28.02.2018

Development with GCC and Eclipse

Troubleshooting guide

Contents

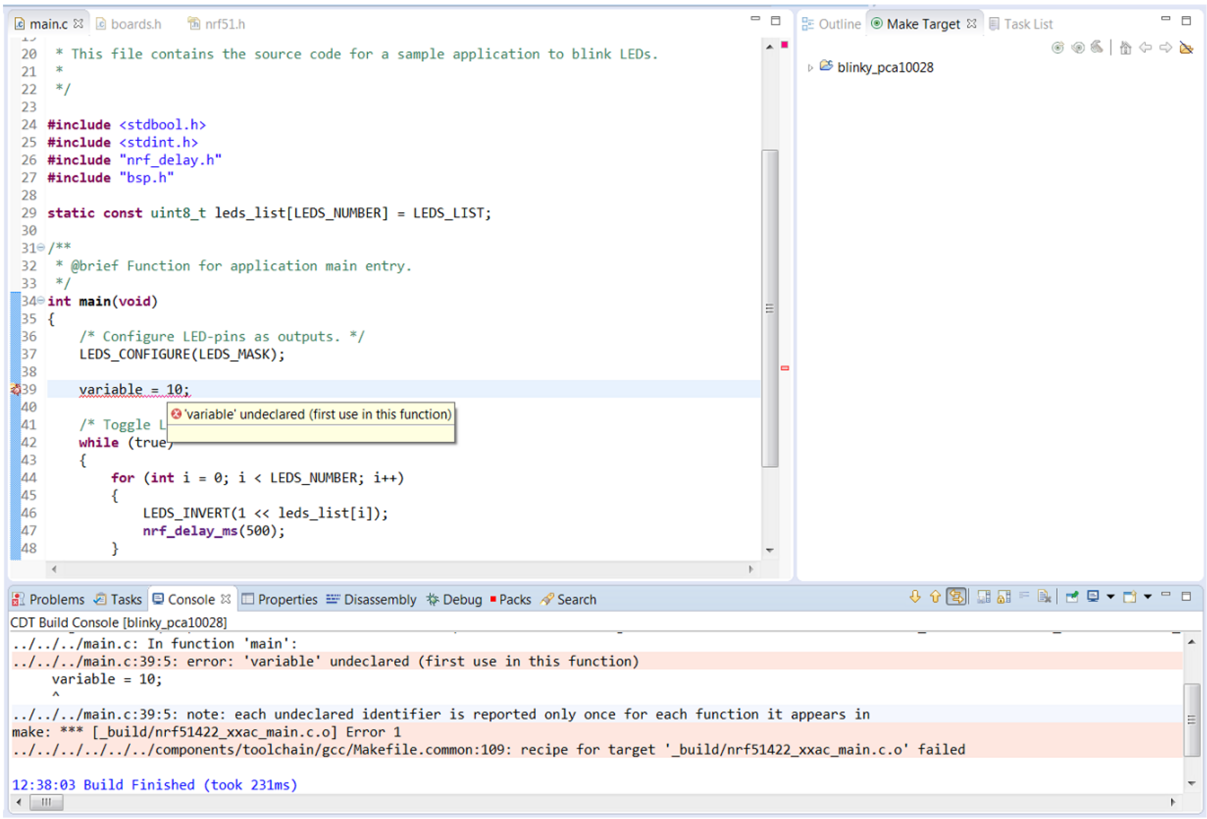
Introduction

This is a brief guide on how to troubleshoot some of the common problems related to Eclipse and GCC setup with the Nordic SDK. I recommend doing a quick search on the forum and in the comment section if you experience a problem not mentioned in this guide, you may find others who have encountered the exact same problem, but do not hesitate to post a new question on the forum if not.

Build errors

The *CDT Build Console* window in Eclipse will show errors or warnings reported by the build process if there are any. It is important to note the difference between errors reported here, and errors that are only shown in the code editor; Eclipse may show errors in the source code despite there being no build errors. If this is the case it likely means that there is a problem with the *auto discovery* configuration mentioned in the “*Enable auto discovery of symbols, include paths and compiler settings”* section of the tutorial. Please skip to the next chapter if this is the symptom.

Below image illustrates an actual build error caused by undeclared variable in the source code. In this case, Eclipse is correctly identifying this coding mistake based on the feedback from the build process (build output).



```
 * This file contains the source code for a sample application to blink LEDs.
 *
 */

#include <stdbool.h>
#include <stdint.h>
#include "nrf_delay.h"
#include "bsp.h"

static const uint8_t leds_list[LEDS_NUMBER] = LEDS_LIST;

/**
 * @brief Function for application main entry.
 */
int main(void)
{
    /* Configure LED-pins as outputs. */
    LEDS_CONFIGURE(LEDS_MASK);

    variable = 10;

    /* Toggle L
    while (true
    {
        for (int i = 0; i < LEDS_NUMBER; i++)
        {
            LEDS_INVERT(1 << leds_list[i]);
            nrf_delay_ms(500);
        }
```

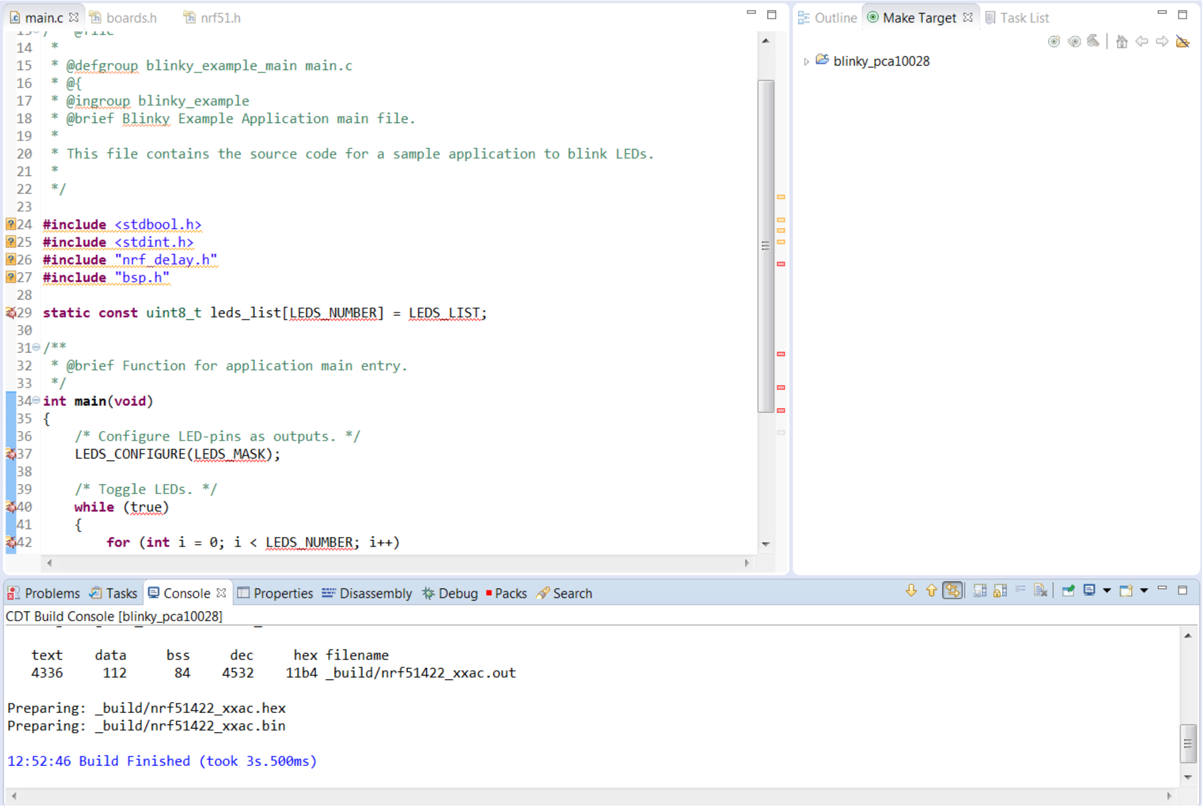
'variable' undeclared (first use in this function)

CDT Build Console [blinky_pca10028]

```
../../../main.c: In function 'main':
../../../main.c:39:5: error: 'variable' undeclared (first use in this function)
     variable = 10;
     ^
../../../main.c:39:5: note: each undeclared identifier is reported only once for each function it appears in
make: *** [_build/nrf51422_xxac_main.c.o] Error 1
../../../../../../components/toolchain/gcc/Makefile.common:109: recipe for target '_build/nrf51422_xxac_main.c.o' failed

12:38:03 Build Finished (took 231ms)
```

Figure 1: Build error: referencing an undeclared variable.

Figure 2 on the other hand shows an example where the project is built without errors, but unable to resolve the macros and variables because of missing include paths and pre-processor definitions, which indicates a problem with the *auto discovery* configuration.



```c
 *
 * @defgroup blinky_example_main main.c
 * @{
 * @ingroup blinky_example
 * @brief Blinky Example Application main file.
 *
 * This file contains the source code for a sample application to blink LEDs.
 *
 */

#include <stdbool.h>
#include <stdint.h>
#include "nrf_delay.h"
#include "bsp.h"

static const uint8_t leds_list[LEDS_NUMBER] = LEDS_LIST;

/**
 * @brief Function for application main entry.
 */
int main(void)
{
    /* Configure LED-pins as outputs. */
    LEDS_CONFIGURE(LEDS_MASK);

    /* Toggle LEDs. */
    while (true)
    {
        for (int i = 0; i < LEDS_NUMBER; i++)
```

```
   text    data     bss     dec     hex filename
   4336     112      84    4532    11b4 _build/nrf51422_xxac.out

Preparing: _build/nrf51422_xxac.hex
Preparing: _build/nrf51422_xxac.bin

12:52:46 Build Finished (took 3s.500ms)
```

Figure 2: Successful build without errors.

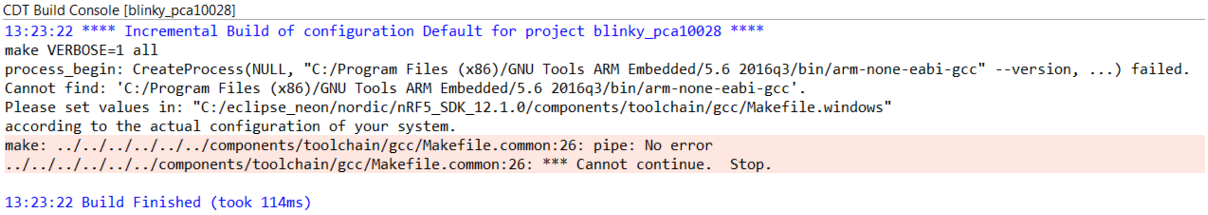
Wrong search paths

A common cause of build errors is that Eclipse fails to invoke the build tools (GNU make, mkdir or rm) or that GNU Make fails to invoke the toolchain because of wrong search paths. To troubleshoot these kinds of errors we need to analyze the output in the CDT Build Console.

Toolchain path

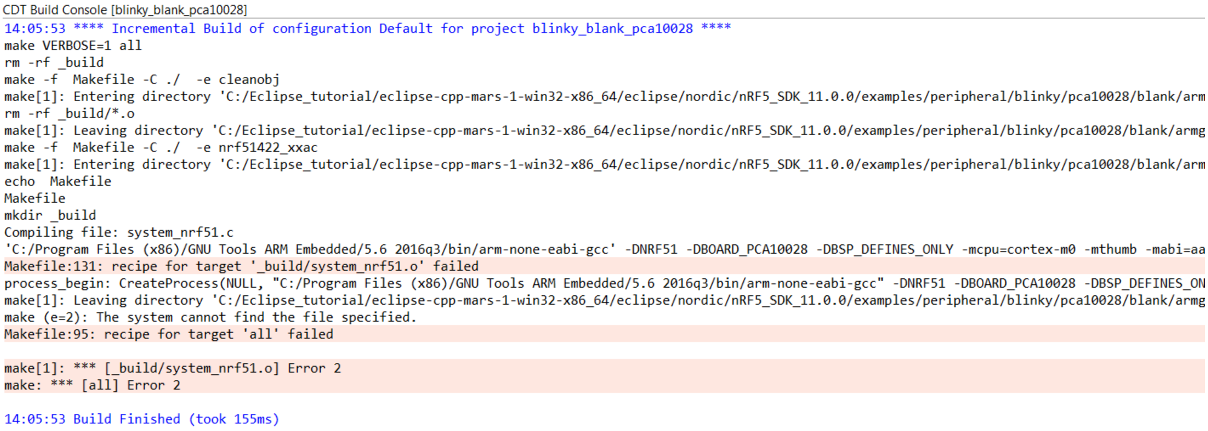
The GNU_INSTALL_ROOT variable declared in Makefile.windows/posix must point to the install directory for the GCC toolchain. Please refer to the "*before we begin*" section for more details.

Figure 3 and 4 shows the build output when the GNU_INSTALL_ROOT variable is pointing to a non-existent directory. Edit the Makefile.windows or Makefile.posix file (depending on OS) in $(SDK_ROOT)/components/toolchain/gcc so that GNU_INSTALL_ROOT corresponds with the version you have installed if you are seeing this.


CDT Build Console [blinky_pca10028]

```
13:23:22 **** Incremental Build of configuration Default for project blinky_pca10028 ****
make VERBOSE=1 all
process_begin: CreateProcess(NULL, "C:/Program Files (x86)/GNU Tools ARM Embedded/5.6 2016q3/bin/arm-none-eabi-gcc" --version, ...) failed.
Cannot find: 'C:/Program Files (x86)/GNU Tools ARM Embedded/5.6 2016q3/bin/arm-none-eabi-gcc'.
Please set values in: "C:/eclipse_neon/nordic/nRF5_SDK_12.1.0/components/toolchain/gcc/Makefile.windows"
according to the actual configuration of your system.
make: ../../../../../../components/toolchain/gcc/Makefile.common:26: pipe: No error
../../../../../../components/toolchain/gcc/Makefile.common:26: *** Cannot continue.  Stop.

13:23:22 Build Finished (took 114ms)
```

*Figure 3: console ouput with SDK version 12 and later where the toolchain path is not set correctly. Should have been C:/Program Files (x86)/GNU Tools ARM/**5.4** 2016q3/bin/arm-none-eabi-gcc to correspond with this particular setup.*

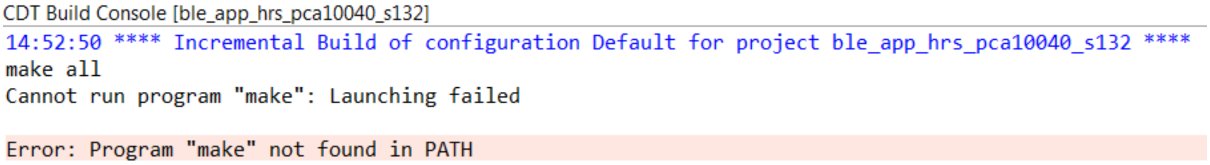


```
CDT Build Console [blinky_blank_pca10028]
14:05:53 **** Incremental Build of configuration Default for project blinky_blank_pca10028 ****
make VERBOSE=1 all
rm -rf _build
make -f  Makefile -C ./  -e cleanobj
make[1]: Entering directory 'C:/Eclipse_tutorial/eclipse-cpp-mars-1-win32-x86_64/eclipse/nordic/nRF5_SDK_11.0.0/examples/peripheral/blinky/pca10028/blank/arm
rm -rf _build/*.o
make[1]: Leaving directory 'C:/Eclipse_tutorial/eclipse-cpp-mars-1-win32-x86_64/eclipse/nordic/nRF5_SDK_11.0.0/examples/peripheral/blinky/pca10028/blank/armg
make -f  Makefile -C ./  -e nrf51422_xxac
make[1]: Entering directory 'C:/Eclipse_tutorial/eclipse-cpp-mars-1-win32-x86_64/eclipse/nordic/nRF5_SDK_11.0.0/examples/peripheral/blinky/pca10028/blank/arm
echo  Makefile
Makefile
mkdir _build
Compiling file: system_nrf51.c
'C:/Program Files (x86)/GNU Tools ARM Embedded/5.6 2016q3/bin/arm-none-eabi-gcc' -DNRF51 -DBOARD_PCA10028 -DBSP_DEFINES_ONLY -mcpu=cortex-m0 -mthumb -mabi=aa
Makefile:131: recipe for target '_build/system_nrf51.o' failed
process_begin: CreateProcess(NULL, "C:/Program Files (x86)/GNU Tools ARM Embedded/5.6 2016q3/bin/arm-none-eabi-gcc" -DNRF51 -DBOARD_PCA10028 -DBSP_DEFINES_ON
make[1]: Leaving directory 'C:/Eclipse_tutorial/eclipse-cpp-mars-1-win32-x86_64/eclipse/nordic/nRF5_SDK_11.0.0/examples/peripheral/blinky/pca10028/blank/armg
make (e=2): The system cannot find the file specified.
Makefile:95: recipe for target 'all' failed

make[1]: *** [_build/system_nrf51.o] Error 2
make: *** [all] Error 2

14:05:53 Build Finished (took 155ms)
```

Figure 4: same as shown figure 3, but with SDK releases prior to version 12.

Path to build tools

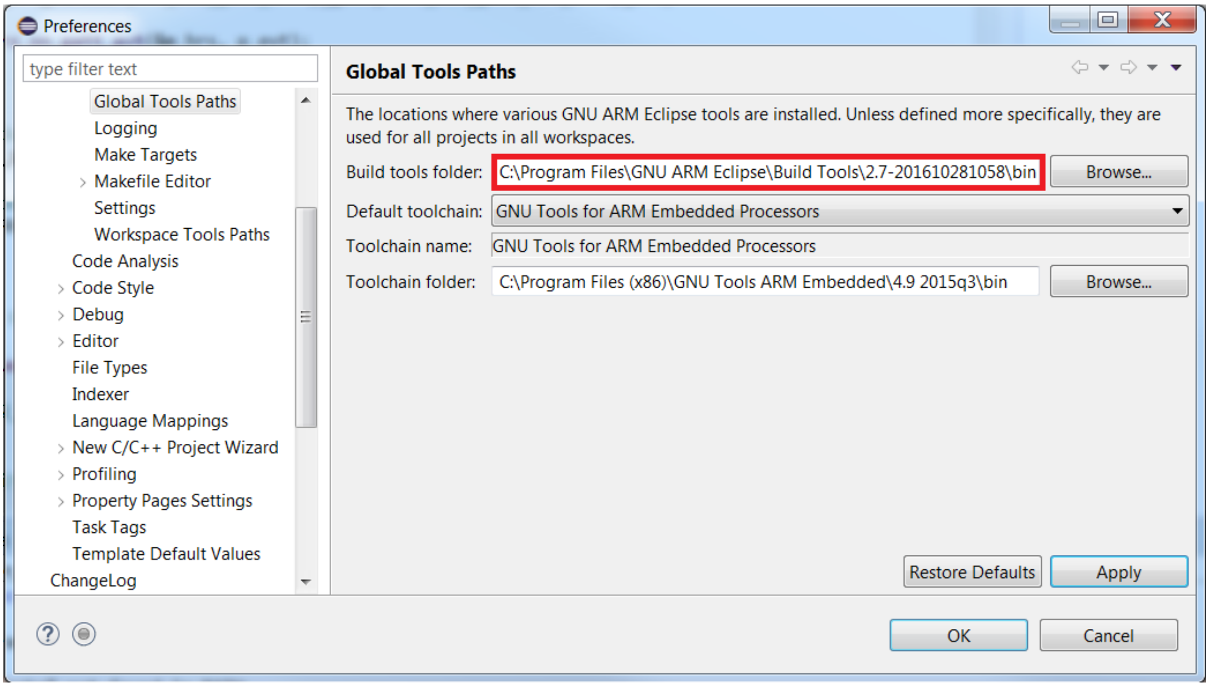
Errors such as "*Cannot run program "make"*, *Cannot run program "mkdir",* etc. typically means that Eclipse is not finding the executable in provided search paths.



```
CDT Build Console [ble_app_hrs_pca10040_s132]
14:52:50 **** Incremental Build of configuration Default for project ble_app_hrs_pca10040_s132 ****
make all
Cannot run program "make": Launching failed

Error: Program "make" not found in PATH
```

Figure 5: GNU Make is found in path

First, make sure that GNU make is installed on the system as explained in the *«before we begin»* section, then verify that the *Build tools folder* field contains the correct path in Eclipse preferences.



Global Tools Paths

The locations where various GNU ARM Eclipse tools are installed. Unless defined more specifically, they are used for all projects in all workspaces.

Build tools folder: C:\Program Files\GNU ARM Eclipse\Build Tools\2.7-201610281058\bin

Default toolchain: GNU Tools for ARM Embedded Processors

Toolchain name: GNU Tools for ARM Embedded Processors

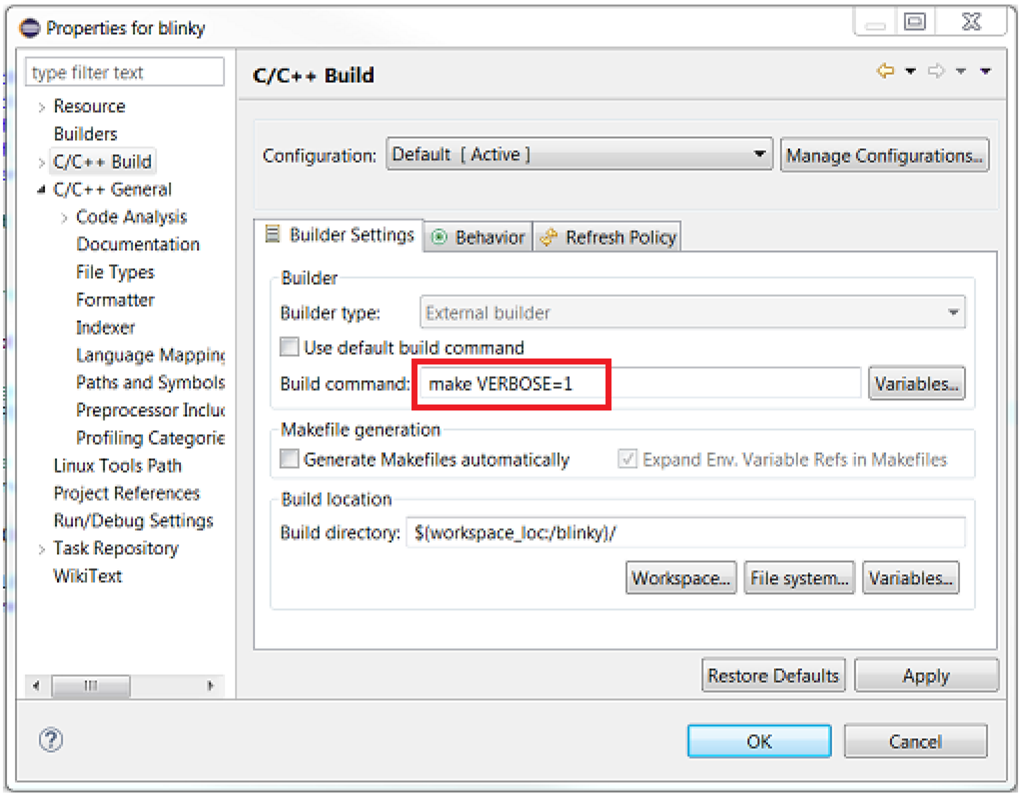
Toolchain folder: C:\Program Files (x86)\GNU Tools ARM Embedded\4.9 2015q3\bin

Figure 6: Add path to build tools folder.

Cannot run program "": Launching failed

This error may occur if the build command is not set, or contains an empty variable (e.g., if ${cross_make} is not set).

Make sure to set the Build command to *make VERBOSE=1* in project properties.



Configuration: Default [Active]

Builder type: External builder

□ Use default build command

Build command: make VERBOSE=1

□ Generate Makefiles automatically ☑ Expand Env. Variable Refs in Makefiles

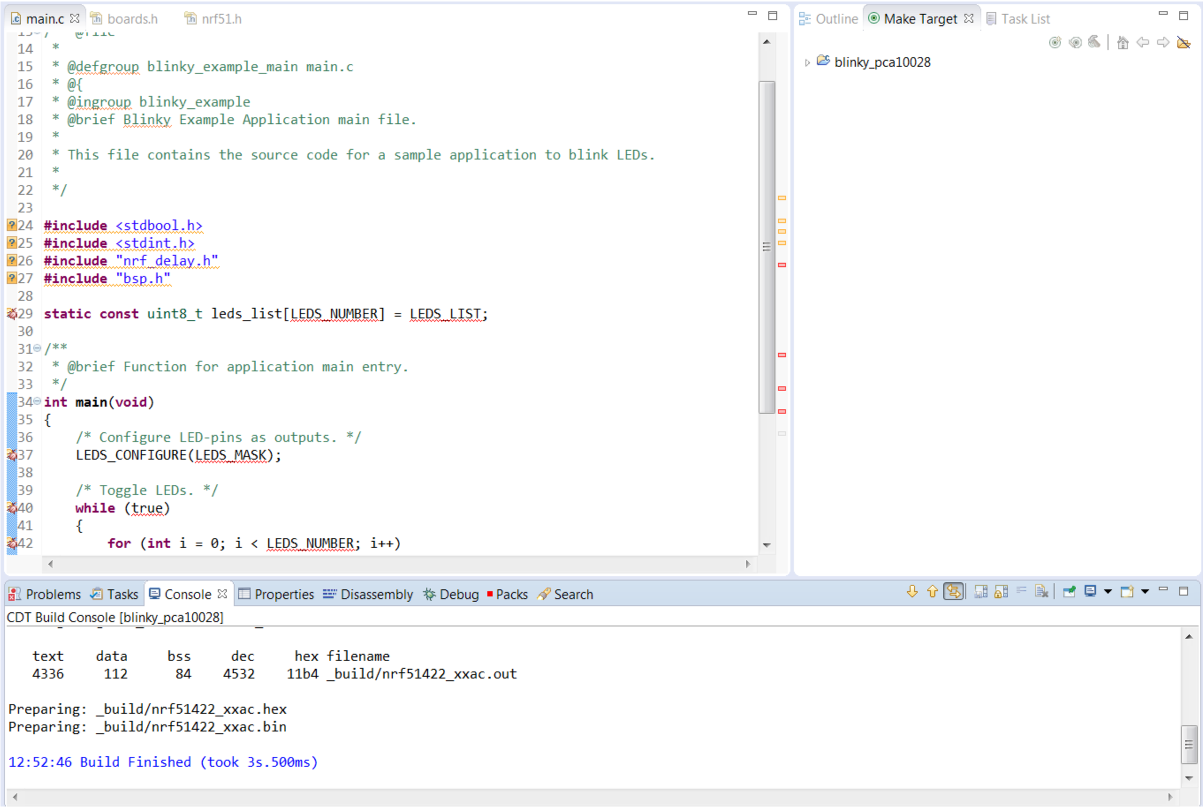
Build directory: ${workspace_loc:/blinky}/

Figure 7: Set build command.

Auto discovery of symbols, include paths and compiler settings

Eclipse's built-in build output parser enables auto discovery of symbols and include paths for Makefile managed projects so you do not have to add them manually when creating new or modifying existing projects. Unfortunately, it does not provide much feedback when configured incorrectly, besides failing to collect information from the build output.

Screenshot below shows an example where *Auto discovery* has not been enabled properly. Notice how the project built successfully according to the *CDT Build Console* yet there are multiple errors shown in the code view.



```
 *
 * @defgroup blinky_example_main main.c
 * @{
 * @ingroup blinky_example
 * @brief Blinky Example Application main file.
 *
 * This file contains the source code for a sample application to blink LEDs.
 *
 */

#include <stdbool.h>
#include <stdint.h>
#include "nrf_delay.h"
#include "bsp.h"

static const uint8_t leds_list[LEDS_NUMBER] = LEDS_LIST;

/**
 * @brief Function for application main entry.
 */
int main(void)
{
    /* Configure LED-pins as outputs. */
    LEDS_CONFIGURE(LEDS_MASK);

    /* Toggle LEDs. */
    while (true)
    {
        for (int i = 0; i < LEDS_NUMBER; i++)
```

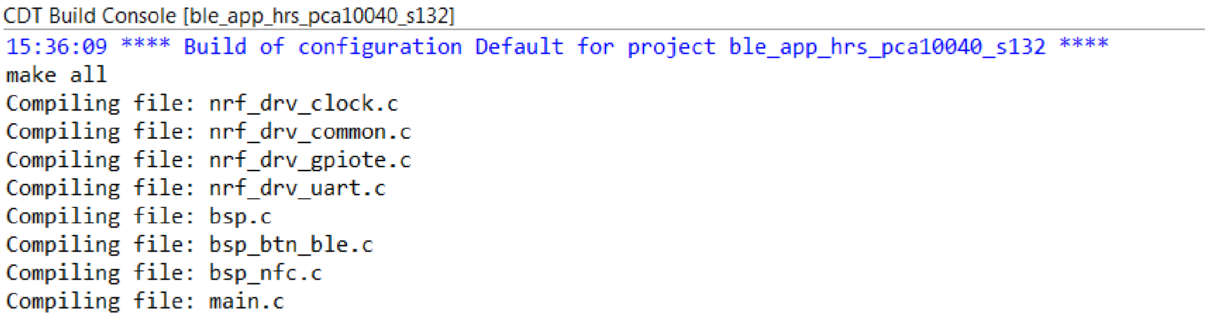
CDT Build Console [blinky_pca10028]

```
   text    data     bss     dec     hex filename
   4336     112      84    4532    11b4 _build/nrf51422_xxac.out

Preparing:  _build/nrf51422_xxac.hex
Preparing:  _build/nrf51422_xxac.bin

12:52:46 Build Finished (took 3s.500ms)
```

Figure 8: Build output is not parsed.

Verbose build logging not enabled

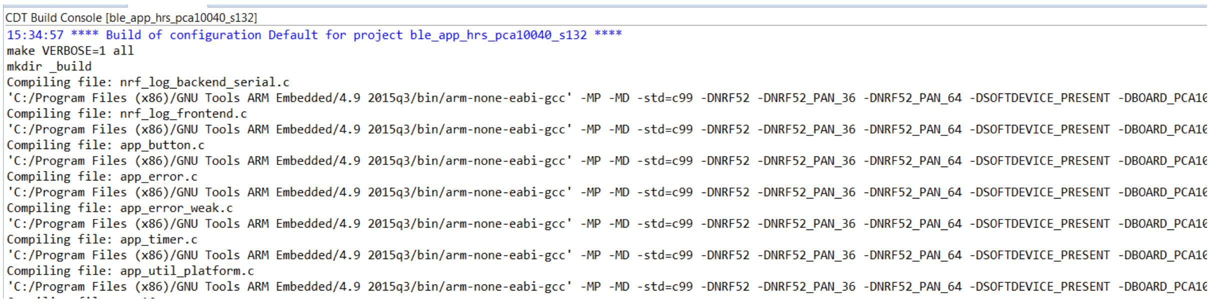
The build output parser relies on verbose build log to extract relevant information. Check if build log contains enough information to retrieve necessary information about the project. The SDK makefile have an option for producing verbose build output, which is enabled by setting the ‘VERBOSE’ makefile variable to ‘1’. Screenshots below illustrates the difference.



CDT Build Console [ble_app_hrs_pca10040_s132]

```
15:36:09 **** Build of configuration Default for project ble_app_hrs_pca10040_s132 ****
make all
Compiling file: nrf_drv_clock.c
Compiling file: nrf_drv_common.c
Compiling file: nrf_drv_gpiote.c
Compiling file: nrf_drv_uart.c
Compiling file: bsp.c
Compiling file: bsp_btn_ble.c
Compiling file: bsp_nfc.c
Compiling file: main.c
```

Figure 9: verbose build log not enabled.



CDT Build Console [ble_app_hrs_pca10040_s132]

```
15:34:57 **** Build of configuration Default for project ble_app_hrs_pca10040_s132 ****
make VERBOSE=1 all
mkdir _build
Compiling file: nrf_log_backend_serial.c
'C:/Program Files (x86)/GNU Tools ARM Embedded/4.9 2015q3/bin/arm-none-eabi-gcc' -MP -MD -std=c99 -DNRF52 -DNRF52_PAN_36 -DNRF52_PAN_64 -DSOFTDEVICE_PRESENT -DBOARD_PCA1
Compiling file: nrf_log_frontend.c
'C:/Program Files (x86)/GNU Tools ARM Embedded/4.9 2015q3/bin/arm-none-eabi-gcc' -MP -MD -std=c99 -DNRF52 -DNRF52_PAN_36 -DNRF52_PAN_64 -DSOFTDEVICE_PRESENT -DBOARD_PCA1
Compiling file: app_button.c
'C:/Program Files (x86)/GNU Tools ARM Embedded/4.9 2015q3/bin/arm-none-eabi-gcc' -MP -MD -std=c99 -DNRF52 -DNRF52_PAN_36 -DNRF52_PAN_64 -DSOFTDEVICE_PRESENT -DBOARD_PCA1
Compiling file: app_error.c
'C:/Program Files (x86)/GNU Tools ARM Embedded/4.9 2015q3/bin/arm-none-eabi-gcc' -MP -MD -std=c99 -DNRF52 -DNRF52_PAN_36 -DNRF52_PAN_64 -DSOFTDEVICE_PRESENT -DBOARD_PCA1
Compiling file: app_error_weak.c
'C:/Program Files (x86)/GNU Tools ARM Embedded/4.9 2015q3/bin/arm-none-eabi-gcc' -MP -MD -std=c99 -DNRF52 -DNRF52_PAN_36 -DNRF52_PAN_64 -DSOFTDEVICE_PRESENT -DBOARD_PCA1
Compiling file: app_timer.c
'C:/Program Files (x86)/GNU Tools ARM Embedded/4.9 2015q3/bin/arm-none-eabi-gcc' -MP -MD -std=c99 -DNRF52 -DNRF52_PAN_36 -DNRF52_PAN_64 -DSOFTDEVICE_PRESENT -DBOARD_PCA1
Compiling file: app_util_platform.c
'C:/Program Files (x86)/GNU Tools ARM Embedded/4.9 2015q3/bin/arm-none-eabi-gcc' -MP -MD -std=c99 -DNRF52 -DNRF52_PAN_36 -DNRF52_PAN_64 -DSOFTDEVICE_PRESENT -DBOARD_PCA1
```

Figure 10: verbose build output enabled.

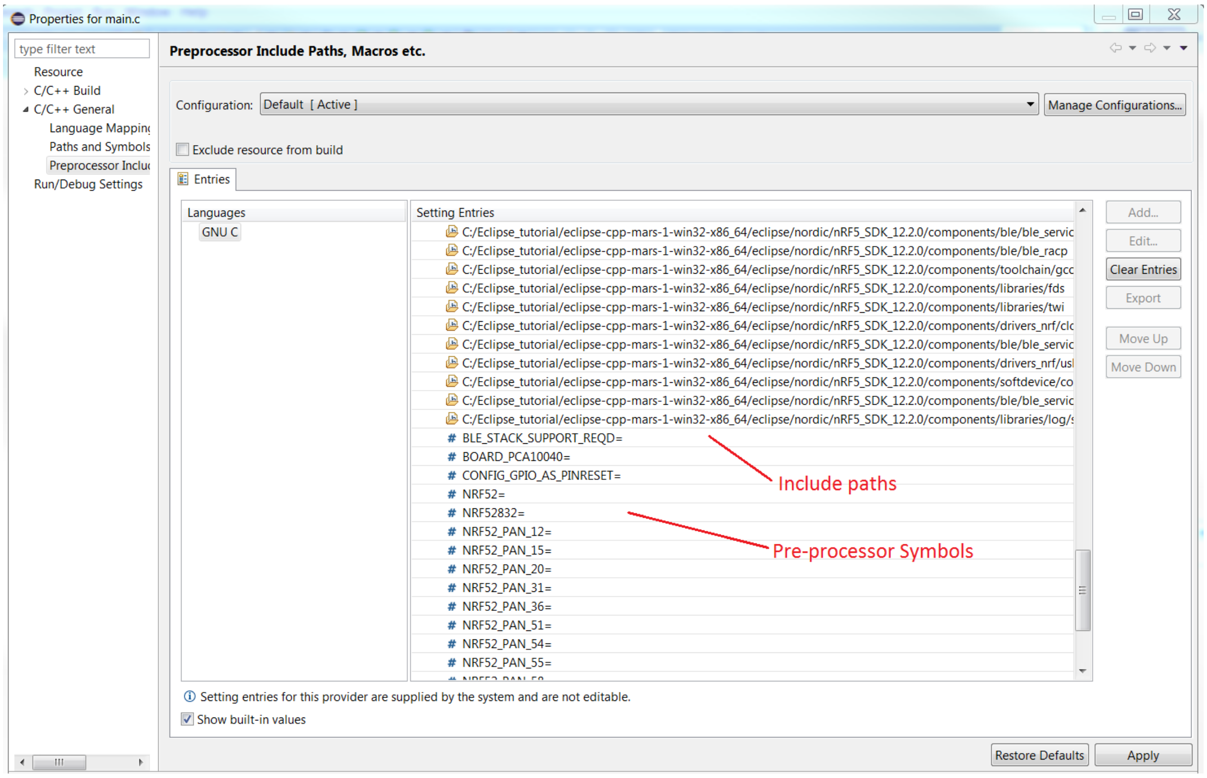
Change the build command as shown in Figure 7 if verbose build output is not enabled. Assuming everything else related to the discovery feature is configured correctly, it should be sufficient to do a clean build followed by a rebuild of the index (right click on project and click index->rebuild) to resolve the errors.

Note, with SDK version 12 or newer it is necessary to use the patched Makefile.common file included in the tutorial to work around what looks to be a limitation with the build output parser.

Partial discovery

The output parser may in some cases discover the include paths, but not the preprocessor symbols. This will also lead to unresolved errors in Eclipse. Verify that the build is using the patched Makefile.common file attached at the end of the tutorial.

It is possible to inspect the include paths and symbols that where added by the build output parser as shown in Figure 11 (select properties for one of the source files. This can be compared against the Makefile to determine what is missing.



Properties for main.c

Preprocessor Include Paths, Macros etc.

Configuration: Default [Active]

☐ Exclude resource from build

Languages: GNU C

Setting Entries

```
C:/Eclipse_tutorial/eclipse-cpp-mars-1-win32-x86_64/eclipse/nordic/nRF5_SDK_12.2.0/components/ble/ble_servic
C:/Eclipse_tutorial/eclipse-cpp-mars-1-win32-x86_64/eclipse/nordic/nRF5_SDK_12.2.0/components/ble/ble_racp
C:/Eclipse_tutorial/eclipse-cpp-mars-1-win32-x86_64/eclipse/nordic/nRF5_SDK_12.2.0/components/toolchain/gcc
C:/Eclipse_tutorial/eclipse-cpp-mars-1-win32-x86_64/eclipse/nordic/nRF5_SDK_12.2.0/components/libraries/fds
C:/Eclipse_tutorial/eclipse-cpp-mars-1-win32-x86_64/eclipse/nordic/nRF5_SDK_12.2.0/components/libraries/twi
C:/Eclipse_tutorial/eclipse-cpp-mars-1-win32-x86_64/eclipse/nordic/nRF5_SDK_12.2.0/components/drivers_nrf/clc
C:/Eclipse_tutorial/eclipse-cpp-mars-1-win32-x86_64/eclipse/nordic/nRF5_SDK_12.2.0/components/ble/ble_servic
C:/Eclipse_tutorial/eclipse-cpp-mars-1-win32-x86_64/eclipse/nordic/nRF5_SDK_12.2.0/components/drivers_nrf/usl
C:/Eclipse_tutorial/eclipse-cpp-mars-1-win32-x86_64/eclipse/nordic/nRF5_SDK_12.2.0/components/softdevice/co
C:/Eclipse_tutorial/eclipse-cpp-mars-1-win32-x86_64/eclipse/nordic/nRF5_SDK_12.2.0/components/ble/ble_servic
C:/Eclipse_tutorial/eclipse-cpp-mars-1-win32-x86_64/eclipse/nordic/nRF5_SDK_12.2.0/components/libraries/log/
# BLE_STACK_SUPPORT_REQD=
# BOARD_PCA10040=
# CONFIG_GPIO_AS_PINRESET=
# NRF52=
# NRF52832=
# NRF52_PAN_12=
# NRF52_PAN_15=
# NRF52_PAN_20=
# NRF52_PAN_31=
# NRF52_PAN_36=
# NRF52_PAN_51=
# NRF52_PAN_54=
# NRF52_PAN_55=
```

Include paths

Pre-processor Symbols

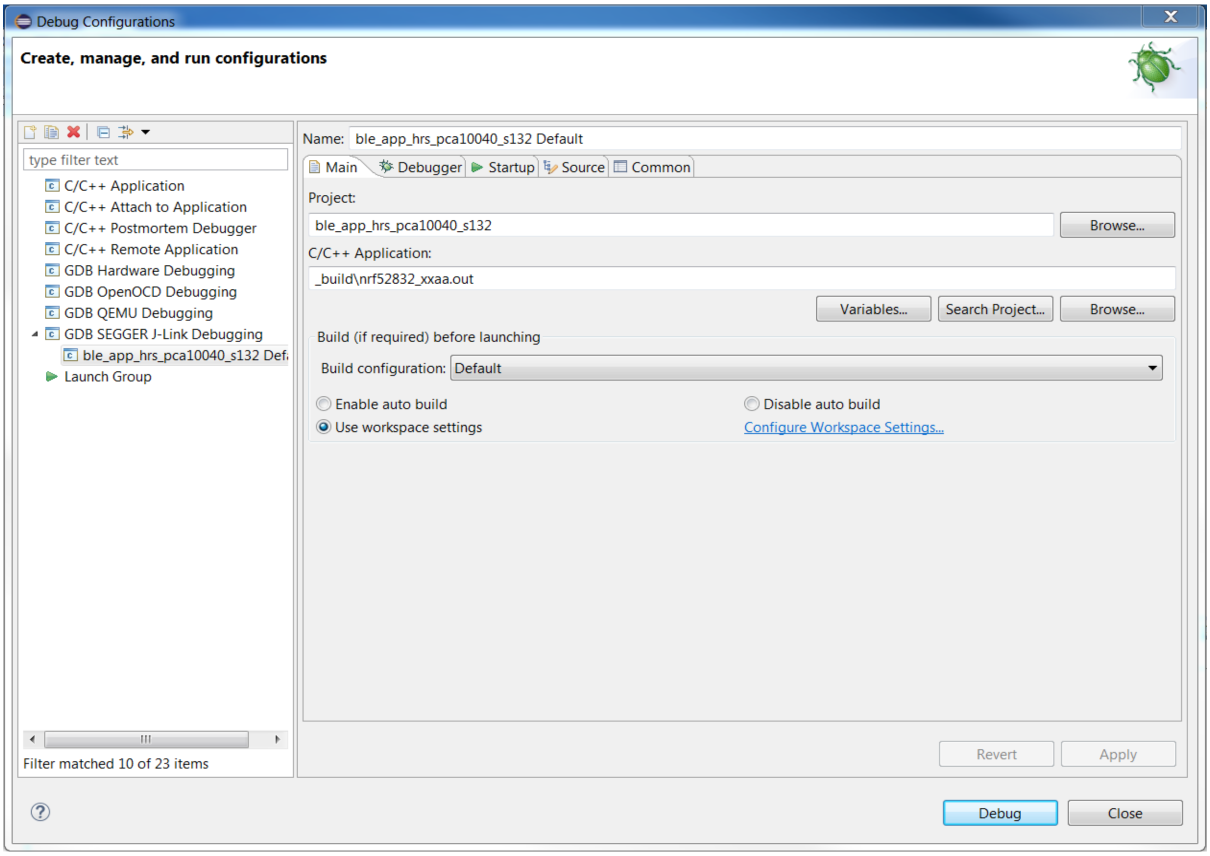
ⓘ Setting entries for this provider are supplied by the system and are not editable.

☑ Show built-in values

Figure 11: Inspect include paths and symbols added by output parser.

Cannot open debug session

Below are some screenshots showing the debug configuration for the nRF52 series, but it would be the same for an nrf51 device expect from the device name. Please go through the configurations shown here, and make sure they correspond with your configuartion.



Debug Configurations

Create, manage, and run configurations

Name: ble_app_hrs_pca10040_s132 Default

Project: ble_app_hrs_pca10040_s132

C/C++ Application: _build\nrf52832_xxaa.out

Build (if required) before launching

Build configuration: Default

○ Enable auto build

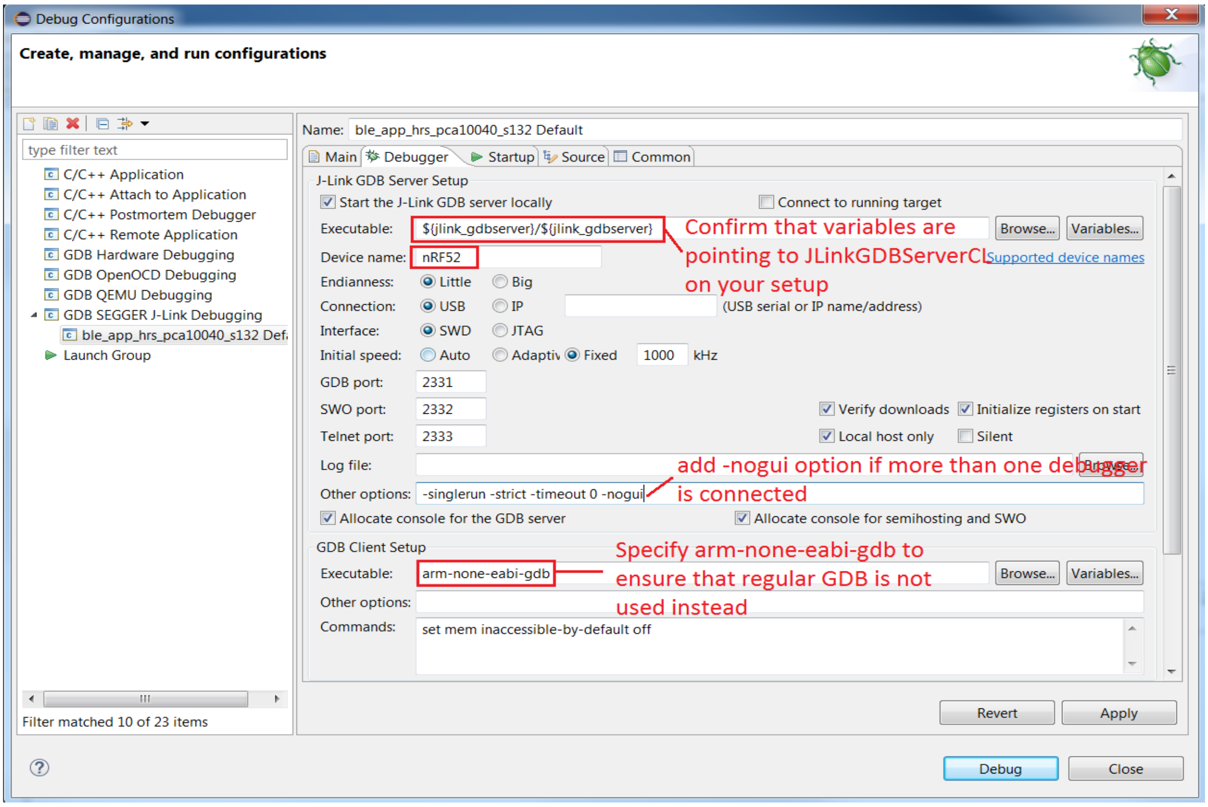
○ Disable auto build

● Use workspace settings

Configure Workspace Settings...

Filter matched 10 of 23 items

*Figure 12: Main tab - check that *.out file is set in C/C++ Application field*



Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- GDB Hardware Debugging
- GDB OpenOCD Debugging
- GDB QEMU Debugging
- GDB SEGGER J-Link Debugging
  - ble_app_hrs_pca10040_s132 Def
- Launch Group

Filter matched 10 of 23 items

Name: ble_app_hrs_pca10040_s132 Default

Main | Debugger | Startup | Source | Common

J-Link GDB Server Setup

☑ Start the J-Link GDB server locally ☐ Connect to running target

Executable: ${jlink_gdbserver}/${jlink_gdbserver}

Confirm that variables are pointing to JLinkGDBServerCL on your setup

Device name: nRF52 Supported device names

Endianness: ● Little ○ Big

Connection: ● USB ○ IP (USB serial or IP name/address)

Interface: ● SWD ○ JTAG

Initial speed: ○ Auto ○ Adaptiv ● Fixed 1000 kHz

GDB port: 2331

SWO port: 2332 ☑ Verify downloads ☑ Initialize registers on start

Telnet port: 2333 ☑ Local host only ☐ Silent

Log file:

Other options: -singlerun -strict -timeout 0 -nogui

add -nogui option if more than one debugger is connected

☑ Allocate console for the GDB server ☑ Allocate console for semihosting and SWO

GDB Client Setup

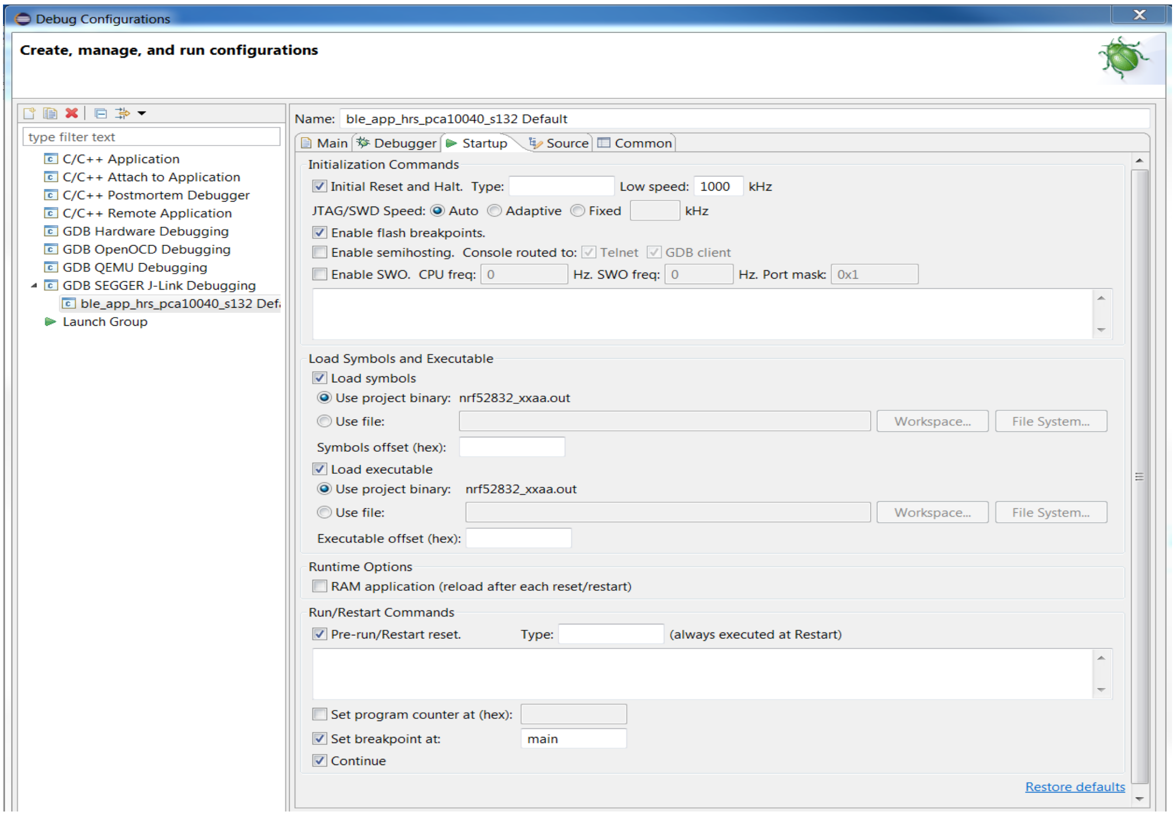
Executable: arm-none-eabi-gdb

Specify arm-none-eabi-gdb to ensure that regular GDB is not used instead

Other options:

Commands: set mem inaccessible-by-default off

Figure 13: Debugger configuration



Debug Configurations

Create, manage, and run configurations

Name: ble_app_hrs_pca10040_s132 Default

Main | Debugger | Startup | Source | Common

Initialization Commands

☑ Initial Reset and Halt. Type: Low speed: 1000 kHz

JTAG/SWD Speed: ● Auto ○ Adaptive ○ Fixed kHz

☑ Enable flash breakpoints.

☐ Enable semihosting. Console routed to: ☑ Telnet ☑ GDB client

☐ Enable SWO. CPU freq: 0 Hz. SWO freq: 0 Hz. Port mask: 0x1

Load Symbols and Executable

☑ Load symbols

● Use project binary: nrf52832_xxaa.out

○ Use file:

Symbols offset (hex):

☑ Load executable

● Use project binary: nrf52832_xxaa.out

○ Use file:

Executable offset (hex):

Runtime Options

☐ RAM application (reload after each reset/restart)

Run/Restart Commands

☑ Pre-run/Restart reset. Type: (always executed at Restart)

☐ Set program counter at (hex):

☑ Set breakpoint at: main

☑ Continue

Figure 14: Startup configuration - semihosting and SWO are enabled by default, but not implemented in code examples.

Please post a new question on the forum that includes the error message(s) if does not work with the settings shown above.

Application will not run

Project builds without errors, but the FW does not work as expected after being loaded to the chip.

Error in linker configuration (memory layout)

nRF5x series ICs comes in several different memory variants while the SDK examples are only configured for those used on the development kits. Typical symptom of incorrect memory layout is that a hardfault exception occurs before the program reaches *main*. An overview of the different variants can be found on the infocenter.

Verify that RAM and ROM settings in the. ld file invoked by the makefile are in accordance to the IC variant used on target board.

Example of a typical memory layout for the nRF51x22_xx**AC** (256/32) variant:

```
MEMORY

{

 /*FLASH and RAM ORIGIN depends on softdevice series and version. Check SD release
notes for appropriate values. Set ORIGIN to 0x0 and 0x20000000 for examples that
do not run on top of softdevice stack or similar */

 FLASH (rx) : ORIGIN = 0x1B000, LENGTH = 0x25000

 RAM (rwx) :  ORIGIN = 0x20002000, LENGTH = 0x6000

}
```

And the same for nRF51x22_xx**AA** (256/16):

```
MEMORY

{

 FLASH (rx) : ORIGIN = 0x1B000, LENGTH = 0x25000

 RAM (rwx) :  ORIGIN = 0x20002000, LENGTH = 0x2000

}
```

As a side note, development kits includes the optional LF crystal, which is enabled by default in the SDK examples. This is often not the case with custom boards and modules due to cost/size constraints. Remember to use the internal LF oscillator if this is the case, otherwise the program will get stuck in an endless loop waiting for the crystal to start.

Code assertions

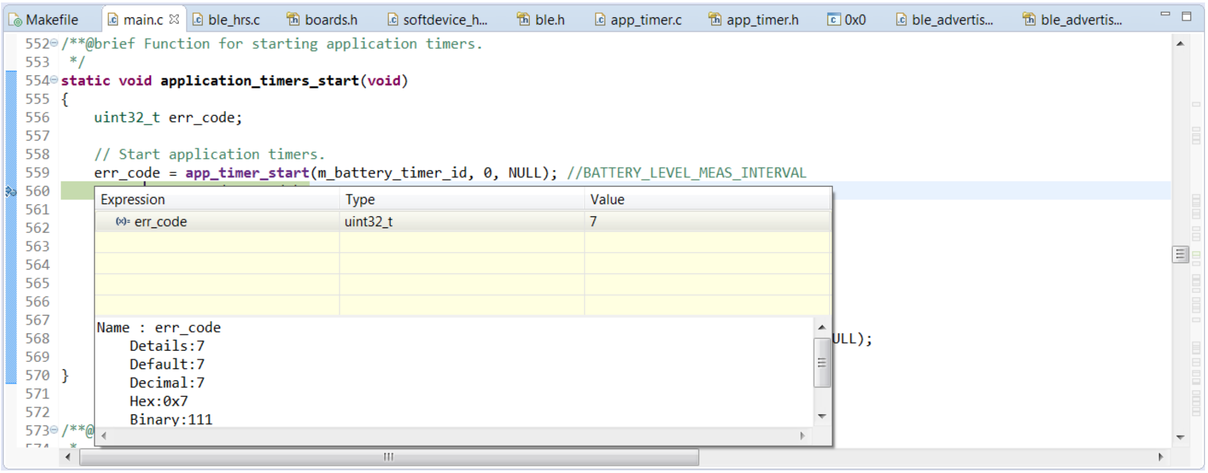
The SDK examples uses code assertions to catch error conditions at runtime (i.e., unexpected return values from function calls), please refer to the SDK documentation for more details (link). Assertions will by default lead to a system reset unless –DDEBUG is added to the list of preprocessor symbols (CFLAGS variable in Makefile).

Example:

Assert condition when starting an application timer instance.

```
    err_code = app_timer_start(m_timer_id,
                               INTERVAL, // INTERVAL = 0, illegal value
                               NULL);
    APP_ERROR_CHECK(err_code); // Error handler invoked if err_code != NRF_SUCCESS
```

The interrupt interval is set to zero in the function call above, which is an illegal value. Thus, the function will return NRF_ERROR_INVALID_PARAM:



```
/**@brief Function for starting application timers.
 */
static void application_timers_start(void)
{
    uint32_t err_code;

    // Start application timers.
    err_code = app_timer_start(m_battery_timer_id, 0, NULL); //BATTERY_LEVEL_MEAS_INTERVAL
```

```
ULL);
```

```
}


/**@
```

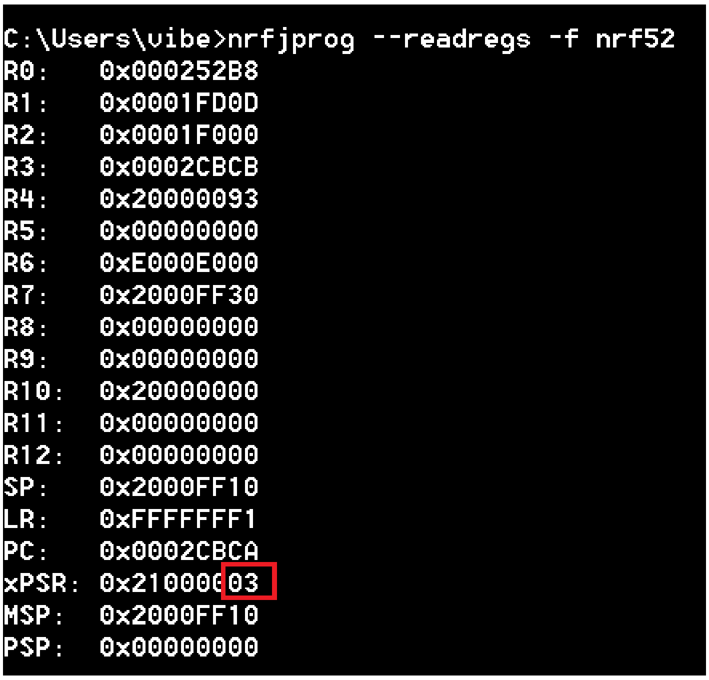
Expression	Type	Value
err_code	uint32_t	7

```
Name : err_code
    Details:7
    Default:7
    Decimal:7
    Hex:0x7
    Binary:111
```

Figure 15: NRF_ERROR_INVALID_PARAM error. Error codes are listed in nrf_error.h header.

Hardfault exception

The default handler for the hardfault exception is an infinite loop, so the program will be stuck if the program triggers this exception. One quick way to determine if this has happened is to read out the interrupt status register and check if the ISR number is set ‘3’.



```
C:\Users\vibe>nrfjprog --readregs -f nrf52
R0:    0x000252B8
R1:    0x0001FD0D
R2:    0x0001F000
R3:    0x0002CBCB
R4:    0x20000093
R5:    0x00000000
R6:    0xE000E000
R7:    0x2000FF30
R8:    0x00000000
R9:    0x00000000
R10:   0x20000000
R11:   0x00000000
R12:   0x00000000
SP:    0x2000FF10
LR:    0xFFFFFFF1
PC:    0x0002CBCA
xPSR:  0x21000003
MSP:   0x2000FF10
PSP:   0x00000000
```

Figure 16: Check interrupt status register

Hardfault exceptions are often caused by incorrect memory layout, but can also be caused by errors in the application code such as illegal memory access. Please post a question on the forum if help is needed to debug the hardfault.