DeviceTree language, continue

- Copy C identifiers from a DeviceTree Node
 - To copy from a DeviceTree node:
 - Expand the appropriate section in the DeviceTree View.
 - Hover your mouse over the node you want to copy.
 - Click on the More actions  menu icon.
 - Click on Copy C Identifier of Node.



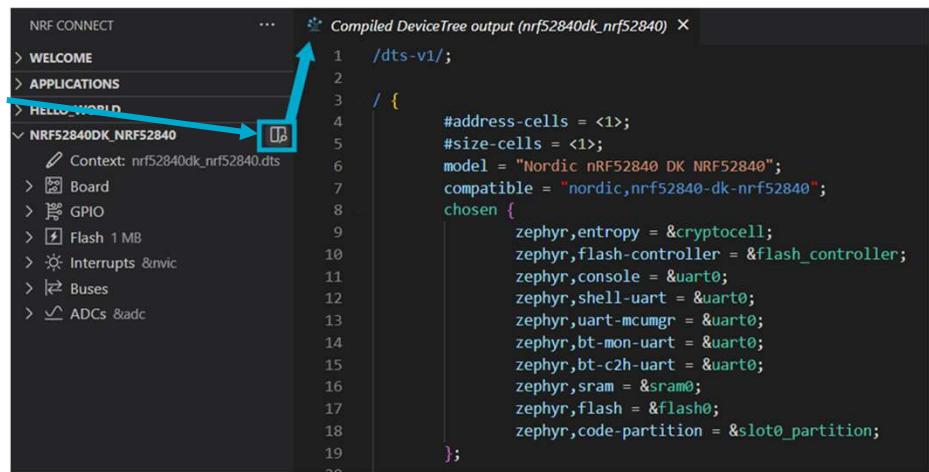
nRF DeviceTree language, continue

- Use in C-files:
 - C file macro argument completion
 - Uses data from the most recent DeviceTree
 - Provide arguments completion for:
 - DT_ALIAS
 - DT_NODELABEL
 - DT_PATH
 - DT_CHOSEN

```
14  /* The devicetree node identifier for the "led0" alias. */
15  #define LED0_NODE DT_ALIAS()
16
17  #if DT_NODE_HAS_STATUS(LED0)
18  #define LED0    DT_GPIO_LABEL led0
19  #define PIN     DT_GPIO_PIN    led0
20  #define FLAGS   DT_GPIO_FLAGS
21
22  #else
23  /* A build error here means the board does not have a
24  #error "Unsupported board: "
25  #define LED0    ""
26  #define PIN     0
27  #define FLAGS   0
28
29
```

nRF DeviceTree language, continue

- Show compiled output:
 - Click to view the compiled DeviceTree
 - Opens in separate window.
 - Click any context item in the DeviceTree View and the compiled output will jump to that section in the code



```
NRF CONNECT ... Compiled DeviceTree output (nrf52840dk_nrf52840) ×  
WELCOME /dts-v1/;  
APPLICATIONS / {  
HELLO_WORLD #address-cells = <1>;  
NRF52840DK_NRF52840 #size-cells = <1>;  
Context: nrf52840dk_nrf52840.dts model = "Nordic nRF52840 DK NRF52840";  
Board compatible = "nordic,nrf52840-dk-nrf52840";  
GPIO chosen {  
Flash 1 MB zephyr,entropy = &cryptocell;  
1 MB #size-cells = <1>; zephyr,flash-controller = &flash_controller;  
Interrupts &nvic zephyr,console = &uart0;  
Buses zephyr,shell-uart = &uart0;  
ADCs &adc zephyr,uart-mcumgr = &uart0;  
}; zephyr,uart-mon-uart = &uart0;  
zephyr,bt-c2h-uart = &uart0;  
zephyr,sram = &sram0;  
zephyr,flash = &flash0;  
zephyr,code-partition = &slot0_partition;
```

Slide 72

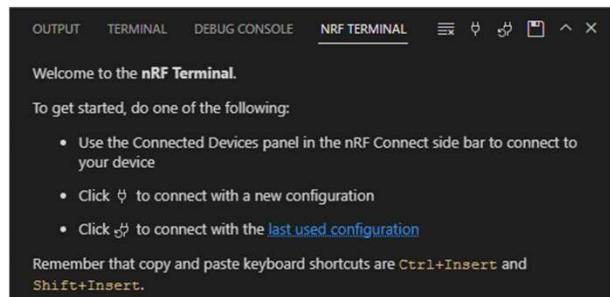
KP0

Bulletify

Kastnes, Paal, 2022-08-03T09:12:12.652

nRF Terminal

- Integrates with nRF Connect for VS Code
- Connects over serial ports or RTT.
- Uses Terminal Panel of the VS Code interface.
- Reconnect automatically (but not for RTT).
- This is a “simple” terminal.
- If more advanced features are needed: Use a different terminal program
- Full documentation for nRF Terminal can be found [here](#)



Slide 73

KPO **Bulletify**

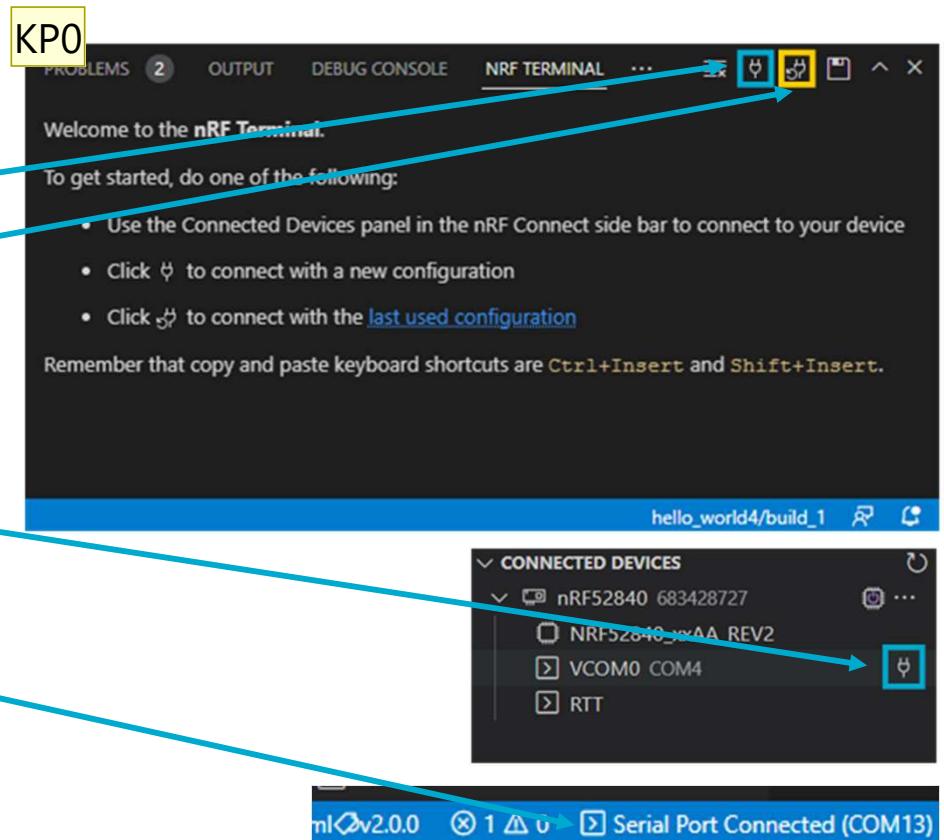
Kastnes, Paal, 2022-08-03T09:12:36.368

KPO 0 **Fix this!!!!**

Kastnes, Paal, 2022-08-03T09:14:24.676

nRF Terminal, connecting a device

- There are two way to connect:
 - From the Terminal title bar:
 - Start Terminal with new configuration
 - Start Terminal with previous configuration
 - From the Connected Devices View:
 - Click on the board you want to connect to.
 - A list of options appear.
 - Hover your mouse over the option that has a terminal icon and click on the plug icon.
- Disconnecting:
 - Click the terminal info in the status bar



Slide 74

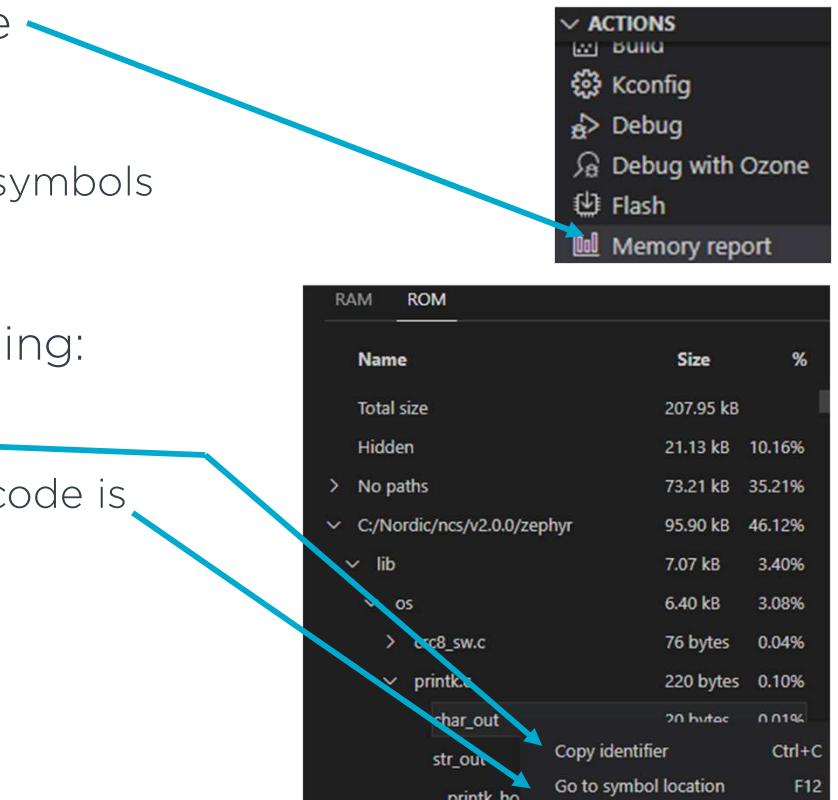
KP0

Fix this!!!!

Kastnes, Paal, 2022-08-03T09:17:03.790

The memory report

- A memory report can be generated here
- Displays
 - A hierarchical list of RAM and Flash (ROM) symbols
 - Memory usage pr symbol
- Each entry has right-click feature providing:
 - An identifier
 - If this is a code symbol a link to the source code is provided



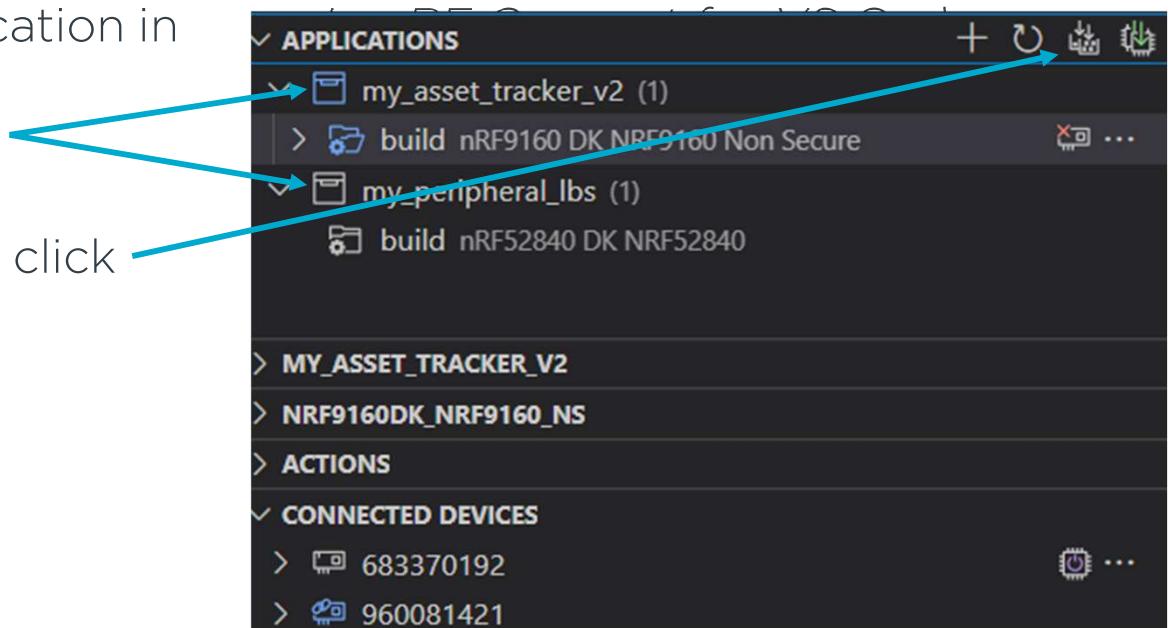


nRF Connect for VS Code

Apps, configs and tips & tricks

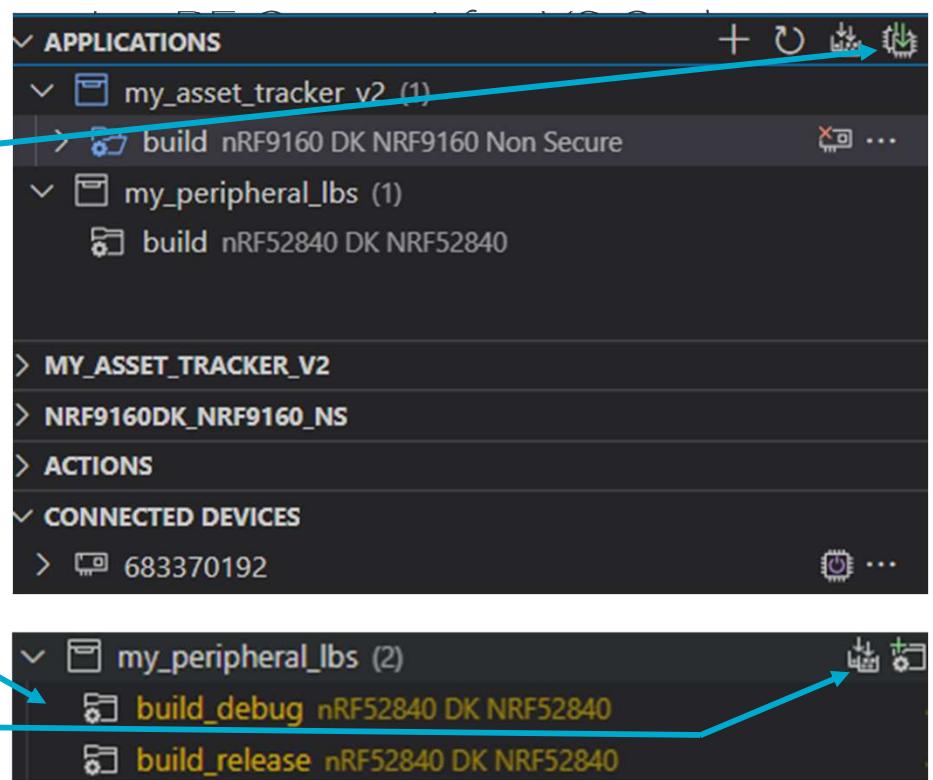
Multiple apps, configs and devices

- You can have many application in your VS Code workspace (2 shown)
- Build all of them with one click



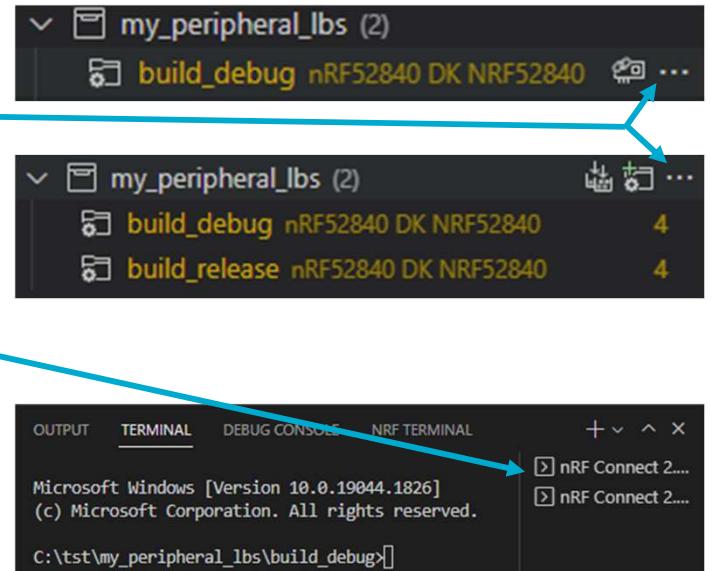
Multiple apps, configs and devices

- You can link a build configuration to a device and flash all linked devices with one click
- You can have many configurations for the same application, for instance release and debug and build all configurations for one app with one click



Using CLI with VS Code

- VS Code refer to CLI as “Terminals”
- You can open a CLI by clicking 
- The terminal will get the correct settings for the configuration
- You can have many terminals
 - Each terminal may have different settings



Editing tips and tricks

- Pop-up documentation:
 - Mouse-over the text of interest
 - The API help or other info will pop up

```
206     return;
207 }
208
209 err = bt_conn_auth_info_cb_register(&conn_auth_info_callbacks);
210 if (err) {
211     printk("Failed to register authorization info callbacks.\n");
212     return;
213 }
214 }
215
216 err = bt_enable(NULL);
```

```
206
207
208
209
210
211
212
213
214
215
216
```

int bt_enable(bt_ready_cb_t cb)

Enable Bluetooth Enable Bluetooth. Must be the called before any calls that require communication with the local Bluetooth hardware. When

Parameters:

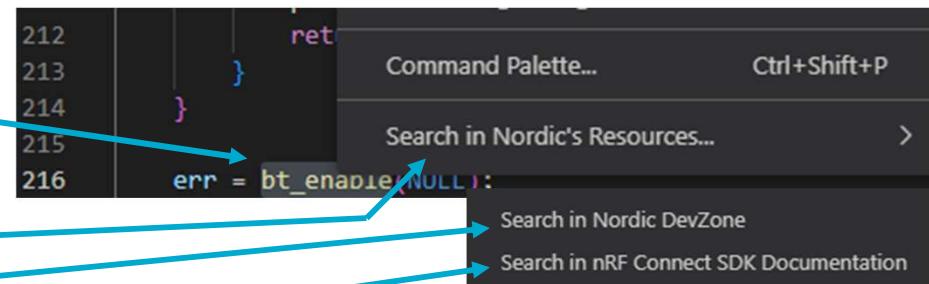
cb – Callback to notify completion or NULL to perform the enabling synchronously.

Returns:

Zero on success or (negative) error code otherwise.

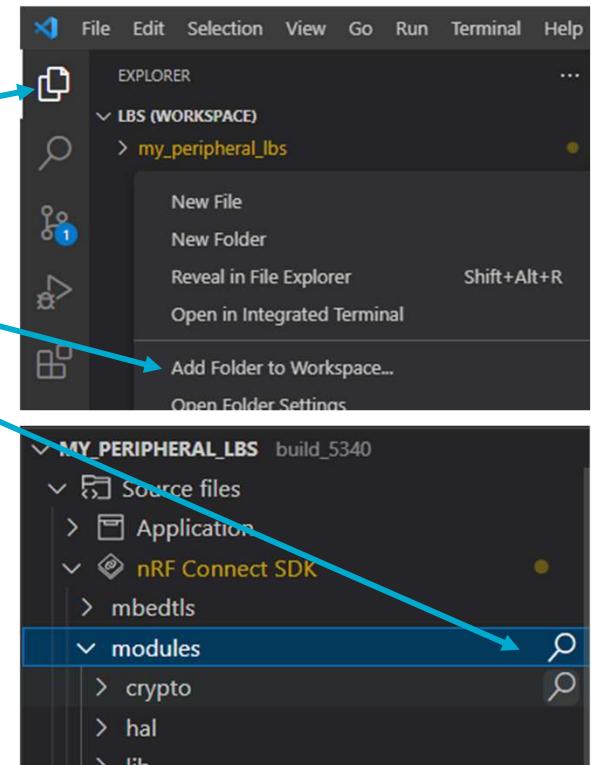
Editing tips and tricks, continue

- Search documentation:
 - Mark and right-click the text of interest
 - Select “Search Nordic’s Resources”:
 - Search in Nordic DevZone
 - Search nRF Connect SDK Documentation
- Code completion (almost always) available by clicking Ctrl+Space



Editing tips and tricks, continue

- Searching:
 - Searching relates to the VS Code workspace
 - To include SDK files in search:
 - Click the Explorer icon
 - Click “Add Folder to Workspace...” and select the directory to include (will include subdirectories as well)
 - To search mark the text and click the relevant search icon in the application view
 - In this view only files included in the build are listed
 - Each directory level has a search icon for limiting the search to a part of the tree



Search bug in July version

- To be able to search using the search icons in the “Source Files”, “Input Files” in the application view a few tricks must be done:
- Set up your applications and build configurations without saving a VS Code workspace.
- Build all configurations once
- Close workspace by clicking “close workspace” in the File menu.
- Click the Explorer icon (in the upper left corner of VS Code)
- Click the “Open Folder” button and navigate and select the folder containing your application (level above the build folders)
- If prompted with Adding application to visible applications, click “Yes”
- If prompted with SDK and Toolchain versions chose the SDK and matching toolchain versions
- Click the nRF Connect for VS Code icon
- You may now save your VS Code workspace and search will work as intended
- This will be fixed in the next main version of nRF Connect for VS Code

nRF Command Line Tools

- Offering: [nrfjprog.exe](#)
- Features:
 - Programming and verification (*)
 - Support nRF51, nRF52, nRF53 and nRF91 Series
 - Via programming interfaces
 - Via bootloaders support
 - Device locking and un-locking
 - Segger J-LINK support
 - Hex file merging
 - For full help type nrfjprog -h

```
C:\WINDOWS\system32>nrfjprog
Usage:
-----
```

-q --quiet	Reduces the stdout info. Must be combined with another command.
-h --help	Displays this help.
-v --version	Displays the nrfjprog and dll versions.
--force	Bypass all questions to continue. Recommended to use this flag when scripting.
--log [<path>]	Enable logging. Default output file is "log.log". Set the <path> option to modify log file location and/or name. If the parent folder of <path> does not yet exist, nrfjprog will attempt to create it. Logger output is always appended to the file. Must be combined with another command.
--jdll <file>	Uses the Segger's JLinkARM dll specified in the given file path instead of searching for the latest version of Segger's JLinkARM dll. Must be combined with another command. Limitations: Unicode paths are not supported
--ini <file>	Uses the nrfjprog settings file specified in the given file path instead of searching for the default nrfjprog.ini file in the installation folder. Must be combined with another command.

(*): Nordics tools are *not* intended for factory or high-volume programming

Zephyr project tool: West

- West features:
 - Repo management
 - “west init ...”, “west update”, “west list” and more
 - Building nRF Connect applications
 - “west build ...”
 - Flashing
 - “west flash”
 - Command line debugging
 - “west debug”
- Installing west
 - West is installed by the Toolchain Manager
 - West can also be installed manually, refer to [documentation](#)

```
C:\Nordic\ncs\v2.0.0>west
usage: west [-h] [-z ZEPHYR_BASE] [-v] [-V] <command> ...

The Zephyr RTOS meta-tool.

optional arguments:
  -h, --help            get help for west or a command
  -z ZEPHYR_BASE, --zephyr-base ZEPHYR_BASE
                        Override the Zephyr base directory. The default is
                        the manifest project with path "zephyr".
  -v, --verbose          Display verbose output. May be given multiple times
                        to increase verbosity.
  -V, --version          print the program version and exit

built-in commands for managing git repositories:
  init:                create a west workspace
  update:              update projects described in west manifest
  list:                print information about projects
  manifest:            manage the west manifest
  diff:                "git diff" for one or more projects
  status:              "git status" for one or more projects
  forall:              run a command in one or more local projects

other built-in commands:
  help:                get help for west or a command
  config:              get or set config file values
  topdir:              print the top level directory of the workspace

extension commands from project manifest (path: nrf):
  ncs-loot:            list out of tree unrebuilt NCS patches
  ncs-compare:         compare upstream manifest with NCS
  ncs-sbom:            generate Software Bill Of Materials including
                        license information

extension commands from project zephyr (path: zephyr):
  completion:          display shell completion scripts
  boards:              display information about supported boards
  build:               compile a Zephyr application
  sign:                sign a Zephyr binary for bootloader chain-loading
  flash:               flash and run a binary on a board
  debug:               flash and interactively debug a Zephyr application
  debugserver:          connect to board and launch a debug server
  attach:              interactively debug a board
  zephyr-export:        export Zephyr installation as a CMake config
  spdx:                create SPDX bill of materials

Run "west help <command>" for help on each <command>.
```