

# Nordic Bluetooth ESL Tutorial

Complete one-to-many application example with PAwR (Periodic Advertising with Responses) and EAD (Encrypted Advertising Data) Bluetooth features, featuring Bluetooth ESL Service and Profile. It includes how to build your ESL tag from scratch.

## Table of content:

About this document .....	2
Section 0: Learn the basics of PAwR and Bluetooth ESL .....	4
Section 1: Getting started with ESL using pre-compiled hex and automatic onboarding.....	5
Section 2: How to install SDK and build the hex files used in this tutorial.....	14
Section 3: Manual on-boarding, using shell command to fully control the ESLs.....	20
Section 4: Links to documentation, spec and training material.....	30
APPENDIX A: Flash DevKits using CLI(nrfjprog) .....	32
APPENDIX B: Enabling logs.....	33
APPENDIX C: Creating display images .....	38
APPENDIX D: Adding new display drivers and creating your TAG application .....	40
APPENDIX E: Alternative ways to transfer images from PC to AP .....	50
APPENDIX F: Measuring current on nRF52833 DevKit with PPK2 .....	53
APPENDIX G: Flashing a nRF52-based custom board .....	57

## About this document

### Versioning:

- Document V 0.9 - 10/05/2024
- Code repositories, documentation, licenses:  
SDK: [nRF Connect SDK 2.5.1](#)  
Sample Application: [nrf-esl-bluetooth Tag 2.5.1](#)  
(Note: ESL source code included in the Bluetooth\_ESL\_Getting\_Started\_Kit.zip includes an extra board definition used for the tutorial)
- Go to [www.nordicsemi.no/esl](http://www.nordicsemi.no/esl) to get latest version of this document

### Scope:

- Get started quickly with Nordic Bluetooth ESL implementation
  - Get all resources needed (hw, sw, tools, trainings, doc, specification) to design your own PAwR based product, starting from Nordic simple PAwR samples or from Nordic Bluetooth ESL samples
- By completing this tutorial, you will learn:
- the basics of PAwR and Bluetooth ESL and where to find additional resources
  - the basics of Nordic Bluetooth ESL implementation and code
  - how to replicate the Nordic Bluetooth ESL demos
  - how to download the nRF Connect SDK and to get familiar with many Nordic tools
  - how to modify the code to build your own TAGs
  - how to use shell commands to control the AP and control tags for demo or development
  - how to measure currents on Nordic DevKits with PPK2

### Demos:

- Demos are included in the video linked in “Section 0: Learn the basics of PAwR and Bluetooth ESL”.

### HW Requirements:

- Mandatory (minimum setup):  
1x [nRF52833DK](#) (for the TAG – Peripheral – simulated display on LED);  
1x [nRF5340DK](#) (for the Access Point – AP - Central);

*You can get DevKits from Distributors (click “buy now” in each DevKit pages linked above) or contact [Nordic Sales](#)*

- For Electronic Paper Display (Optional)  
1x [Waveshare shield version \(version B\) \(SKU 26506\)](#) ;  
1x [250x122 2.13inch Waveshare EPD \(SKU 12672\)](#) or [400x300 4.2inch Waveshare EPD \(SKU 13186\)](#)  
(for out of the box experience, use the displays on the links above; make sure on the back of the

screen you see V4 label for the 2.13inch and V2 label for the 4.2inch: these are the ones tested and supported for this tutorial)

- For Current Measurements (Optional)  
1x [Power Profiler Kit II](#)

**Notes:**

- It is suggested to start with this tutorial and then access the online documentation and source code for Bluetooth ESL
- Links to doc, code, learning material are listed in “Section 4: Links to documentation, spec and training material”
- The precompiled hex files referred in this document are for demo purposes only. Customers must modify the code and build their own hex files/applications for production purposes
- For licenses, refer to licenses in the online code repositories linked above and nRF Connect SDK

**“Bluetooth\_ESL\_Getting\_Started\_Kit” Zip - content** (download latest at [www.nordicsemi.no/esl](http://www.nordicsemi.no/esl)):

- Nordic Bluetooth ESL tutorial.pdf (this document)
- Precompiled hex files:
  - two for AP: automatic onboarding; manual onboarding;
  - three for nRF52833DK Tags: simulated display, 2.13inch, 4.2inch (see HW requirements above)
- nrf-esl-bluetooth folder with:
  - source code for the application
  - python tools (ESL TLV generator, decryption tool for responses, etc)
  - display images (two for 2.13inch, two for 4.1inch)



## Section 0: Learn the basics of PAwR and Bluetooth ESL

Before starting, it is strongly recommended to get some basic concepts about PAwR and Bluetooth ESL Service and Profile by watching the video available at [www.nordicsemi.no/esl](http://www.nordicsemi.no/esl). In this video, you will get familiar also with several demos.

In addition to the video, there is a collection of links related to PAwR and Bluetooth ESL (and more) in “Section 4: Links to documentation, spec and training material”. This is meant to be used as support/reference for the topics in the tutorial.

## Section 1: Getting started with ESL using pre-compiled hex and automatic onboarding

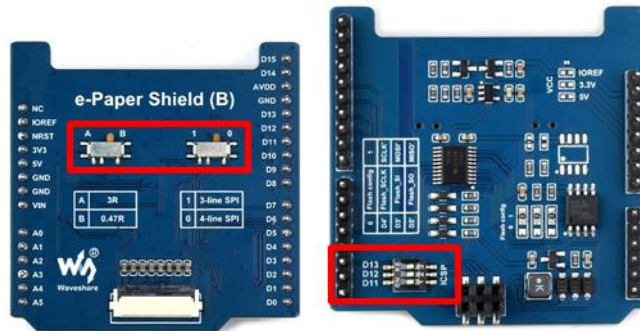
In this section, a step by step guide is presented to put together a simple demo using Nordic Bluetooth ESL implementation. Nordic auto-onboarding feature is used on the AP, so that the AP handles automatically the association and synchronization of ESL Tag. Once completed this section, you can follow “Section 3: Manual on-boarding, using shell command to fully control the ESLs ” for manual onboarding via shell command.

### Notes:

- AP with automatic onboarding is meant to be used mainly for development purposed.
- The AP can support 100+ tags in this mode
- For current measurements, refer to “APPENDIX F: Measuring current on nRF52833 DevKit with PPK2”
- in this section, logs are disabled. If you need to activate logs, see “APPENDIX B: Enabling logs”

### STEPS:

- 0) Unzip “Bluetooth\_ESL\_Getting\_Started\_Kit.zip”. Copy nrf-esl-bluetooth folder into C (i.e. C:\nrf-esl-bluetooth)
- 1) [Skip this step if you don’t have the exact Shield and EPD specified above]  
Put together the nRF52833DK and EPD (Electronic Paper Display) in the following way :
  - a. On the Waveshare shield (version B), there are several switches on the front and the back



Make sure the switches are in the right position depending on the display size you plan to use:

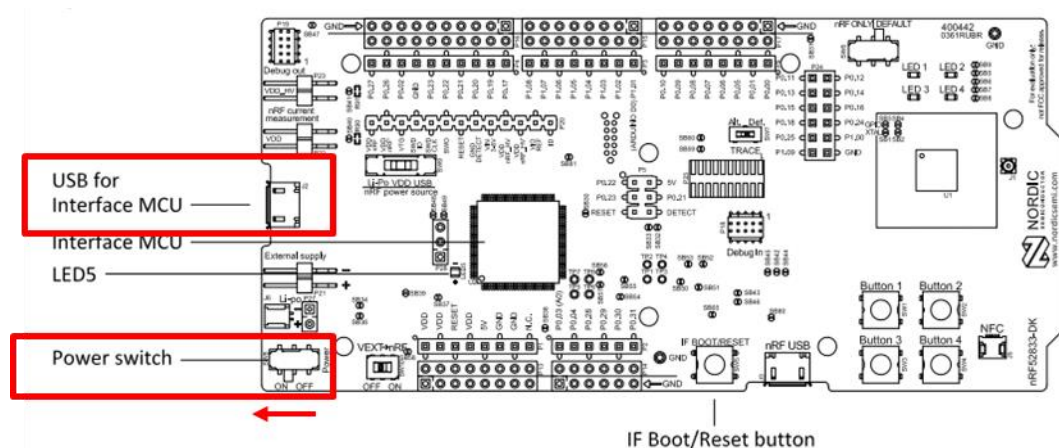
- For 2.13inch: (on the front) to **0**, **B**, and (on the back) to **D11**, **D12**, **D13**
- For 4.2inch (V2): (on the front) to **0**, **A**, and (on the back) to **D11**, **D12**, **D13**

- b. Plug the shield on top of the nRF52833DK

- c. Finally connect the EPD display to the shield (display is blank)



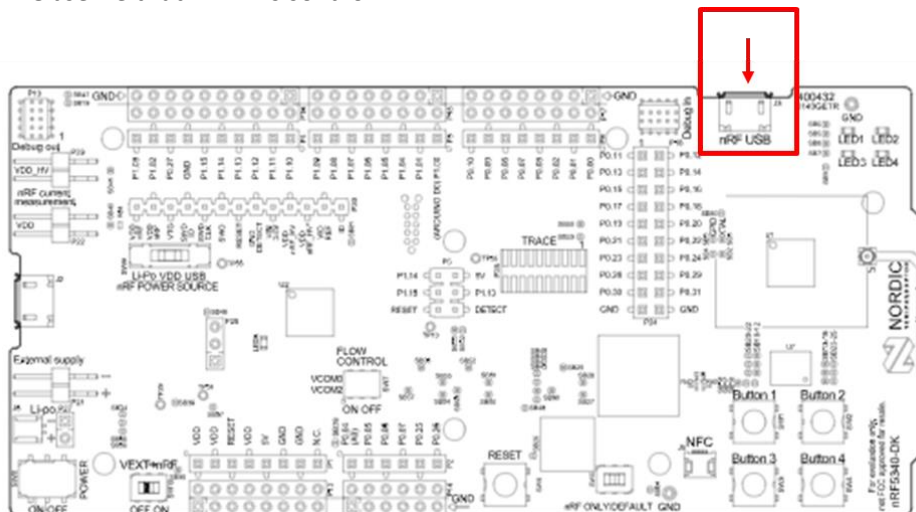
- 2) Flash the AP-Central (nRF5340DK) and the TAG-Peripheral (nRF52833DK) with precompiled hex files as follows:
  - a. Install the latest [nRF Command Line Tools](#) (10.23.2 or later) (from cmd, you can verify version: `nrfjprog -v`)
  - b. Install [nRF Connect for Desktop](#) (if already installed, from “settings” make sure to update it, and then install and update all the apps)
  - c. Connect one DK at the time to the PC via “USB for interface MCU” and turn it ON



- d. Open “Programmer” app from the nRF Connect for Desktop tool and flash “erase and write” the DK with the right hex file located in the “hex files” subfolder (Programmer instructions [here](#)). After flashing, before adding a new file to be flashed on another board, click on “Clear files”
- If you prefer command line to flash the DK, you can find instructions in “APPENDIX A: Flash DevKits using CLI(nrfjprog)”;* *if you want to build your own images and flash with VS Code, you can find instructions in “Section 2: How to install SDK and build the hex files used in this tutorial”*

- nRF52833DK TAG:

- a. use this hex if you have shield with 2.13inch display →  
ESL\_52833\_2\_13\_EPD.hex
  - b. use this hex if you have shield with 4.2inch display →  
ESL\_52833\_4\_2\_EPD.hex
  - c. use this hex for Simulated Display on LED (no shield/no EPD display) →  
ESL\_52833\_LED.hex
- nRF5340DK AP: use the following hex  
nRF5340DK\_AP\_AUTO.hex  
(note: when “erase and flash” the nRF5340DK AP with Programmer app, the content of the external QSPI flash is not erased – i.e. any previous AP database and display images are kept; to erase external flash, you can find instructions in “APPENDIX A: Flash DevKits using CLI(nrffjprog)”)
    - e. After programming, turn all DKs OFF.
- 3) [Skip this step if you don't have the exact Shield and EPD specified above]  
Install the following tools on your PC:
    - a. Install latest [GO Language](#)
    - b. Then install [Mcumgr Command-line Tool](#), by running the following command from Command Line CMD (requires GO>=1.18)  
go install github.com/apache/mynewt-mcumgr-cli/mcumgr@latest
  - 4) Connect the nRF5340 (AP-Central) to the Host PC and prepare it for next steps, as follows:
    - a. Connect the nRF5340DK (AP) to the PC using secondary USB port “**nRF USB**”, and turn it ON. Observe that LED 1 is solid on.



- b. Open a shell terminal: for example, you can use “Serial Terminal” app inside nRF Connect for Desktop: after selecting the AP on top left corner, press enter on the top bar, and it

should return a line “ESL\_AP:..”. If the AP does not respond, select a different COM port.



**Important Note:**

The AP exposes several Virtual COM Ports over USB.

One is for shell commands (generally it is automatically selected by “Serial Terminal” tool – COM6 in this example).

The other Virtual COM Port is used later by MCUmgr tool to transfer display images from PC to AP. Take note of which is the other available COM Port (e.g. you can “Disconnect from port” and see all the available COMs, then reconnect the AP to the Serial Terminal)

- c. *[Skip this step if you don't have the exact Shield and EPD specified above]*

Transfer the display images from the PC to the AP in the following way.

From the folder where the display images are located (C:\nrf-esl-bluetooth\gui\images), open cmd and run:

```
mcumgr --conntype="serial" --connstring="COM4,baud=115200,mtu=512" fs upload
esl_image_bt /ots_image/esl_image_00
```

```
mcumgr --conntype="serial" --connstring="COM4,baud=115200,mtu=512" fs upload
esl_image_nordic /ots_image/esl_image_01
```

```
mcumgr --conntype="serial" --connstring="COM4,baud=115200,mtu=512" fs upload
esl_image_empower /ots_image/esl_image_09
```

```
mcumgr --conntype="serial" --connstring="COM4,baud=115200,mtu=512" fs upload
esl_image_mcu_block /ots_image/esl_image_0A
```

Note: Change **COM4** to your extra COM port as mentioned in the previous point. File transfer will look something similar to this, for each image

```
C:\Users\loam\Downloads>mcumgr --conntype="serial" --connstring="COM7,baud=115200,mtu=512" fs upload esl_image_bt /ots_image/esl_image_00
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
Done
```

As an alternative to mcumgr, you can check other tools in “APPENDIX E: Alternative ways



to transfer images from PC to AP”

Images are stored in external flash in nRF5340DK. So, if repeating the demo or after programming, generally, no need to repeat this step.

When the AP connects to each tag (and auto-onboarding is active), the AP configures the tag and it will also transfer the right images to the right tag, before synchronizing them. If tags already have these images because tag is already in the AP database, images are not transferred.

In this demo (using auto-onboarding feature), the AP decides which image to send to which tag, depending on the PID provided by the tag. More on this (advanced) topic in “APPENDIX D: Adding new display drivers and creating your TAG application”.

In “APPENDIX C: Creating display images” instructions on how to create your own images.

- d. Push button 2 on nRF5340DK to erase database of previously onboarded tags (database is also stored in external flash, and it is not erased when programming the DK).
  - e. Push button 1 on nRF5340DK. This will activate automatic scanning and onboarding. Observe LED1 is blinking now on nRF5340DK, signaling the feature is running. By using this feature, AP will continuously scan for new ESL TAG, connect to it, configure them, transfer display images to them and pass PAWR synchronization to the TAG. The AP builds a database with all tags and it is kept in external flash on the nRF5340DK. Such database includes: Bluetooth Address, ESL address (ID +Group), Unique response key, Bonding key, etc. Serial Terminal will print status of ongoing onboarding. With default configuration, Tags are added from Group 0 with increasing ESL\_ID (0,1,2,3,4,5,6,7) until reaching 8th. Then, the next Tag will be added to Group 1 with ESL\_ID 0, and so on. For handling more devices, shell commands should be used. ([on the documentation more details on this logic](#))
- 5) Start the nRF52833DK (TAG-Peripheral) application:
- a. Make sure the nRF52833DK is powered and the power switch on nRF52833DK is now turned ON
    - if you turn ON the DK and all LED are still off, it means that the ship mode is activated. Some tags have a ship mode feature activated in these hex files (it can be disabled on software easily using a kconfig); when booting up, the ship mode initialize/reset display and then it enters deep sleep, minimizing power consumption.
  - b. Push Button 2 (or use NFC) to exit shipping mode. LED will start blinking (see next step)
    - Note: For demo purposes, these hex files have an optional feature activated that prints in TEXT on the display the status of the tag and its Bluetooth address when it goes to Unassociated/Associated. (see next point)
    - Note: no specific app on the phone is needed for waking up via NFC. Make sure your phone has NFC active and just tap the phone on the NFC antenna
    - Note: once the device exit shipping mode, it is possible you scan NFC again and read the meta data which includes the Bluetooth Address. In this case, you need

an App on the phone to read NFC tags

- 6) Observe the nRF52833DK TAG while getting associated automatically by the AP (no action needed on the TAG)

When the AP (which is in automatic onboarding mode) scans this specific TAG, the TAG will go automatically through the following states:

- Unassociated (tag still not scanned by the AP)  
Observe LED 4 initially will blink fast (i.e. advertising)  
[if PPK 2 is used to measure current as explained in “APPENDIX F: Measuring current on nRF52833 DevKit with PPK2”, you will observe pulses every ca 450ms]  
[if using a shield with EPD display, observe EPD Display: it should print its status as “unassociated”]
- Configuring (AP has scanned and connected to the TAG)  
When connected to the AP, LED4 will be solid ON.  
In this phase, the AP will configure the TAG and transfer the display images using OTS service
- Synchronized (AP has passed PAwR synchronization to the TAG using PAST)  
When synchronized to the AP, LED4 will blink slowly  
(one very short LED pulse every time it receives synchronization – around 1.76s).  
Note:
  - if using a shield with epaper display, observe the display: it should have now changed its status to Associated and prints the ESL Address
  - on AP shell command, you can monitor the onboarding process and see which tag is associated with which ESL Address. (if logs are enabled also on the tag side, address and status is printed on shell command and RTT – see “APPENDIX B: Enabling logs”)

The LED 4 (status LED) on the TAG DevKit gives very useful indication of the status (here a summary):

- blinking fast: TAG is advertising (TAG can be either unassociated or unsynchronized)
- solid light: TAG is connected to the AP
- blinking slowly: TAG is synchronized to the PAwR train (it blinks when it receives the sync packet/PAwR subevent)

- 7) From the AP, you can trigger actions (i.e. send commands over PAwR):  
example, by pushing Button 4 and Button 3 in the nRF5340DK(AP), pre-built TLV commands are sent. Every time you push these buttons, some commands rotates: it will send a different commands first to group 0, then 1, then 2, then back to 0 (AP will also print information on which group is toggled in the shell command).  
By default, there are 4 commands memorized per group under button 3 and 2 commands per group for button 4.

(if you want to limit to group zero use "esl\_c groups\_per\_button 1" on shell command, this will be stored in flash)

You can try:

- Push Button 4 for first time on the nRF5340DK:  
LED 1 in the nRF52833DK will start blinking with a predefined pattern.  
By pushing button 4 again in the AP, it will request TAGs in Group 0 to turn off the LED1 (on nRF52833DK).  
*Tip: after you push the button, wait 1.7s before pushing one more time: this will make sure the Tag has received the command (the TAG wakes up every PA interval, around 1.7s)*
- Push Button 3 for first time in the nRF5340DK:
  - a. *[if you don't have shield/EPD]*  
Observe LED3 (simulated display) on nRF52833DK TAG: it will blink 5 times anytime the TAG receives command to change the display image.
  - b. *[If you have shield/EPD]*  
Observe EPD display image will change. Press again button 3 to change again display image.

Please refer to ["User Interface" section in the documentation](#)

On the AP, you can also play with shell command to read the AP outputs and to trigger easily some pre-defined commands:

example:

esl\_c pawr update\_pawr 12 0

→ this command sends to Group 0 a set of TLVs (addressed to tags with ESL ID 0, 1...10) with command to change display to Image 0

esl\_c pawr update\_pawr 13 0

→ this command sends to Group 0 a set of TLVs (addressed to tags with ESL ID 0, 1...10) with command to change display to Image 1

esl\_c pawr update\_pawr 3 0

→ this command sends to Group 0 a broadcast TLV to run LED1 ON (logically in ESL spec, this is "LED 0")

esl\_c pawr update\_pawr 4 0

→ this command sends to Group 0 a broadcast TLV to run LED1 OFF

More ["Predefined ESL sync packet" on Central ESL subcommands documentation](#)

It is also possible to build and send specific payload for the PAwR sync and receive responses from tags. This is covered more in details in "Section 3: Manual on-boarding, using shell command to fully control the ESLs "

- 8) If you want to associate more TAGs to the AP, program a new nRF52833DK with the right hex file as above and then repeat steps 5, 6 and 7. You can associate more than 100 tags to the AP.
- 9) To repeat the association process again from scratch, press the following buttons:
  - a. On nRF5340DK (AP): Button 2 to delete the database of associated TAGs;
  - b. On nRF52833 (TAG): Button 1 to move the TAG into unassociated state and to start advertising.

To enable current measurements with PPK2 follow “APPENDIX F: Measuring current on nRF52833 DevKit with PPK2”.

If, instead, you want to associate and handle tags “manually” via shell command (e.g. decide which ESL address to assign, etc), refer to “Section 3: Manual on-boarding, using shell command to fully control the ESLs”.

#### IMPORTANT NOTES:

- If you reset or switch off/on the Tag (nRF52833DK), it enters shipping mode. To wake up again the tag, use button 2 or NFC. (exception: tag does not enter ship mode if reset is due to DFU).  
Note: Tag keeps association status after the power cycle.
- If you reset or switch off/on the AP (nRF5340DK), it stop the automatic scanning. To start scanning and onboarding automatically tags, push again Button 1.  
Note: AP keeps association status and database after power cycle
- During association, the AP and TAGs store Bluetooth LTK bond keys. If bond key is deleted in any of the side (e.g. by pushing button 2 on AP OR by pushing button 1 on the TAG), in order to re-associate the tag, also the other side needs to be removed first (e.g. by pushing button 2 on AP OR button 1 on the TAG).
- If you reset or switch off/on the AP (nRF5340DK) while tags are synchronized, clearly all tags will lose sync. However, it takes around 6x PA time (around 10s) before tags realize sync is lost and transition to unsynchronized status. So, after resetting the AP, wait around 10 seconds: only when tags start advertising again (see LEDs), try to onboard again.

#### ADVANCED TOPICS

- Note: while using auto-onboarding, it is possible to use the shell command to send more “predefined ESL sync packets” or to send specific TLV and get responses, to connect to tags while they are sync, to transfer images and to manage the database. Before trying shell commands with the automatic onboarding feature on, it is suggested to try “Section 3: Manual on-boarding, using shell command to fully control the ESLs” first in order to get familiar with AP shell command. For now, note these useful shell commands for automatic onboarding:



- to remove all tags from AP database:  
*esl\_c remove\_all\_tags*  
*equivalent to button2 on the AP DevKit*
  - to remove a specific tag from the database (e.g. <esl\_addr> = 0002)  
*esl\_c remove\_tag <esl\_addr>*
  - to reduce number of groups toggled by button 3 and button 4 on the AP:  
*esl\_c groups\_per\_button 1*
- It is possible to export/import database. More on this is documented in the gui/README.md file for the python tools
  - [here](#) is the code where mapping between buttons and pre-defined commands is done
  - [here](#) is the code where pre-defined commands are built

## Section 2: How to install SDK and build the hex files used in this tutorial

Requirements: make sure the following are installed in your PC, before moving forward

- [nRF Command Line Tools](#)
- [nRF Connect for Desktop](#)
- [VS Code](#)
- Copy nrf-esl-bluetooth folder into C (i.e. C:\nrf-esl-bluetooth)

/\*

If problems with toolchain, you can refer to the following links:

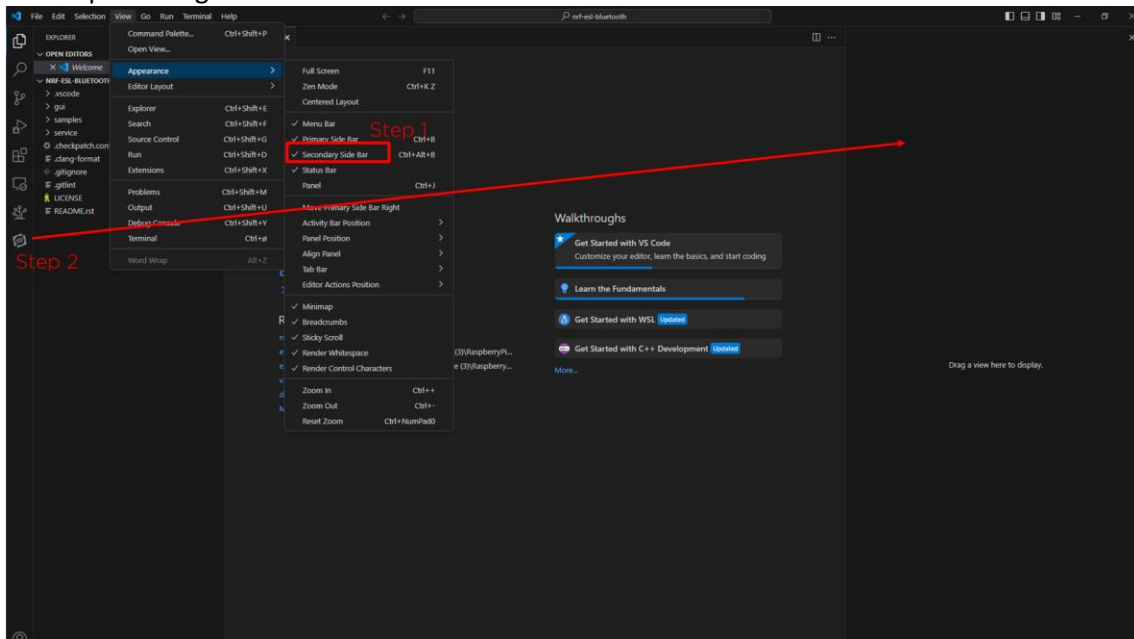
- [nRF Connect for VS Code extension pack - Documentation](#)
- [Installing the nRF Connect SDK](#)
- [lesson 1 - DevAcademy](#)

or contact Nordic on [DevZone](#)

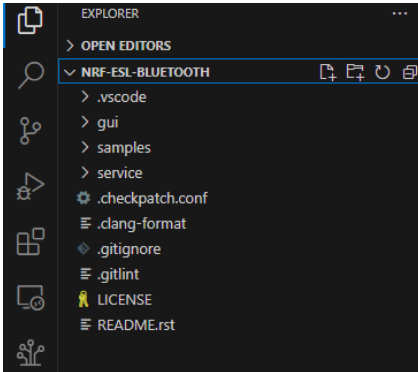
\*/

### Toolchain & SDK installation for Windows:

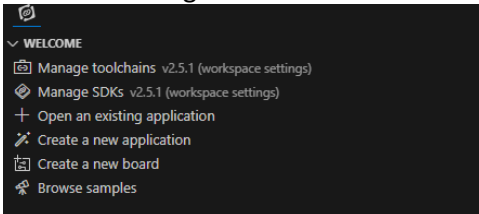
- Open “VS Code”
  - From top bar menu, go to View->Extensions. Then search (copy and paste) and install: **nordic-semiconductor.nrf-connect-extension-pack**
  - Suggestion: open the Secondary Side Bar, and drag and drop the nRF Connect extension. In this way, File Explorer is on the left and nRF Connect extension is always on the right
- See steps in image:



- From File, click on “Add Folder to Workspace” and select the C:\nrf-esl-bluetooth folder. VS Code will load this folder in the Explorer view



- From “Welcome” section of the nRF Connect extension:  
(if you don’t find it: go to View->Open View-> and type “nrf connect” -> welcome)  
Click on “Manage Toolchain” and choose 2.5.1 for this example and follow the instructions



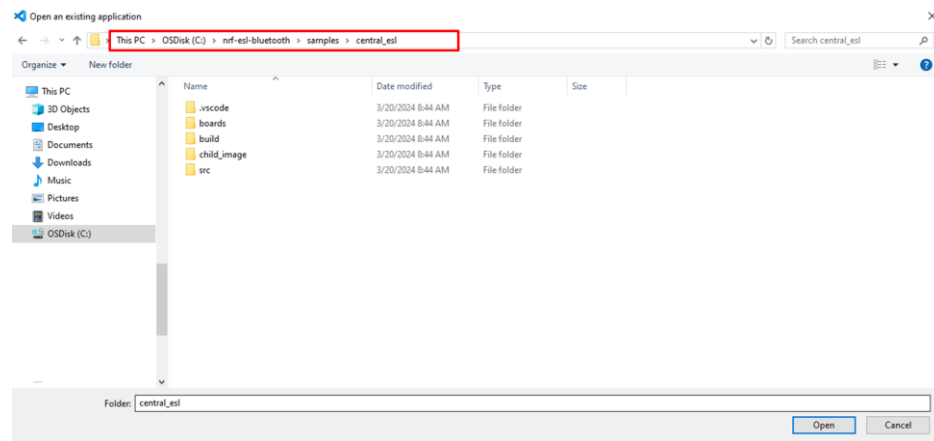
- From “Welcome” section, click on “Manage SDK”: use 2.5.1 for this example and follow the instructions
- For lowering power consumption when using the ssd16xx EPD driver display (2.13inch display) available in Zephyr repo, modify this file in the SDK just installed:
  - From “Welcome” section, click “Open SDK Directory” and choose the SDK you are currently using.
  - Navigate and open this file (e.g. add it to VS Code workspace):  
zephyr/drivers/display/ssd16xx.c
  - Modify this line (919), removing “static”, then save and close the file:  
from **static** int ssd16xx\_init(const struct device \*dev)  
to int ssd16xx\_init(const struct device \*dev)

#### NOTE:

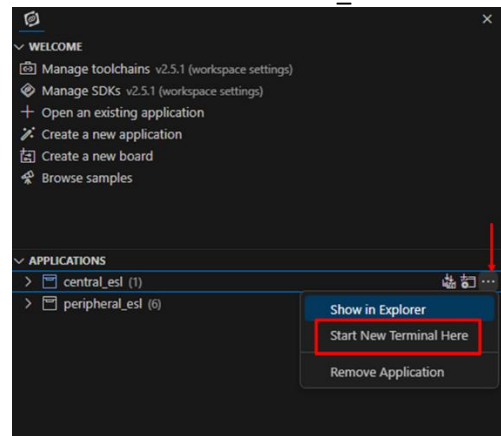
in “APPENDIX D: Adding new display drivers and creating your TAG application”, more details are presented in relation to the display drivers (zephyr and off-tree drivers).

### Building the hex for the AP and Flashing the nRF5340DK

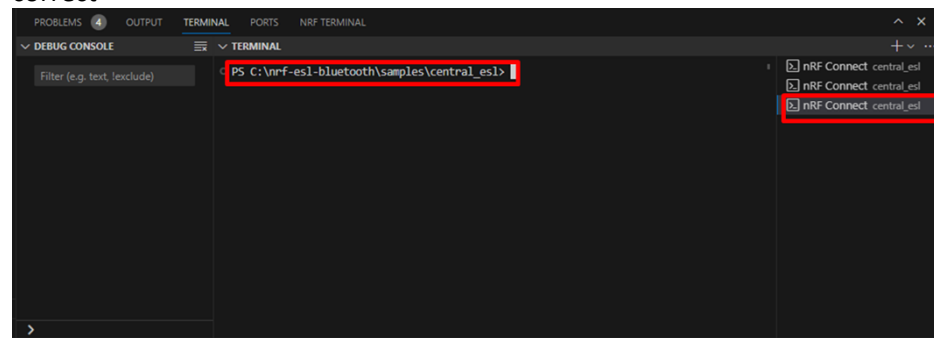
- Here the instructions to build and flash “nRF5340DK\_AP\_AUTO.hex” into the nRF5340DK
- From the nRF Connect extension, under WELCOME, click on “Open an existing application”, and select: C:\nrf-esl-bluetooth\samples\central\_esl  
Then click “Open”



- The application is now listed under “APPLICATIONS”.  
Click on “...” next to central\_esl and click on “Start New Terminal Here” :




- New terminal will open at the bottom, pointing to the right folder.  
Make sure it is “nRF Connect” terminal (not “powershell” or other) and the folder path is correct

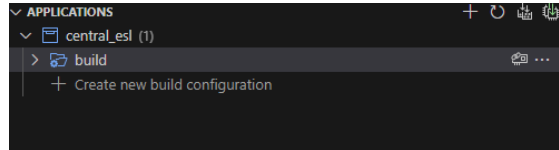


- Run this command for pristine build:  
`west build --build-dir build_nRF5340DK_AP_AUTO -p -b nrf5340dk_nrf5340_cpuapp`

(note: -p means pristine built. This will delete your previous build; west documentation [here](#))

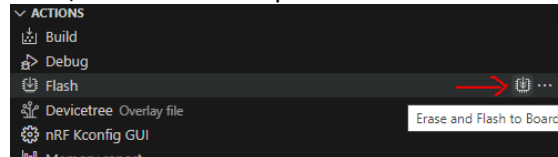


- After build is complete, you will notice a new subfolder listed in APPLICATIONS, under samples\central\_esl:
  - Click on the new “build” folder to select/highlight it (the icon should change to )



- Connect the nRF5340DK (turned ON) to the PC (use the “USB for Interface MCU” port on the DevKit).

Then, under ACTIONS push this “Erase and Flash”



Alternative: it is possible to find the right hex file to be used for flashing the nRF5340DK with Programmer app or nrfjprog: this file is named “merged\_domains.hex” and located in C:\nrf-esl-bluetooth\samples\central\_esl\build\zephyr  
Then use Programmer (in nRF Connect for Desktop) or CLI to flash the DK as in “APPENDIX A: Flash DevKits using CLI(nrfjprog)”

NOTE: to enable/use logs, see “APPENDIX B: Enabling logs”

- **Building the hex for the TAG and flashing nRF52833DK:**

- From the VS Code extension, under WELCOME, click on “Open an existing application”, and select: C:\nrf-esl-bluetooth\samples\peripheral\_esl. Then click “Open”
- Follow same steps as before, however now use the following commands in order build

Notes:

- these are powered optimized builds, so logs are disabled. To enable logs, see “APPENDIX B: Enabling logs”
- in this case, different build folders are created for each screen type
- 2.13inch display: ESL\_52833\_2\_13\_EPD.hex  

```
west build --build-dir build_ESL_52833_2_13_EPD -p -b nrf52833dk_nrf52833 -- -
DCONF_FILE="prj_release.conf" -DOVERLAY_CONFIG="nrf52833dk_nrf52833_b2in13_epd.conf"
-DCONFIG_ESL_POWER_PROFILE=y -
DDTC_OVERLAY_FILE="conf/nrf52833dk_nrf52833/nrf52833dk_nrf52833_b2in13_epd.overlay"
-Dmcuboot_DTS_ROOT="C:/nrf-esl-bluetooth/samples/peripheral_esl/"
```

Notes:

- ...b2in13... the “b” refers to the fact that DFU over external flash is enabled and the external SPI flash is located on the Waveshare shield (only on version “B” of the shield!)

- in Linux you can point to dts folder for mcuboot using:
 

```
-Dmcuboot_DTS_ROOT=$PWD
```
- **4.2inch display: ESL\_52833\_4\_2\_EPD.hex**

```
west build --build-dir build_ESL_52833_4_2_EPD -p -b nrf52833dk_nrf52833 -- -
DOVERLAY_CONFIG="nrf52833dk_nrf52833_4in2_v2_epd.conf;power_profiler.conf" -
DCONFIG_ESL_SHIPPING_MODE=y -
DDTC_OVERLAY_FILE="conf/nrf52833dk_nrf52833/nrf52833dk_nrf52833_4in2_v2_epd.overlay;
conf/nrf52833dk_nrf52833/power_profiler.overlay" -Dmcuboot_DTS_ROOT="C:/nrf-esl-
bluetooth/samples/peripheral_esl/"
```
- **Simulated Display on LED (no shield/no EPD display): ESL\_52833\_LED.hex**

```
west build --build-dir build_ESL_52833_LED -p -b nrf52833dk_nrf52833 -- -
DCONF_FILE="prj_release.conf" -
DOVERLAY_CONFIG="nrf52833dk_nrf52833_power_profiler_release_led.conf" -
DDTC_OVERLAY_FILE="conf/nrf52833dk_nrf52833/nrf52833dk_nrf52833_power_profiler_relea
se.overlay"
```
- To Flash, select the right build folder under APPLICATIONS, then in ACTIONS click Erase and Flash as described above.  
 Note: if you want to flash DK manually using CLI or Programmer, you need to find the right hex file. The hex files just built are located inside each build folder: C:\nrf-esl-bluetooth\samples\peripheral\_esl\build\_folder\_name\zephyr

For ESL\_52833\_2\_13\_EPD, the right hex files is “merged.hex”, because this built include MCUboot.

For ESL\_52833\_4\_2\_EPD and ESL\_52833\_LED, the right hex is “zephyr.hex, as DFU is disabled (no MCUboot)

More information on build types is [here](#)

*Then use Programmer app (in nRF Connect for Desktop) or CLI to flash the DK as in “APPENDIX A: Flash DevKits using CLI(nrfjprog)”*

- **NOTES:**
  - in “APPENDIX D: Adding new display drivers and creating your TAG application”, more details are presented in relation to ESL application and the different build configurations.
  - to enable/use logs, see “APPENDIX B: Enabling logs”

Before playing with the code, it is strongly recommended to go to <https://academy.nordicsemi.com/> and attend the following three free courses (with LinkedIn Certificates) to learn more about the Nordic nRF Connect SDK:

- nRF Connect SDK Fundamentals
- nRF Connect SDK Intermediates
- Bluetooth LE Fundamentals  
 (strongly recommended also for experts in Bluetooth LE who never used Bluetooth API in nRF Connect SDK before)

Once you have good understanding of nRF Connect SDK, you can check “APPENDIX D: Adding new display drivers and creating your TAG application” on how to build your own tag application.

## Section 3: Manual on-boarding, using shell command to fully control the ESLs

With manual on-boarding, it is possible to use the shell commands to associate tags, send/receive commands/responses, connect to tags and test larger amount of devices with more freedom on how to allocate tags to specific groups. The AP does not keep any database. Database can be built on the PC/Host side.

### Requirements:

- Having completed the “Section 0: Learn the basics of PAwR and Bluetooth ESL” and “Section 1: Getting started with ESL using pre-compiled hex and automatic onboarding”

### Set-Up

- Prepare the AP (nRF5340DK):
  - o Clean external QSPI flash by using nrfjprog as described in APPENDIX A: Flash DevKits using CLI(nrfjprog)
  - o Flash nRF5340DK with pre-compiled “nRF5340DK\_AP\_MANUAL.hex”  
*[this has been built by first modifying the prj.conf in the central\_esl project in VS Code, setting the following to “n”*  
`CONFIG_BT_ESL_AP_AUTO_MODE=n`  
`CONFIG_BT_ESL_TAG_STORAGE=n`  
`CONFIG_BT_ESL_AP_AUTO_PAST_TAG=n`  
`]`
  - o Connect nRF5340DK to the Host PC using nRF\_USB and open a serial terminal as described in “Section 1: Getting started with ESL using pre-compiled hex and automatic onboarding”
- For the Tags, you can use any of the nRF52833DK configurations as used in previous sections

Note: in case needed, for command syntax refer to [documentation](#)

Run the [commands](#) in this section in the following order to understand all major AP procedures, paying attention to the comments. Commands in [blue](#) are for the serial Terminal shell interface of the AP.

### One-off initialization

*These are needed only once after flashing the DK:*

```
esl_c auto_ap 0 //make sure Auto Onboarding is off; by default “Old ESL_AP_AUTO_MODE” should be 0
esl_c unbond_all //clean Bluetooth LE bonds
esl_c bond_dump // verify no bond is stored
```

### Loading display images from PC to AP

*In case you want to transfer some display images from AP to Tag, make sure images have transferred over SMP/USB from PC to the AP. This can be done also at run time. Open CMD from inside the folder these images are located, and then run the following commands, for example, to transfer display images form PC*

to AP (change the COM port and file name accordingly based on your secondary port available for the AP). As an alternative to mcumgr, you can check another GUI tool in “APPENDIX E: Alternative ways to transfer images from PC to AP”.

```
mcumgr --conntype="serial" --connstring="COM5,baud=115200,mtu=512" fs upload IMAGE_FILE_NAME_1 /ots_image/esl_image_00
mcumgr --conntype="serial" --connstring="COM5,baud=115200,mtu=512" fs upload IMAGE_FILE_NAME_2 /ots_image/esl_image_01
```

## Associating an Unassociated tag

*The following procedures assume you know the Bluetooth address of the tag. In our demo, the address is printed on the display or can be scanned over NFC. If you don't know the Bluetooth address of the tag, you can scan for tags using the following commands on the AP.*

[for debugging, there are alternative ways to find Bluetooth address: you can use nRF Connect for Mobile (only Android phones shows the address) or nRF Connect for Desktop (links in Section 4: Links to documentation, spec and training material”)]

```
esl_c acl scan 1 1 // it scans for one device
```

otherwise, eg. If you have more devices advertising, you can scan continuously:

```
esl_c acl scan 1 0 // starts scanning
```

```
esl c acl list_scanned //use this to print the scanned devices
```

```
esl_c_acl_clear_scanned //use this to periodically clean the list to allow for refresh
```

```
esl_c acl scan 0 1 // use this to stop scanning
```

*Note: since the tag is unassociated (e.g. push button 1 on the nRF52833DK), the assumption is that the Tag does not have any stored Bluetooth LE bonds. Same for the AP (i.e. the AP does not have any stored bond associated to the Bluetooth Address of this tag).*

`esl_c_acl list_conn` //check all connections are available for ACL (in this case we will use connection index 0). You can use this command at any time to verify the status of the connection

`esl_c acl connect_addr 1 DA:52:7C:17:93:3A //1` stands for random address; make sure tag is advertising (otherwise it will timeout); successful connection should return #DISCOVERY: 1,0x00; it will also return the connection index (in this case Conn\_idx:00); Try this command multiple times if it fails to enter Discovery. If no success, check that the TAG is powered and advertising, make sure it is in unassociated and that the AP has no previous bonds stored for this address. More on how to handle with these corner cases below in this section.

`esl_c_acl_bt_key_export` //this prints the bond key. Save it; the “1” after the address indicates address type; example:

[illegible]



`esl_c acl configure 0 0200` //this configures the tag: 0 is the connection index; 0200 is the ESL address you want to assign to the tag. Address in hexadecimal, where two MSB are the Group ID, and two LSB are ESL ID. In this case it is Group 2, ID 0

`esl_c esl_c dump` // this prints info for the tag and AP. Save this content for this tag (only here you can see the response key that has been written into this tag by the AP; this is printed only at this stage of the tag configuration and cannot be read from the tag again)

in this example the `TAG_RESPONSE_KEY` is:

RSP\_KEY : 0xad24949a607d4ab253551f422d7bcc90f17ee60ff0b0dd64

Save this key, as it is used later to decrypt the responses from the tag.

//following two commands assumes `esl_image_00` and `esl_image_01` have been already transferred from PC to AP as mentioned above

`esl_c obj_c write_filename 0 0 esl_image_00` // transfer image\_00 from AP to the tag

//wait for confirmation from AP

e.g. #OTS\_WRITTEN:1,00,0,14806,0x9aeafeed

`esl_c obj_c write_filename 0 1 esl_image_01` // transfer image\_01 from AP to the tag

//wait for confirmation from AP

`esl_c update_abs 0` //transfer absolute time to the tag

`esl_c update_complete 0` //send update complete (required by the standard)

`esl_c acl past 0` // it triggers PAST procedure to sync the tag to PAwR. Only if this is successful, the tag will consider itself “associated”.

Log should look like:

#PAST:1,00

Disconnected:Conn\_idx:0x00 (reason 0x13)

*Just for robustness, at this stage you can try to ping the tag by sending a command over PAwR (see next section)*

*If past is successful and you got the ping (i.e. no imminent need to connect to this tag), delete the bond from the AP:*

`esl_c unbond_ble 1 DA:52:7C:17:93:3A`

*If connection is lost ahead of completing the association procedure, try to connect again to the tag with the same bond. If it fails, try erasing the bond on the AP, and reconnecting again.*

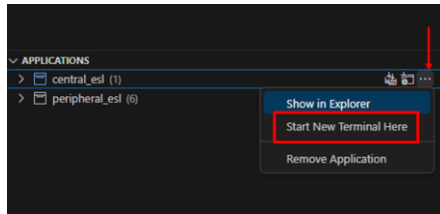
### **Sending commands to Synchronized tag**

`esl_c pawr sync_buf_status 2` // Optional command: it checks status of tx buffer. If full, you need to empty before sending new data (e.g. use “`esl_c pawr dump_sync_buf`”); 2 in this case is the group ID (PAwR)

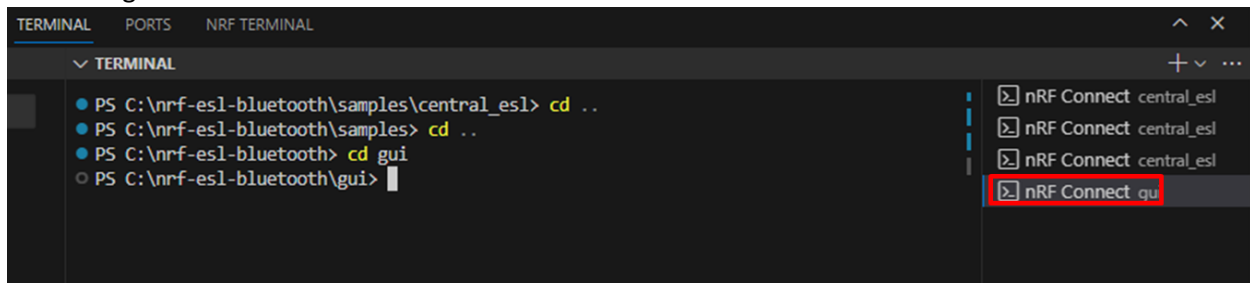
subevent);

To build the raw payload in hexademical for the commands in each subevent, you can use the ESL\_TLV\_GENERATOR.PHY, included in the GUI folder. Hereafter how to use it.

In VS Code, open “nRF Connect” terminal (e.g. from any APPLICATIONS folder, click on the three dots and “Start New Terminal”).



Then navigate to the GUI folder inside nrf-esl-bluetooth:



If never installed before, first install PySide:

pip install PySide6

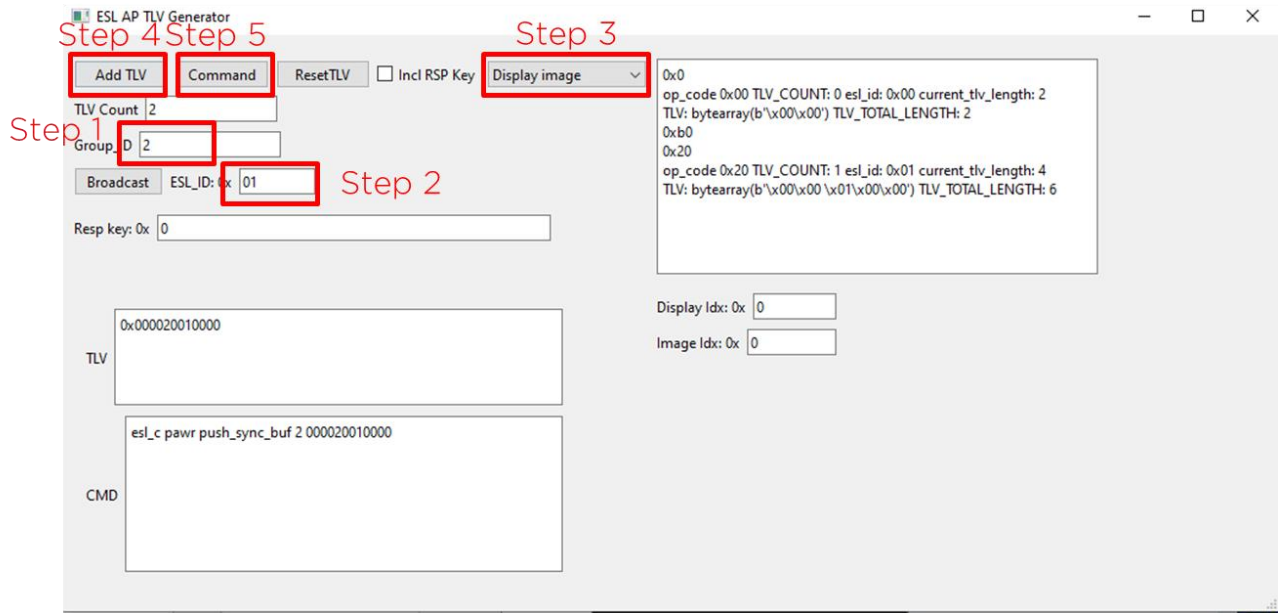
Then run the following command to open the TLV generator by Nordic:

python esl\_tlv\_generator.py

With this tools, it is easy to create some TLVs (which then can be sent using the AP shell command to the synchronized tags):

- 1) Select Group ID
- 2) specify the ESL\_ID for the first command
- 3) select the Command and play with the parameters
- 4) Push “Add TLV”. The first TLV is printed;  
//repeat 2, 3, 4, if you want to send more commands at the same time on the same subevent/same group;  
make sure to not exceed 48B payload; It can be to same tag or different tags in the same group;
- 5) Push “Command” to generate the command;

Click “ResetTLV” to start again



Now you can use the printed command (CMD) in the shell interface of the AP.  
e.g. to ping Tag 00 in Group ID 2, the raw TLV is 0x0000. So in the AP shell command, we run:

`esl_c pawr push_sync_buf 2 0000` // ping tag 00 in group ID 2; “0000” is the raw TLVs payload in hex sent to group 2 (OpCo is 00=Ping, Parameter is 00 – which corresponds to the ESL ID for the Ping command). Wait for reply before sending new command to the same group (it should arrive in less than 2x PA time; in this case around 3.4s). Note: given the structure of the raw command sent here, we know that we expect the tag with ID 00 to respond in slot 0. (i.e. the Host PC should be aware on in which slot it expects a response from which tag). This is important for decrypting the response (more on this later).

AP should return something like:

`#PUSH_SYNC_BUF:002 len 3`

`#RESPONSE:002#SLOT:0,0x2be372461d;` where 2 is the group; 0 is the slot

`esl_c pawr dump_sync_buf 2` // Use this to read the content of all the slots associated with group 2, and to clear those buffers. For example, response will look like:

`#GROUP:002,0x0f310a72b6c2f32ff9df4a13db4e940b`

`#RESPONSESLOT:00,0x0f3192067efdc700c2866a5bdabefb82` //this is the encrypted response (`RESPONSE_SLOT_ENCRYPTED_DATA`) in the slot 0 for subevent 2. Note: the mapping between response slot and ESL ID depends on how the TLVs chain has been created in the command sent to the group (refer to the service/profile to see in which slot each tag should send its response). In this example, we know that the response in slot 0 is coming from Tag 00 in Group 2; so we use the `TAG_RESPONSE_KEY` of Tag ID 00 to decrypt the payload received in response slot 0.





To help with this, the `ead_ccm.py` tool can be used (from same GUI folder).

This tool requires, one off installation of this library:

`pip install pyCryptodomex` )

then run:

`python ead_ccm.py -a RESPONSE_SLOT_ENCRYPTED_DATA -k TAG_RESPONSE_KEY -D -I`

In this case:

`python ead_ccm.py -a 0x0f3192067efdc700c2866a5bdabefb82 -k 0xad24949a607d4ab253551f422d7bcc90f17ee60ff0b0dd64 -D -I`

(you can check `python ead_ccm.py -h` to see all parameters)

The tool will reply with:

`decryptdata: b'0434100200'`

where (in hexadecimal):

04 -> indicates the number of following bytes

34 -> indicates that this is ESL EAD payload

100200 -> it is the actual payload (refer to service and profile; 10 is OpCode; 0200 is parameter in hex in little endian -> 0x0002 is the actual response in big endian; refer to service/profile to understand the syntax of the ping response)

*Note:*

*Similar to the AP with automatic on boarding, for simple testing you can also use some simple [“Predefined ESL sync packet”](#)*

*To change image (e.g. for group 2, for ESL ID from 0 to 10):*

`esl_c pawr update_pawr 12 2` //change to img0

`esl_c pawr update_pawr 13 2` //change to img1

*To get LED0 to flash, you can use a broadcast command (for group 2):*

`esl_c pawr update_pawr 1 2` // LED 0 flashing broadcast.

#### IMPORTANT:

Ping each tag every few minutes(e.g. 10-15min).

If a tag does not receive a ping for 1h, it goes to Unsynchronized.

#### SENDING TIMED COMMANDS:

The AP has an Absolute Time counter. This value is written into the ESL Current Absolute Time characteristic while on an ACL connection, so the TAG keeps the same absolute time. The AP can send a



past command after update\_complete anyhow).

//also in this case, you can verify the tag is sync by a Ping. If the tag is in Synchronized state, forget the bond key in the AP:

```
esl_c unbond_ble 1 DA:52:7C:17:93:3A
```

NOTE: if a tag does not respond to a ping, try again after few seconds (at least 6x PA interval). A missed ping can be due to collision or can be other reasons (e.g. tag has lost synchronization or run out battery, etc). Try to scan for the tag to see if it is advertising.

**AP scans for a tag previously associated (i.e. tag is advertising)**

**\*\*\*\*\*CASE A) Connecting to an Unsynchronized Tag (assumption: Tag is still associated to this AP)**

//assuming the tag has lost the sync and it is unsynchronized (tag is advertising). You can discover a tag has lost sync in two ways: by a missing response on a command (e.g. ping), or by scanning (tag is advertising). For testing, to simulate a lost of sync, you can stop and restart the PAwR on the AP (see later how to do this from shell commands)

To reconnect to the tag which is in Unsynchronized state:

```
//Import key the Bluetooth bond key saved previously
```

```
esl_c acl bt_key_import da527c17933a1
```

[illegible]

```
//connect to the tag:
```

```
esl_c acl connect_addr 1 DA:52:7C:17:93:3A
```

```
//If the connection is establish, put the tag in sync mode again:
```

```
esl_c update_abs 0 //transfer absolute time to the tag
```

```
esl c update complete 0 //update complete (required by the standard)
```

`esl_c acl past 0` //also in this case, you can verify the tag is sync by a Ping. If the tag is in Synchronised state, forget the bond key in the AP:

```
esl c unbond ble 1 DA:52:7C:17:93:3A
```

Note: if the tag is Unsynchronized for 1h, it will go to Unassociated;

\*\*\*\*\*CASE B)Connecting to an previously Associated Tag which moved to Unassociated (assuming AP is not aware of this) \*\*\*

In this case, it can happen you try to pair to an Unassociated Tag (which is advertising), while in the AP you use the previous bond (AP does not know tag moved to Unassociated).

For testing, to simulate this scenario, you can push button 1 in the TAG DevKit to move it to unassociated. Typical symptom, AP tries to connect to the TAG:

[illegible]

(Try to repeat `esl_c acl connect_addr 1 DA:52:7C:17:93:3A` command multiple times to exclude other issues e.g. collisions)

If the response looks like the following, it means Tag is rejecting the connection:

ESL AP:~\$ Connected:Conn idx:00

Disconnected:Conn\_idx:0x00 (reason 0x16)

It means AP failed to pair due to security reasons:

- Either AP is trying to use the old key, while the Tag is unassociated, i.e. tag has no Bluetooth bond
- or the Tag is associated to another AP, and currently is unsynchronized
- or the Tag is still associated, but the AP has (accidentally – ahead of the right timeout) deleted the old key

In this case, try to erase the key in the AP:

```
esl c unbond ble 1 DA:52:7C:17:93:3A
```

then try again to reconnect as if it is a new tag

```
esl c acl connect addr 1 DA:52:7C:17:93:3A
```

If this is successful, re-associate the tag.

If failing, it may be that the TAG is associated to another AP or still having old keys, or there are other problems with the communication.

## Other scenarios

A)

When trying to connect to an advertising tag, if the tag is not reachable (tag is off, or out of range), after a timeout, the AP will return a connection fail:

ESL AP:~\$ Connected Failed:Conn idx:00

B)

If the tag is not properly configured (e.g. no address, response key, etc), the Tag should not synchronize.

You will see something like:

`esl_c acl past 0`

`#PAST:1,00`

`#PAST:1,00`

`#PAST:1,00`

`#PAST:1,00`

`#PAST:1,00`

`ESL_AP:~$ Disconnected:Conn_idx:0x00 (reason 0x16)`

Either the tag has not been configured (`esl_c acl configure`) and/or the tag has not received `esl_c update_complete` ahead of the `past` command.

C)

To simulate tags to lose sync and go unsynchronized, it is possible to stop the PAwR train on the AP:

`esl_c pawr stop_pawr` //wait around 6xPA – 10s and tags will lose PAwR sync (e.g. wait the LED on the tag to start blinking fast)

`esl_c pawr start_pawr` //use it to turn on the PAwR again on the AP

D)

To simulate one tag to go unassociated while AP keeps old bond information, push button 1 on the tag DK

E)

to Unassociate a tag politely, first establish an acl connection and then use:

`esl_c unassociated 0`

(note: this will not erase the display images). Before reconnecting from the AP, wait few seconds and first remove old bond key in the AP, then try to connect again

F)

To factory reset the tag, first establish an acl connection and then use:

`esl_c factory 0`

(note: this will erase also the display images). Before reconnecting, wait few seconds and first remove old bond key in the AP.

F)

To Disconnect from acl connection (without putting tag in sync)

`esl_c acl disconnect 0`

## Section 4: Links to documentation, spec and training material

### - **Nordic Semiconductor news channels**

Subscribe [here](#) to one or more content provided by Nordic: Product Update Notifications, Product Lifecycle Information, News Release, Wireless Q free magazine, Get Connected Blog, Webinar, DevAcademy

### - **nRF Connect SDK**

nRF Connect SDK is the universal software by Nordic Semiconductor. Among other things, it features the world-renowned Bluetooth 5.4 qualified Nordic SoftDevice Controller.

This SDK is supporting current SoC families (nRF52, nRF53, nRF91, nRF70 – e.g. Bluetooth LE, Matter, Thread, Wi-Fi 6, LTE-M/NB-IoT, etc) and all future upcoming SoC families (e.g. nRF54H20, nRF54L15, etc).

Use DevAcademy to get started with nRF Connect SDK and to learn about the Bluetooth API in the context of nRF Connect SDK. These two free courses are highly recommended:

- [nRF Connect SDK Fundamentals](#)
- [nRF Connect SDK Intermediate](#)
- [Bluetooth LE Fundamentals](#)

### - **Learn more about Bluetooth 5.4, PAwR, EAD and Bluetooth ESL specification:**

- PAwR/EAD:
  - DevZone nRF Connect SDK guide: “[PAwR: a practical guide](#)” (with PAwR simple examples: [periodic\\_adv\\_rsp](#) and [periodic\\_sync\\_rsp](#))
  - Bluetooth [Core spec](#) and [Core supplement](#)
  - Bluetooth Core Specification v5.4 – [Technical Overview](#)
  - Bluetooth SIG training [Video](#)
- Bluetooth ESL
  - Bluetooth SIG [Service](#), [Profile spec](#)

### - **SW development for Bluetooth ESL:**

- Use the Nordic Bluetooth ESL samples (nrf-esl-bluetooth) provided as part of the Bluetooth\_ESL\_Getting\_Started\_Kit.zip (or the online [github](#) repository) as a starting point for your application, but first:
    - Go through this tutorial from Section 0 to Section 3, and follow the “APPENDIX D: Adding new display drivers and creating your TAG application” which includes an example to create your own TAG
- Documentation:
- [Nordic Bluetooth ESL \(AP and TAG\) Doc](#)
  - [Nordic Bluetooth ESL Service](#)
  - Refer to the nRF Connect SDK [documentation](#), including:
    - [SoftDevice Controller](#)

- [Bluetooth APIs](#)
  - [Samples, Applications](#) & Bluetooth [libraries](#)
  - Working with [nRF52](#) , [nRF53](#)
  - [Security](#)
  - Use [PPK2](#) to measure currents while developing the software
  - Ask [DevZone](#) for support questions and contact Nordic Sales to share case number
- **HW Development**
  - Choose the [Nordic Bluetooth SoC](#) that meet your needs  
([here](#) PS and HW documentation for both SoC, DevKits, etc)
  - Start with Nordic HW files (reference schematics and layouts): e.g.
    - [“Reference Layout” for nRF52833SoC](#) (TAG)
    - [“Hardware files” for nRF52833DK](#) (TAG)
    - [“Hardware files” for nRF5340DK](#) (AP)
    - [“Hardware files for Thingy53”](#) (AP)
  - For display circuitry, ask your supplier. Most of them provide reference schematic.  
Generally SPI plus few control GPIOs are required.  
(example in “APPENDIX D: Adding new display drivers and creating your TAG application”)
  - Design your [PCB Antenna](#) (use Nordic HW as reference example) or choose an off the shelf component
  - Get SoC samples from [Disti](#) or ask Nordic [Sales](#)
  - Ask [DevZone](#) for free schematic/layout review and free Bluetooth LE Antenna Tuning
  - [Add support for your custom board to the SDK](#)  
(see [DevAcademy nRF Connect SDK Intermediate Course Lesson 3](#), plus [doc](#), [webinar training](#))
- Use Nordic QDID for Bluetooth listing (e.g. [here QDID](#) for nRF52833). Learn more here on [Bluetooth](#) qualification
- **More resources:**
  - Get Technical Support on [DevZone](#) for both HW or SW questions
  - Get in touch with [Nordic Sales](#)
  - Explore [Nordic Partners \(Modules, Design and Solution\) and approved Distributors](#)
  - Leverage the Nordic tools and app for generic Bluetooth LE designs:
    - [nRF Connect Bluetooth LE](#)
    - [nRF Connect for Mobile](#) (e.g. scan, connect, DFU, etc)
    - [nRF Sniffer](#) for Bluetooth LE
    - [nRF Connect Device Manager](#) (DFU)
    - Other ([nRF Blinky](#), [nRF Toolbox](#), [nRF Edge Impulse](#), [Thingy:52](#), [nRF Mesh](#), [nRF Mesh sniffer](#))

## APPENDIX A: Flash DevKits using CLI(nrfjprog)

As alternative to Programmer app inside nRF Connect for Desktop, it is possible to use nrfjprog tool from cmd/terminal.

Requirement: nRF Command Line Tools v10.12.0 or later (nrfjprog -v to check the version)

Steps:

- Open CMD from the folder where the hex files are located.
- (optional, only if problems during flashing) recover a DK:  
nrfjprog --recover
- To erase and flash:
  - o For nRF5340DK, run:  
*(optional) erase external flash on nRF5340DK first*  
nrfjprog -e --qspi eraseall  
nrfjprog --recover  
  
*erase and flash nRF5340 SoC*  
nrfjprog -e  
nrfjprog -f NRF53 --program HEX\_FILE\_NAME.hex --recover --verify  
nrfjprog --pinreset
  - o For the nRF52833DK, to erase and flash nRF52833 SoC run:  
nrfjprog -e  
nrfjprog --program HEX\_FILE\_NAME.hex --verify -r

If you have more target devices connected to the PC, use the argument --snr to specify the serial number of the target. Example:

*list all the target devices:*

```
nrfjprog --ids
```

```
1050620370
```

```
1050044164
```

*reset a specific one:*

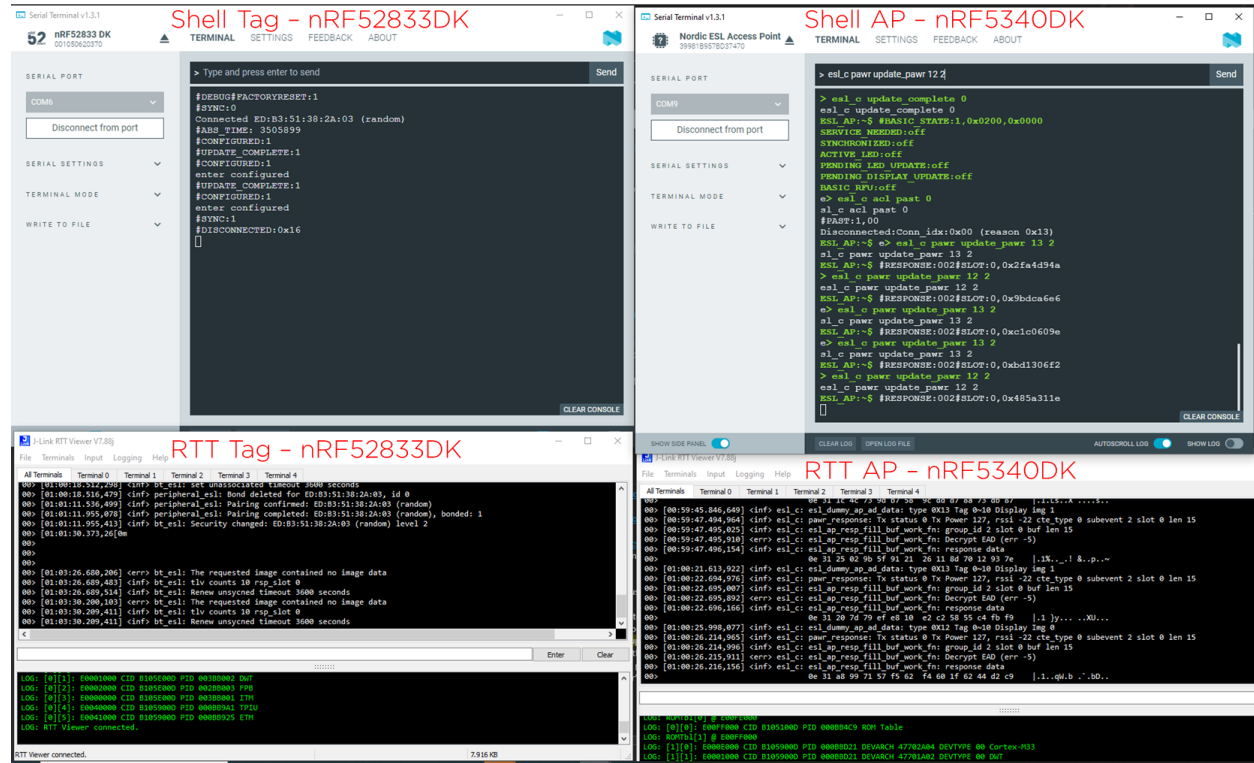
```
nrfjprog --snr 1050044164 -r
```





## APPENDIX B: Enabling logs

Sometimes, for troubleshooting or for reporting issues to DevZone, it is important to collect the right logs. For this project, if you have an AP (nRF5340DK) and an ESL Tag (nRF52833DK) it is possible to collect both shell and RTT logs for both. .



### Logs in nRF5340DK AP:

AP has already logs enabled by default. Both USB needs to be connected to the PC (DK turned ON).

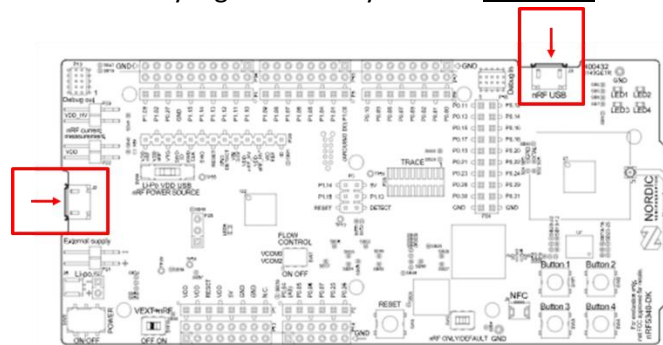


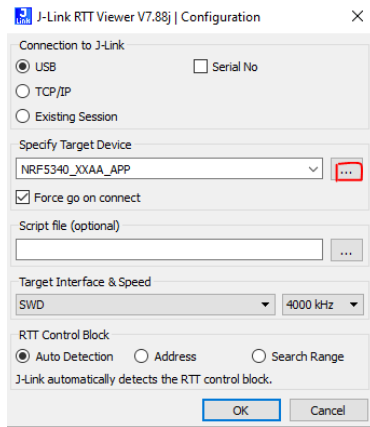
Figure 1. nRF5340 DK front view

nRF\_USB is used for AP Shell and for display image transfer (SMP - MCUmgr).

The other USB port (Interface MCU), which is generally used for flashing, can be used for RTT logs:

launch the “J-Link RTT Viewer” (installed automatically when installing nRF Command Line Tools), specify

“nRF5340\_XXAA\_APP” (click on three dots to find it) as Target device and click OK.



### Logs in nRF52833DK TAG:

In the hex files used in “Section 1: Getting started with ESL using pre-compiled hex and automatic onboarding”, logs are disabled (as those hex are optimized for low power).

To enable logs, when building the files (Section 2: How to install SDK and build the hex files used in this tutorial), in the west build command you need to remove references to low power options (different depending on the build). Example:

- Replace “prj\_release.conf” with “prj.conf”
- Remove `DCONFIG_ESL_POWER_PROFILE=y`
- Remove “power\_profiler.conf” and “power\_profiler.overlay”
- Remove “nrf52833dk\_nrf52833\_power\_profiler\_release\_led.conf” and “nrf52833dk\_nrf52833\_power\_profiler\_release.overlay”

Here the final build commands:

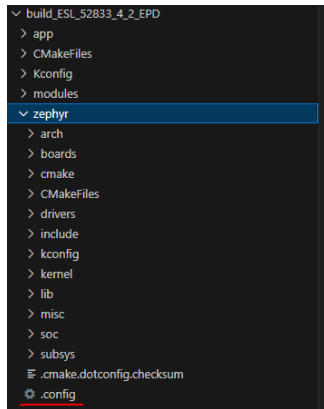
- **2.13inch display: ESL\_52833\_2\_13\_EPD\_with\_logs.hex**  

```
west build --build-dir build_ESL_52833_2_13_EPD_LOGS -p -b nrf52833dk_nrf52833 -- -
DCONF_FILE="prj.conf" -DOVERLAY_CONFIG="nrf52833dk_nrf52833_b2in13_epd.conf" -
DDTC_OVERLAY_FILE="conf/nrf52833dk_nrf52833/nrf52833dk_nrf52833_b2in13_epd.overlay" -
Dmccuboot_DTS_ROOT="C:/nrf-esl-bluetooth/samples/peripheral_esl/"
```
- **4.2inch display: ESL\_52833\_4\_2\_EPD\_with\_logs.hex**  

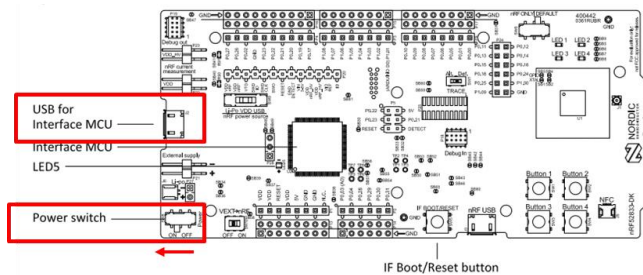
```
west build --build-dir build_ESL_52833_4_2_EPD_LOGS -p -b nrf52833dk_nrf52833 -- -
DOVERLAY_CONFIG="nrf52833dk_nrf52833_4in2_v2_epd.conf" -DCONFIG_ESL_SHIPPING_MODE=y -
DDTC_OVERLAY_FILE="conf/nrf52833dk_nrf52833/nrf52833dk_nrf52833_4in2_v2_epd.overlay" -
Dmccuboot_DTS_ROOT="C:/nrf-esl-bluetooth/samples/peripheral_esl/"
```
- **Simulated Display on LED (no shield/no EPD display): ESL\_52833\_LED\_with\_logs.hex**  

```
west build --build-dir build_ESL_52833_LED_LOGS -p -b nrf52833dk_nrf52833 -- -
DCONF_FILE="prj.conf"
```

Note: after building the application, in the build folder, you can verify the values assigned to all configurations (build\_folder→Zephyr→.config)



After building and flashing the DevKit, you need only one USB (and DK turned ON):

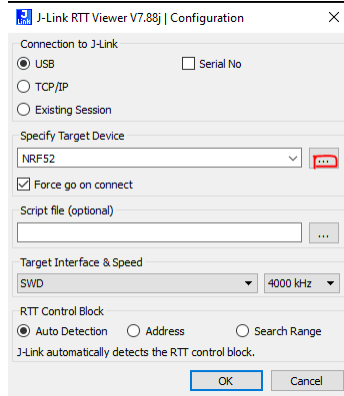


now both Shell command on ESL Tag and RTT logs are available, over the same USB.

For RTT, Open J-Link RTT Viewer (installed automatically when installing nRF Command Line Tools), specify “nRF52” (click on three dots to find it) as Target device and click OK.

RTT can be used also for custom boards where UART is not available (J-Link - e.g. nRF52xxxDK - to be

connected to the custom board).



Shell command (over UART) on the ESL Tag is an extra tool for development and troubleshooting (e.g. to trigger some action on the tag).

Use a CLI interface (e.g. Serial Terminal) to access the Virtual COM from the DevKit (you may try different COMs available from the DK) . Enter a command (e.g. help or esl help) , and it should return

ESL\_TAG:~\$ .....

These commands are currently not documented.

However, besides shell commands, the shell will also print out some status depending on the commands



from the AP.

The screenshot shows the Serial Terminal v1.3.1 application. On the left, the 'SERIAL PORT' is set to 'COM6' and 'nRF52833 DK' is selected. The 'SERIAL SETTINGS' are visible. The main terminal window shows the following output:

```
> |Type and press enter to send| Send

ESL TAG:~$ *** Booting nRF Connect SDK v2.5.1 ***
rr 0x00010000 Reset by Sense pin
ESL TAG:~$ h> help
elp
Please press the <Tab> button to see all available commands.
You can also use the <Tab> button to prompt or auto-complete all commands or its subcommands.
You can try to call commands with <-h> or <--help> parameter for more information.

Shell supports following meta-keys:
  Ctrl + (a key from: abcdefklmpuw)
  Alt  + (a key from: bf)
Please refer to shell documentation for more details.

Available commands:
  WHOAMI    :Tell UI what role I am
  clear     :Clear screen.
  esl       :ESL commands
  help      :Prints the help message.
  history   :Command history.
  kernel    :Kernel commands
  resize    :Console gets terminal screen size or assumes default in case the
             readout fails. It must be executed after each terminal width change
             to ensure correct text display.
  retval    :Print return value of most recent command
  shell     :Useful, not Unix-like shell commands.

ESL TAG:~$ > esl help
esl help
esl - ESL commands
Subcommands:
  state          :Show current state machine
  unassociated   :ESL debug commands
  factory        :ESL debug commands
  bond_dump     :ESL dump bond peer addr
  unbond        :ESL dump bond peer addr
  disconnect    :ESL disconnect peer
  adv           :ESL advertising start/stop
  led           :LED control for debugging
  display       :DISPLAY control for debugging
  led_work_dump :LED work item status dump for debugging
  display_work_dump :DISPLAY work item status for debugging
  sensor        :Read Sensor value for debugging
  busy          :ESL debug command to enable retry response
  accept        :ESL pairing key comply
  security      :ESL request security change
  configuring_bit :ESL set/get configuring bit
  abs           :ESL debug command to read/write absolute time
  check         :ESL Calculate Image checksum
  image_dump    :ESL debug command to dump image obj content
  bd_addr       :ESL device BD Addr
  esl           :ESL debug commands
```

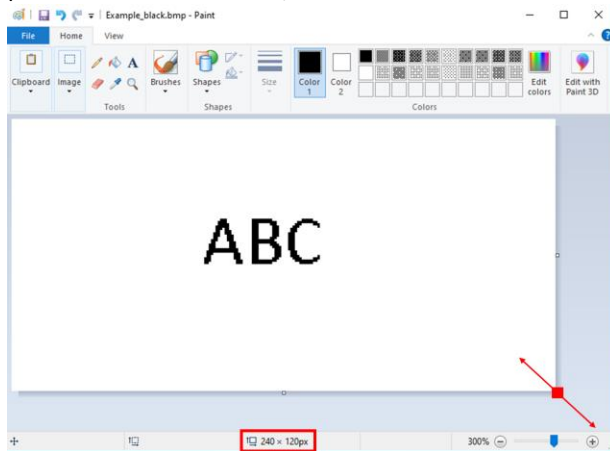
At the bottom of the terminal window, there are buttons for 'CLEAR LOG', 'OPEN LOG FILE', 'AUTOSCROLL LOG' (checked), and 'SHOW LOG'.

## APPENDIX C: Creating display images

For this demo, two approaches are presented to generate display images.

Before conversion, prepare the images in BMP format (e.g. use Paint):

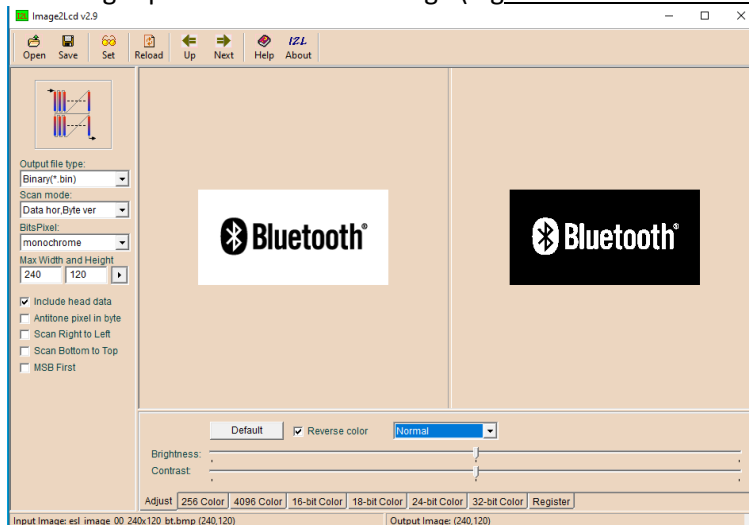
- Change the resolution of the image to match the display size and settings  
2.13inch -> W=240, H=120  
4.2inch -> W=400, H=300  
(do not invert W with H, since these match with the display settings in the software application!)



- Make sure to save the file as “Monochrome Bitmap”

For the **2.13inch Waveshare Display used in this demo** (W=240, H=120 - 2 colors B/W - using Zephyr display driver), Image2Lcd ([Waveshare download](#)) can be used in the following way:

- Select right parameters for the image (e.g. for 2.13inch 240x120) same as in this picture, then Save





For the **4.2inch Waveshare EPD (V2) used in this demo** (W=400, H=300 - 2 colors B/W - using off-tree display drivers), a python script is provided by Nordic. After preparing the monochrome BMP image for this display, from the GUI folder, in order to generate the output bin image, run:  
(you may need to install dependencies: `pip install numpy` and `pip install matplotlib`)

```
python epd_simulator.py -i input_image_filename.BMP -a g -o output_image_filename.bin
```

To verify the output bin file, you can run:

```
python epd_simulator.py -i output_image_filename.bin -a d
```

More details on the `epd_simulator.py` tool on README.md inside the GUI folder.

Note: generating images also has dependency on display drivers used in the tag application. To generate images for your EPD display, it is best to contact your supplier and ask for tools and support.

## APPENDIX D: Adding new display drivers and creating your TAG application

**Mandatory requirement for this section:** complete [DevAcademy nRF Connect SDK Intermediate course](#)

*Note: the following are general guidelines. Specific implementation may differ depending on end-requirements.*

The ESL Peripheral sample provided by Nordic uses a single code base and supports different boards and different display drivers. This approach is beneficial for maintenance and troubleshooting as, for example, you can always build the code on Nordic DevKit to exclude HW designs issues on the custom board. The application is abstracted so that, with minimal changes in config files and overlays, it is possible to add new boards and displays.

The current architecture allows for multiple drivers and multiple display sizes to be built on top of the same board (see nRF52833DK configurations and overlays). Pin assignments specific for the display should not be part of the display drivers nor of the board definitions, but rather part of overlays.

Hereafter we demonstrate this approach using off-tree drivers. Then we show also Zephyr driver approach, discussing Pros and Cons.

Note: EPD displays and drivers can be complex. Here, the goal is to provide an example / framework. Customers should rely on support from each EPD vendor to add drivers and to get corresponding tools to generate images.

### HW abstraction layers between application and display drivers

The ESL application is using some functions to control the display (e.g. `display_init`, `display_control`, `display_unassociated`, `display_associated`, `display_epd_onoff`, `display_clear_font`, `display_print_font`, `display_update_font`) that are abstracted from the HW. The display callbacks are defined in `esl\hw\esl_hw_impl`.

When adding a display drivers, you need to implement these callbacks using the driver functions. This is done in `peripheral_esl\hw\display\`, where you can see that there are several files doing this. If you add a new specific driver, you will need to create a new file of this kind to define how the ESL application uses the driver. Examples:

- `dummy_display.c` – for dummy display, nothing happen when display image change
- `led_display.c` – for simulated display on LEDs
- `epd_display.c` – for the Waveshare off-tree drivers defined in `\peripheral_esl\driver\EPD`
- `epd_shield_display.c` – for the [ssd16](#) display driver natively included in zephyr/nRF Connect SDK

Note: These files have some dependences also the HW, as they need to use specific pins to control the display. This can be abstracted quite easily using the “nodes”. For example, in `epd_display.c`, see the



following section.

```
/* nRF52 family Devkit has arduino header as SPI interface */
#if IS_ENABLED(CONFIG_DT_HAS_ARDUINO_HEADER_R3_ENABLED)
#define SPI_NODE DT_NODELABEL(arduino_spi)
/* Minew ms18f8 and stag85 use spi1 */
#elif IS_ENABLED(CONFIG_BOARD_MS138F7_NRF52833)
#define SPI_NODE DT_NODELABEL(spi1)
/* nRF54L Devkit uses SPI00 for now */
#elif IS_ENABLED(CONFIG_NRF54L_SPI00)
#define SPI_NODE DT_NODELABEL(spi00)
#else
#error "No SPI node found"
#endif /* CONFIG_DT_HAS_ARDUINO_HEADER_R3_ENABLED */

const struct device *spi = DEVICE_DT_GET(SPI_NODE);
PINCTRL_DT_DEFINE(SPI_NODE);
const struct pinctrl_dev_config *pcfg = PINCTRL_DT_DEV_CONFIG_GET(SPI_NODE);
static struct display_capabilities capabilities;
static struct display_buffer_descriptor buf_desc;

#if defined(CONFIG_ESL_POWER_PROFILE)
const struct gpio_dt_spec reset_gpio = GPIO_DT_SPEC_INST_GET(0, reset_gpios);
const struct gpio_dt_spec dc_gpio = GPIO_DT_SPEC_INST_GET(0, dc_gpios);
const struct gpio_dt_spec busy_gpio = GPIO_DT_SPEC_INST_GET(0, busy_gpios);
[...]
```

```
*(volatile uint32_t *) (DT_REG_ADDR(DT_NODELABEL(arduino_spi)) | 0xFFC) = 0;
*(volatile uint32_t *) (DT_REG_ADDR(DT_NODELABEL(arduino_spi)) | 0xFFC);
*(volatile uint32_t *) (DT_REG_ADDR(DT_NODELABEL(arduino_spi)) | 0xFFC) = 1;
```

## Off-tree drivers

One level down there are display drivers. The most elegant and modular approach presented here is the off-tree drivers used for the 4.2 inch display in this demo.

Nordic has ported the framework from Waveshare and adapted with small changes. In peripheral\_esl\driver\EPD, see the files as examples:

- EPD\_4in2\_V2.c and EPD\_4in2\_V2.h are specific drivers for the 4.2 inch display, implementing these functions:

- void EPD\_4IN2\_V2\_Init(void);
- void EPD\_4IN2\_V2\_Init\_Fast(void);/\*UBYTE Mode);\*/
- void EPD\_4IN2\_V2\_Init\_4Gray(void);
- void EPD\_4IN2\_V2\_Clear(void);
- void EPD\_4IN2\_V2\_Display(UBYTE \*Image);
- void EPD\_4IN2\_V2\_Display\_Fast(UBYTE \*Image);
- void EPD\_4IN2\_V2\_Display\_4Gray(UBYTE \*Image);
- void EPD\_4IN2\_V2\_PartialDisplay(UWORD x, UWORD y, UWORD w, UWORD l, UBYTE \*Image);
- void EPD\_4IN2\_V2\_Sleep(void);
- void EPD\_4IN2\_V2\_WriteDisplay(UWORD x, UWORD y, UWORD w, UWORD l, UBYTE \*Image);
- void EPD\_4IN2\_V2\_TurnOnDisplay\_Fast(void);

These files are taken from Waveshare ([see e-Paper subfolder](#)) and adapted slightly.

Note that these files are also abstracted: the low layers are kept out (as described here after)

- DEV\_Config.c and DEV\_Config.h implements the low layers. Also in these cases these are taken from Waveshare (see [Config subfolder](#)), but adapted to Nordic nRF Connect SDK. They implement:

```
void DEV_Digital_Write(UWORD Pin, UBYTE Value);
UBYTE DEV_Digital_Read(UWORD Pin);
void DEV_SPI_WriteByte(UBYTE Value);
void DEV_SPI_Write_nByte(uint8_t *pData, uint32_t Len);
void DEV_Delay_ms(UDOUBLE xms);
UBYTE DEV_Module_Init(void);
void DEV_Module_Exit(void);
```

These are the same for all the Waveshare display drivers.

Note: The pin assignment is done using overlays and Nodes as mentioned previously, and the mapping is done in DEV\_Config.c:

```
const EPD_Pin CS_PIN = {.gpio = GPIO_DT_SPEC_GET(DT_PARENT(DT_DRV_INST(0)), cs_gpios)};
const EPD_Pin RST_PIN = {.gpio = GPIO_DT_SPEC_INST_GET(0, reset_gpios)};
const EPD_Pin DC_PIN = {.gpio = GPIO_DT_SPEC_INST_GET(0, dc_gpios)};
const EPD_Pin BUSY_PIN = {.gpio = GPIO_DT_SPEC_INST_GET(0, busy_gpios)};
const EPD_Pin PWR_PIN = {.gpio = GPIO_DT_SPEC_INST_GET_OR(0, vcc_gpios, {})};
```

- CMakeLists.txt and Kconfig adds extra selections for build system.  
When selecting one of these displays at build time, e.g. using CONFIG\_EPD\_4IN2\_V2=y, the right driver is used.

Important to note that the display drivers above are very specific for each display size: see the name of the function “EPD\_4IN2\_V2\_Init”. To make it generic, this is solved in the upper layer: see again the peripheral\_esl\hw\display\ epd\_display.c file:

```
else if (IS_ENABLED(CONFIG_EPD_4IN2_V2)) {
    epd_display_fn.epd_init = EPD_4IN2_V2_Init_Fast;
    epd_display_fn.epd_clear = EPD_4IN2_V2_Clear;
```

```
epd_display_fn.epd_display_full = EPD_4IN2_V2_Display_Fast;
epd_display_fn.epd_write_display = EPD_4IN2_V2_WriteDisplay;
epd_display_fn.epd_turn_on_display = EPD_4IN2_V2_TurnOnDisplay_Fast;
epd_display_fn.epd_sleep = EPD_4IN2_V2_Sleep;
```

So, to sum up: `epd_display.c` implements call backs used by the ESL application. `EPD_4in2_V2.c` implements call backs used by `epd_display.c`. `DEV_Config.c` implements callbacks used by `EPD_4in2_V2.c`.

By using this structure, with minimal changes, it is easy to add a new display driver from WaveShare. See for example `EPD_1in54b_V2.c/.h` in `peripheral_esl\driver\EPD`. This driver is also using the same low layer functions defined implemented in `DEV_Config.c`, and the ESL application can still use the same `epd_display.c` to control the display.

Once added a new display driver, small changes may need to be made on the ESL application. (e.g. big screens may need different filesystem, or the application needs to know the width and height or how many images can be stored by that tag). This is done using some extra `*.conf` and `*.overlay`.

In `\peripheral_esl\conf\nrf52833dk_nrf52833`, check these files:

- `nrf52833dk_nrf52833_4in2_v2_epd.conf`  
for example, notice the flag `CONFIG_EPD_4IN2_V2=y` and the other parameters that may depend on the chosen display and display driver `CONFIG_ESL_DISPLAY_WIDTH=400`  
`CONFIG_ESL_DISPLAY_HEIGHT=300`  
`CONFIG_BT_ESL_IMAGE_MAX=2`  
`CONFIG_ESL_IMAGE_FILE_SIZE=15006`  
`CONFIG_ESL_IMAGE_BUFFER_SIZE=15006`  
etc
- `nrf52833dk_nrf52833_4in2_v2_epd.overlay`  
for example, notice how `cs-gpios`, `dc-gpios`, etc are defined

To tell the build system to pick the right file, we use the flag `CONFIG_EPD_4IN2_V2=y` which has an impact on several CMakeLists, for example:

- in `peripheral_esl\CMakeLists.txt`, we force the build system to use `epd_display.c`

```
if ((CONFIG_EPD_4IN2_V1) OR (CONFIG_EPD_4IN2_V2) OR (CONFIG_EPD_1IN54B_V2)
OR (CONFIG_EPD_2IN9B_V3) OR (CONFIG_EPD_2IN13B_V3) OR (CONFIG_EPD_2IN13_V4))
add_subdirectory(driver/EPD)
target_sources(app PRIVATE hw/display/epd_display.c)
endif()
```
- in `peripheral_esl\driver\EPD\CMakeLists.txt`, we specify which display driver to select (e.g. `EPD_4in2_V2.c`)

Note: If you need to port a completely new driver from a different vendor, a similar off-tree driver architecture can be used. In that case, you need a new file in `peripheral_esl\hw\display\` to implement the

application callbacks using the new driver, and a new folder in `peripheral_esl\driver\` with the new driver implementation.

Note: for LED, a similar HW abstracted approach is used. The Peripheral application is using the functions defined in `peripheral_esl\hw\led\led.c`. This file is then linked to the board definition (see the Nodes) and to the lower layers drivers (`#include <dk_buttons_and_leds.h>`). You can search for `DK_LED` in the application (including the service) to see where LEDs are used.

## Zephyr Display drivers

For the 2.13inch display built in “Section 2: How to install SDK and build the hex files used in this tutorial”, the native driver in zephyr is used for SSD16xx IC. The advantage of this is to reuse something provided already with the SDK, but the drawback is that this driver is not low power optimized and it is very rigid: different displays from different vendors, even if using the same IC driver, are very different due to some specific OTP settings. Hence, in general, it is not advised to use these drivers for ESL, but rather use off-tree drivers and get support from the display vendor.

Just for reference, you can see that in this case `CONFIG_SSD16XX=y` and `CONFIG_DISPLAY=y` are selected in the conf files (`nrf52833dk_nrf52833_b2in13_epd.conf`), plus some extra dts files are included in `nrf52833dk_nrf52833_b2in13_epd.overlay` to define some pins (`#include "waveshare_epaper_HINK-E0213-G01.dtsi"`—see `\peripheral_esl\dts` folder).

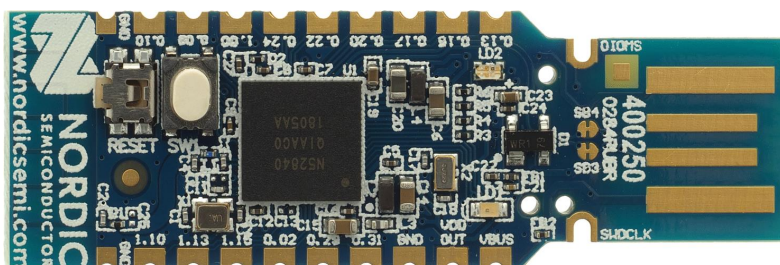
In this case, from the `peripheral_esl\CMakeLists.txt`, the `CONFIG_SSD16XX=y` flag force the build system to select the `epd_shield_display.c`. This file, as mentioned previously, implements the application callbacks by using the zephyr driver [ssd16](#).

## Creating your own tag - example

Assuming you have display drivers already in place and the application is working on the DevKit, let’s see now what it takes to create your own tag on a custom board.

For this exercise, instead of designing and building the board from scratch, we use the [nRF52840Dongle](#) ( [spec here](#)) and we will connect some pins to the WaveShare shield with the 4.2inch display. The overall HW will be named: “mytag\_nRF52840”

To allow the dongle to be powered from external regulated source (e.g. coin cell or PPK2, instead of USB), you can cut SB2 and Solder SB1 on the back of the dongle, and power the dongle from VDD OUT (1.8V to 3.6). For programming, we use SWD interface. Spi0 is used for SPI communication to the display.





52840 DONGLE	NRF52 FUNCTION	CONNECTION TO WAVESHARE SHIELD	WAVESHARE SHEILD
VDD_OUT		VDD_IO	Power input
GND		GND	Ground
0.13	SCLK	SCLK/D13	SPI clock input
0.15	MISO	MISO/D12	SPI data output
0.17	MOSI	MOSI/D11	SPI data input
0.20	EPD_CS	D10	e-Paper chip selection
0.22	EPD_DC	D9	e-Paper data/command
0.24	EPD_RST	D8	e-Paper reset
0.09 *	EPD_BUSY	D7	e-Paper busy

\*P 0.09 is by default assigned to NFC. So, we need to add this flag (CONFIG\_NFCT\_PINS\_AS\_GPIO=y) to tell the build system to use this GPIO as a normal GPIO. In a real board, avoid using this pin. In general, always see Pin Definition in the SoC spec before assigning pins. (e.g. in nRF52840, some pins should not be used for high speed communication, such as SPI)

#### NOTE:

The nRF52840 Dongle is meant to be as rapid prototyping platform and for demo/PoC purposes. For Production, instead, it is suggested to design a new PCB and use a SoC or a 3<sup>rd</sup> party Module. To design and build your own board, you can start from the ref design and documentation in the “**HW Development**” in “Section 4: Links to documentation, spec and training material”. Please note that, for PAwR, if you want to achieve low power, you must have an accurate low frequency xtal (see Nordic ref designs).

#### Step1: Boad definition

Once you have at least the schematic of your board, you need to create a “board definition” (software description of the board) that will be used by the build system in the Nordic toolchain in conjunction with the application code. In “Section 4: Links to documentation, spec and training material”), there links to doc and training about custom board definitions.

In this case, for the board mentioned above (nRF52840 Dongle + Waveshare shield + 4.2inch EPD), the board definition is provided here: \peripheral\_esl\boards\arm\mytag\_nRF52840

This is based on the schematic of the dongle (e.g. see button0) and on the pin assignment table above (e.g. SPI). However, note that in the board definition we have not included some pins (e.g. EPD\_DC, EPD\_RST, EPD\_BUSY). We define those in the overlay, so it is consistent with the code structure used in the DevKit (this approach brings more flexibility and modularity on the code side, in case different drivers are used).

#### Step2: Match the application with the board

Make little changes to the application code to adapt to the board configuration.

Example:

- in the display driver `epd_display.c`, the `Arduino_spi` node is used. The SPI used in this `mytag_nRF52840` is instead `spi0`. To add this to the driver, one way is to add the following lines in `red`

```
#if IS_ENABLED(CONFIG_DT_HAS_ARDUINO_HEADER_R3_ENABLED)
#define SPI_NODE DT_NODELABEL(arduino_spi)
/* Minew ms18f8 and stag85 use spi1*/
#elif IS_ENABLED(CONFIG_BOARD_MS138F7_NRF52833)
#define SPI_NODE DT_NODELABEL(spi1)
/* nRF54L Devkit uses SPI00 for now */
#elif IS_ENABLED(CONFIG_NRF52840_SPI00)
#define SPI_NODE DT_NODELABEL(spi00)
#else
#define SPI_NODE DT_PARENT(DT_DRV_INST(0))
#endif
#if !defined(SPI_NODE)
#error "No SPI node found"
#endif /* CONFIG_DT_HAS_ARDUINO_HEADER_R3_ENABLED */
```

and changing these to select `SPI_NODE`

```
*(volatile uint32_t *) (DT_REG_ADDR(SPI_NODE) | 0xFFC) = 0;
*(volatile uint32_t *) (DT_REG_ADDR(SPI_NODE) | 0xFFC);
*(volatile uint32_t *) (DT_REG_ADDR(SPI_NODE) | 0xFFC) = 1;
```

Make sure “`#define DT_DRV_COMPAT generic_epd`” is present in the file as well.

#### - Buttons

In the DevKit, two buttons are used by the ESL application. `Button0` for unassociating the tag (as debug feature) and `Button1` is used for exiting shipping mode.

- `Button0` is used in the `peripheral_esl/src/main.c` (search for “`DK_BTN1_MSK`”). Since this lib is used ([here](#)), this is compatible with `Button0`, as described in the [documentation](#). `Button 0` pin is defined in the board definition.
- `Button1` can be changed by a config (`CONFIG_ESL_WAKE_GPIO`). In this example, we can disabled the shipping mode, so this button does not need to be specified

#### - LEDs

in the DevKit, there are 4 LEDs used by the ESL application. In this board, there is `LED1` (green) and `1x RGB LED2` defined. “`peripheral_esl/hw/led.c`” is where LEDs are used. Since custom board definition (see `mytag_nrf52840.dts`) respect naming convention “`gpio-leds`” as in the DevKit, the application does not need any change.

Note: `led.c` can also support pwm leds.



### Step3: Board configuration

Create a board configuration folder. In this case, we have copied the most relevant files from the nRF52833dk\_nrf52833 conf folder (e.g. the ones used when building the application with 4.2inc as described in “Section 2: How to install SDK and build the hex files used in this tutorial”), renamed them, and placed them into this folder \peripheral\_esl\conf\mytag\_nRF52840. Here there are many things that can be changed depending on your wishes and depending on your board. Example:

- in the mytag\_nRF52840\_4.in2\_v2\_epd.conf , we changed some parameters:  
CONFIG\_BT\_DIS\_PNP\_PID=0x1234  
CONFIG\_BT\_DIS\_MODEL="Nordic\_ESL\_MY\_TAG"  
CONFIG\_BT\_DEVICE\_NAME="Nordic\_ESL\_MY\_TAG"  
and added:  
CONFIG\_NFCT\_PINS\_AS\_GPIOS=y  
Depending on your board components and architecture, there are several other parameters that can be changed here (e.g. display size, LED number, images to be stored, etc)
- in mytag\_nRF52840\_4.in2\_v2\_epd.overlay, we changed the four pins (CS, DC, RESET, BUSY), matching with the board pins in our custom board (table above). Note, we also commented out the definitions in app.overlay. To learn more on overlay and device tree, see this [“how to” documentation](#)

Note that:

- other generic parameters are defined in prj.conf / prj\_release.conf  
(e.g. you may probably want to change also VID, Device Manufacturer, etc in prj.conf)
- changing some parameters may break the Bluetooth ESL spec. If unsure, ask Nordic.

### Step 4: Build the application

Build the code. For example by using:

```
west build --build-dir build_MYTAG_52840_4_2_EPD -p -b mytag_nrf52840 -- -
DOVERLAY_CONFIG="mytag_nrf52840_4in2_v2_epd.conf" -
DDTC_OVERLAY_FILE="conf/mytag_nrf52840/mytag_nrf52840_4in2_v2_epd.overlay" -Dmcuboot_DTS_ROOT="
C:/nrf-esl-bluetooth/samples/peripheral_esl/" -DBOARD_ROOT="/"
```

To Flash the hex into your custom board, you can see “APPENDIX G: Flashing a nRF52-based custom board”.

What we have showed so far is a starting point to add more features, in similar way as done in the nRF52833DevKit, with minimal or zero changes in the code base. You can get inspired by the following configurations from the nRF52833DevKit configuration folder:

- to add DFU on external flash, refer to the “nrf52833dk\_nrf52833\_b4in2\_epd.conf” and “nrf52833dk\_nrf52833\_b4in2\_epd.overlay”



- to make the application low power, see power\_profiler.conf and power\_profiler.overlay
- to use littlefs file system (e.g. if you use larger display images, and cannot be buffered at once in RAM while received from the AP), refer to “nrf52833dk\_nrf52833\_4in2\_epd\_lfs.conf” and “nrf52833dk\_nrf52833\_4in2\_epd\_lfs.overlay”.

#### Step 5: test with the AP

Testing the tag with the AP is the next step. If you use manual on-boarding, no changes to the AP are needed. If you use automatic onboarding and you want the images to be transferred during the configuration phase, a little change is needed to the AP. Indeed, in automatic onboarding, the AP checks the PID of the tag it is trying to associate with and, based on that, it decides which image to send during the configuration (before PAST). This is defined in \service\esl\_client.c: search for “static void esl\_c\_auto\_configuring\_tag(uint8\_t conn\_idx)”. Here you can add a case for your new tag (in this example, PID=0x1234).

```
case 0x1234:
    LOG_INF("My_TAG");
    LOG_INF("image D");
    qk_data.tag_img_idx = 0;
    qk_data.img_idx = 0xD;
    ret = k_msgq_put(&kimage_worker_msgq, &qk_data, K_NO_WAIT);
    if (ret) {
        LOG_ERR("No space in the queue for qk_data");
    }

    LOG_INF("image F");
    qk_data.tag_img_idx = 1;
    qk_data.img_idx = 0xF;
    ret = k_msgq_put(&kimage_worker_msgq, &qk_data, K_NO_WAIT);
    if (ret) {
        LOG_ERR("No space in the queue for qk_data");
    }

    break;
```

As you can see, the AP will send Image 0xD and Image 0xF to this tag during configuration while auto-onboarding.

Naturally, you need to make sure the display images are have been previously transferred from your PC to the AP:





```
mcumgr --conntype="serial" --connstring="COM4,baud=115200,mtu=512" fs upload  
mytag_displayimage0 /ots_image/esl_image_0D
```

```
mcumgr --conntype="serial" --connstring="COM4,baud=115200,mtu=512" fs upload mytag_displayimage1  
/ots_image/esl_image_0F
```

(for this example, for **mytag\_displayimage0** and **mytag\_displayimage1** you can use the images as with the 4.2inc example)

RTT logs is enable in this case and can be used to test the application once flashed into the board. To test power consumption, you need to build the code together with power\_profiler.conf and power\_profiler.overlay.

#### Step 6: Going to production

if you use MCUboot, before going to production, you must change the MCUBoot Key (i.e. never use Nordic development keys in your final product). Power consumption options needs to be also enabled.

There also other debug features that may be removed (such as the button to unassociate the tag, the simulated display on LED, and the status indication LED).

Beside this, for production, in case the tag is in unassociated state, some customers may want to keep the latest display image (instead of printing debug text as in the demo). To do that, it can be enough to “return” immediately inside the “void display\_unassociated(uint8\_t disp\_idx)” in the peripheral\_esl\hw\display\\*\*\*\*\*\_display.c file. You may also add a Watchdog.

## APPENDIX E: Alternative ways to transfer images from PC to AP

### Bulk upload SCRIPT using MCUMgr

In previous sections we have listed some commands to transfer the display images from PC to AP. For example, for out of the box demo in “Section 1: Getting started with ESL using pre-compiled hex and automatic onboarding”, we suggested:

```
mcumgr --conntype="serial" --connstring="COM4,baud=115200,mtu=512" fs upload  
esl_image_bt /ots_image/esl_image_00  
  
mcumgr --conntype="serial" --connstring="COM4,baud=115200,mtu=512" fs upload  
esl_image_nordic /ots_image/esl_image_01  
  
mcumgr --conntype="serial" --connstring="COM4,baud=115200,mtu=512" fs upload  
esl_image_empower /ots_image/esl_image_09  
  
mcumgr --conntype="serial" --connstring="COM4,baud=115200,mtu=512" fs upload  
esl_image_mcu_block /ots_image/esl_image_0A
```

The idea is to pre-load several images into the AP, and then the AP will transfer them to the ESL tags, depending on their PID. For tags with PID 0x5482, images 00 and 01 will be sent automatically to the tags during first configuration. For tags with PID 0x5488, images 09 and 0A. More details on this is at the end of “APPENDIX D: Adding new display drivers and creating your TAG application”.

However, for demo purposes, if several different types of tags are to be handled, a python script can be used to replace the commands above.

Inside the GUI folder, ap\_smp\_util.py can be used for this:

- (one off) download newtmgr (e.g. newtmgr.exe) and add paste the executable in the GUI folder (from here, e.g. for windows: [apache-mynewt-newtmgr-bin-windows-1.12.0](https://github.com/NordicSemiconductor/FreeRTOS-AP/blob/master/FreeRTOS-AP/FreeRTOS-AP-Tools/newtmgr-bin-windows-1.12.0) )
- first modify the “image\_list.txt” file accordingly to the images you want to transfer from PC to AP and to their locations in the file system of the AP.
- then run the python script:  
python ap\_smp\_util.py -A set

Note:

- Images on the AP can be changed at run time using this script or the other methods. Then the new images can be transferred to the tags while on a ACL connection using specific shell commands
- ap\_smp\_util.py has several other useful functions for the ESL project. Check the README.md file in the GUI folder to learn more.

### Alternative to MCUMgr Client: AuTerm

AuTerm is a new tool listed in Zephyr [mcumgr tools](#) that can be used as alternative to the MCUMgr client (Go Language) mentioned above. One of the advantages of this tool is that it is faster in transferring the files, and it supports file system SMP Client used in this tutorial.

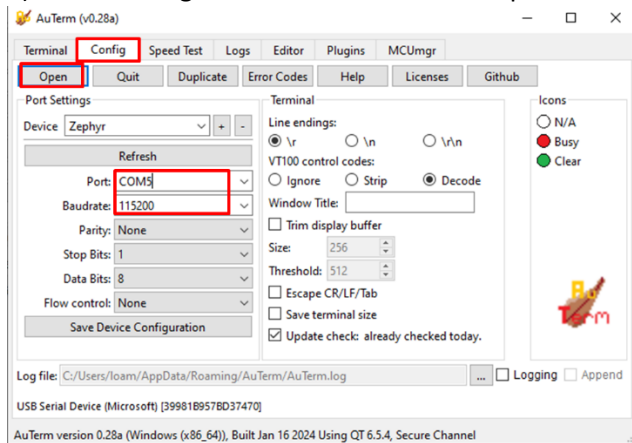
Example: to transfer a display image from PC to AP the following procedure is equivalent to the following command used in previous sections

```
mcumgr --conntype="serial" --connstring="COM5,baud=115200,mtu=512" fs upload IMAGE_FILE_NAME_0 /ots_image/esl_image_00
```

1) download and launch AuTerm;

here some precompiled releases <https://github.com/thedjnK/AuTerm/releases> (click on Assets)

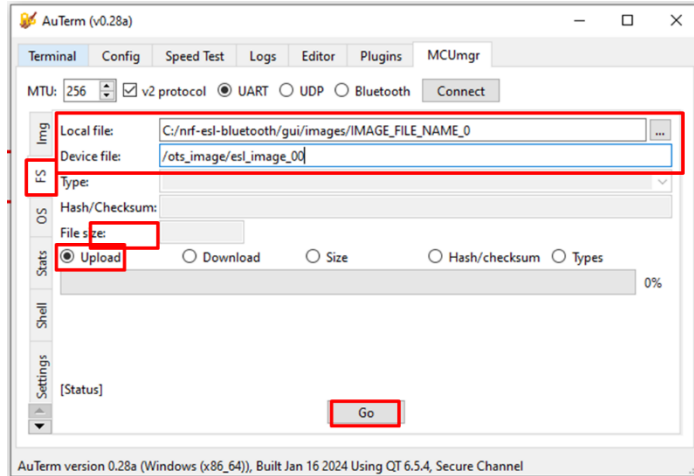
2) Select the right Port and then click on Open



3) select MCUMgr on top bar and FS on the left bar. Then specify:

- Local file path on the PC (e.g. C:\nrf-esl-bluetooth\gui\images\IMAGE\_FILE\_NAME\_0)
- Device file path on the AP (e.g. /ots\_image/esl\_image\_00)

Finally click on Go.



(you can also Connect/Close COM connection directly from this window)

Documentation and source code is here <https://github.com/thedjnK/AuTerm>

## APPENDIX F: Measuring current on nRF52833 DevKit with PPK2

Before doing the actual measurements, two setups for measuring currents with a PPK2 (in source mode) on a Devkit are proposed:

### STANDARD SET-UP (requires cutting a trace):

Prepare the 52833DK (TAG) for Power Consumption measurements as follows:

- [Cut SB40 on nRF52833DK](#) and make sure the switch on the nRF52833DK is ON  
NOTE: if you cut the trace, remember to use a jumper when not using the PPK2, e.g. when flashing the DevKit
- Connect the PPK2 (documentation [here](#)) to the PC and to the nRF52833DK as described in the figure hereafter. (make sure the switch on the PPK2 is ON)

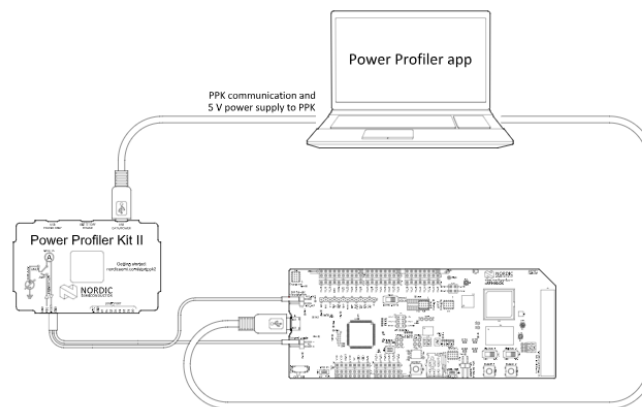
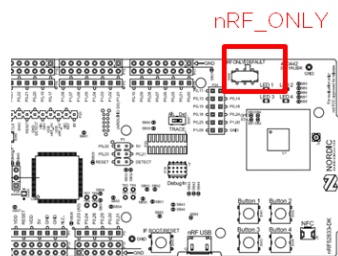


Figure 1. Measuring current in Source Meter mode

PPK2	DK
VOUT	P22 VDD_nRF
GND	P21

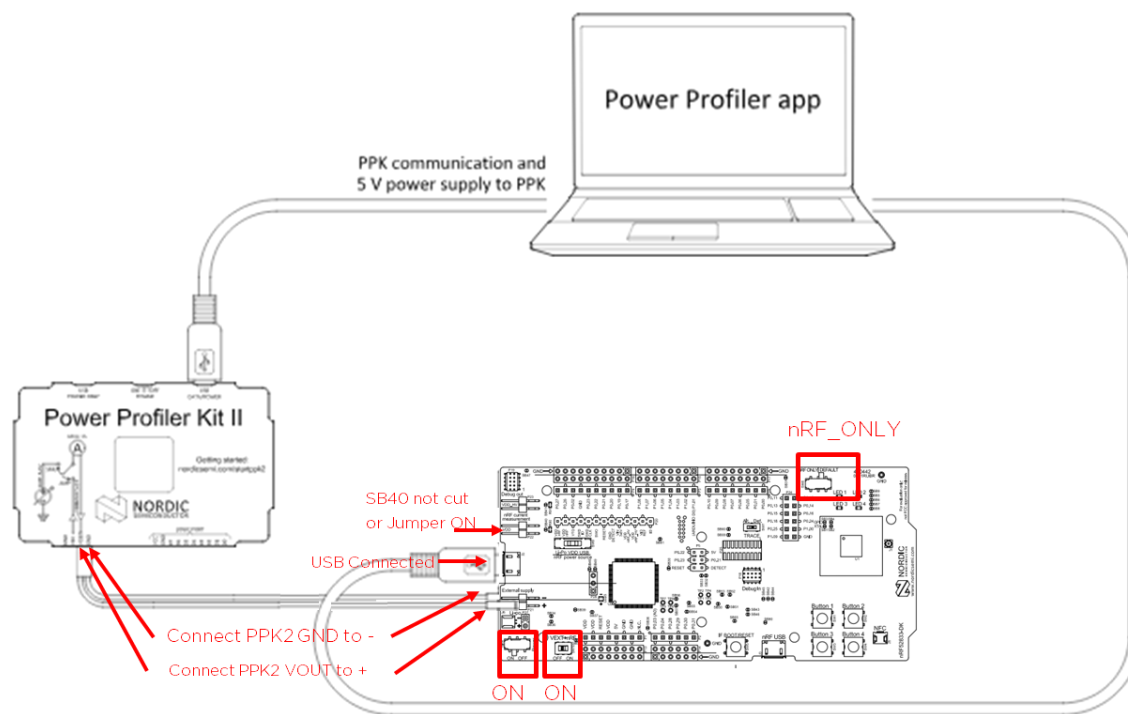
Table 1. Pin connections in Source Meter mode

- Set the SW6 switch to “nRF\_ONLY”. SW10 is always OFF in this configuration (standard)



### ALTERNATIVE SET-UP (it does not require cutting a trace):

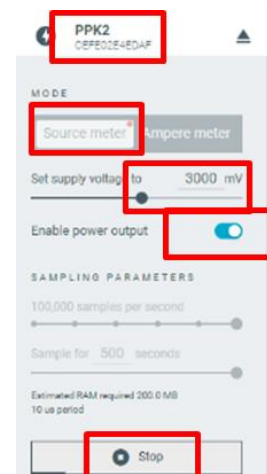
Prepare the 52833DK (TAG) for Power Consumption measurements as in the picture:



### PERFORM THE MEASUREMENT

After choosing one of the two approaches, trigger the measurement:

- Open “Power Profiler” app in the nRF Connect for Desktop software.
- Select the PPK2 on the top left corner
- Configure PPK2 in “Source Meter” mode, “Set supply voltage” to **3000mV** and click on “Enable power output” to enable power.
- Click on “Start” to start measurement.
- Push button 2 in the nRF52833DevKit to exit Ship Mode (LED will still not work after exiting ship mode (see notes below); you should be able to see it exited ship mode from the currents.)



- f. If the nRF52833DevKit was previously associated by the AP, and the AP database has not been erased and AP is still actively scanning, the AP will scan and sync this tag again automatically.  
Otherwise, to start from scratch, you can push button 1 on the nRF52833DevKit and button 2 on the AP to start the association process again (as mentioned in step 9 - in the “Section 1: Getting started with ESL using pre-compiled hex and automatic onboarding”)
- g. Once synchronized to the AP, current consumption should now be less than 5uA



#### Notes:

- In both set-ups, SW6 is set to nRF ONLY. So nRF52 SoC will be isolated from some of the DK components in order to measure only the nRF52833 current properly. This means: **LEDs will stop working**. However, from the PPK 2, you will see the pulses from the radio, so it is easy to see if radio is advertising or it is synchronized to PAwR pulses, or sending a response. If you have shield with EPD display, TAG can still receive and process commands from the AP to change display image.
- Keep USB plugged-in while doing measurements to avoid leakages between SoC and other components on the board
- To flash/program the DevKits, put back the switches in normal position (SW6 in DEFAULT and SW10 in OFF)
- If there is a shield, provide only 3V from PPK2 (even if nRF52 has wider voltage range) to avoid voltage mismatch between shield and nRF52 GPIOs – see next points
- Currents of shield is not included in the measurement. To include currents in the measurement:

- On the back of the shield, cut the trace between VCC and IOREF.



- Solder a jumper between VCC (back of the shield shield) and connect to VDD\_nRF in the Dev Kit: in this way the shield is powered from VDD\_nRF, same as per the nRF52 SoC



- Now also the shield is powered by the PPK2 (i.e. same voltage level as per the nRF52 GPIOs). It is possible to change the PPK2 voltage (according to the supplier spec: 1.8V to 3.6V for nRF52833; for the shield, refer to supplier)
- Note: Shield is not optimized for low power (it includes level shifter and extra components that draw extra currents). When you design your custom board, check with your epaper supplier for most optimized configuration to achieve low power



## APPENDIX G: Flashing a nRF52-based custom board

To flash your application to a custom board with custom connections, you can use a nRF52 DK as described here.

Assumptions:

- Custom board is powered independently with VCC 3V source (e.g. coin cell or PPK2)
- Custom board needs to expose at least: SWD IO, SWD CLK, VCC, GND
- DevKit is in default settings

Connect:

- GND from custom board to a GND pin on DevKit and make sure GND\_DETECT (if available in P20, typically the eighth pin from the left) is also grounded (note, some nRF2xxx DK do not have GND\_DETECT)
- VCC from custom board to SWD\_VTG on DevKit (third pin from left in P20) (Important: it is important custom board is powered with VCC=3V: SWD\_VTG will sense the 3V coming from the custom board. This will make sure that the external target is flashed, and not the SoC on the nRF52xxx DK)
- SWD IO from custom board to SWD\_IO on the DevKit (fourth pin from left in P20)
- SWD CLK from custom board to SWD\_CLK on the DevKit (fifth pin from left in P20)

Here the connections:

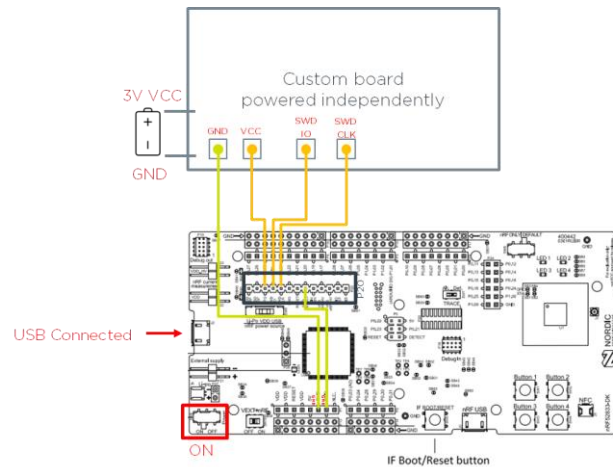


Figure 2: Interface MCU

Alternatively, if you want to power the custom board from the DevKit, you can use VDD\_nRF (first pin from left in P20) to power the board (VCC – 3V). Remember to connect also VCC to SWD\_VTG as in the previous case. Make sure to remove the battery/power supply from the custom board, in this case.

Here the connections:

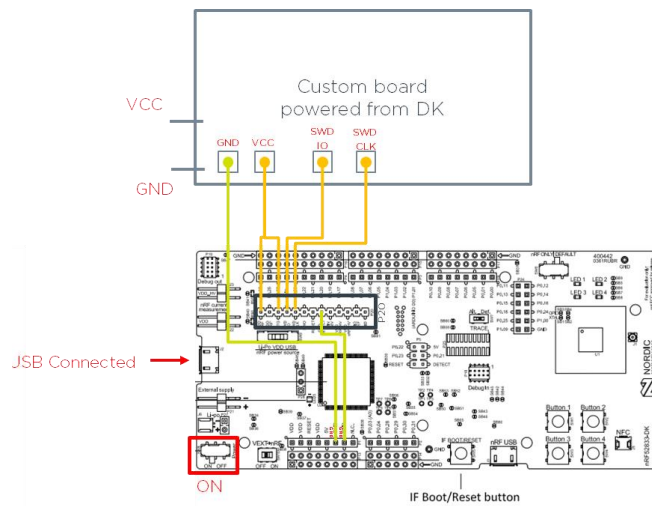


Figure 2: Interface MCU

For additional info, go to [devkit documentation](#)

Once the connections are done between the nRF52xxx DK (programmer) and the custom board (target), in order to flash, use the option available in VS Code or use Programmer app in nRF Connect for Desktop (make sure to re-select the board on top left corner) or nrfjprog (APPENDIX A: Flash DevKits using CLI(nrfjprog)). **IMPORTANT:** in this case, after programming the board, power cycle it.