Porting Guide – STM32CubeIDE

STM32 Cortex M Series

Version 5.1.1

Published November 1, 2023

Table of Contents

Chapter 1

PX5 RTOS porting guide for STM32 Evaluation Kits - Overview

PX5 RTOS samples are available for several STM32 evaluation kits, but if you need to port it to a different one for which there is no sample, this document describes the process to achieve that using STM32CubeIDE and resources which can be obtained from the PX5 RTOS website.

While this document shows the steps for the porting process, if you need more details you can refer to the PX5 RTOS Binding User Guide.

Note that this porting guide refers to pre-built object code versions of px5.c and px5_binding.s (px5.o and px5_binding.o). Furthermore, the evaluation is limited to a maximum of 10 threads. Once the limit is reached, an EINVAL error code is returned from the pthread_create API. If a full source code evaluation is required, please contact PX5 at sales@px5rtos.com .

Chapter 2

Porting Steps

This chapter describes the process of porting PX5 RTOS to an STM32 Cortex M microcontroller using the STM32CubeIDE development tool.

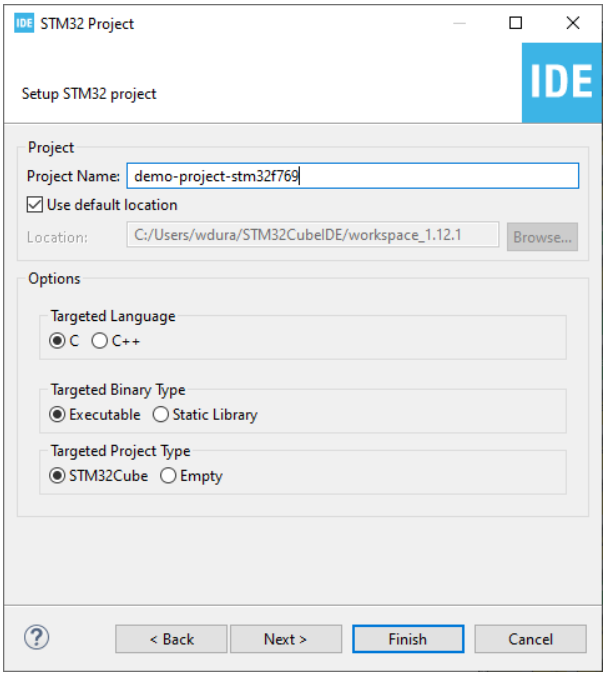
Step 1: Generating the basic project using STM32CubeIDE

This process was created using STM32CubeIDE version 1.13.2.

Create a new project: *File/New/STM32 project*, on *board selector* pick the Cortex M evaluation board you have then click *next*.

For this guide, we'll be using the STM32F769i-DISCO, but the process is similar to other Cortex M MCUs.

Name your project and click *Finish*.



In order to generate a minimal version of the project, we'll be excluding most of peripherals. To do so, under the *Pinout and Configuration* tab:

System Core: Disable IWDG and WWDG

Analog: Disable all ADCs

Timers: Disable all

Connectivity: Disable all

Multimedia: Disable all

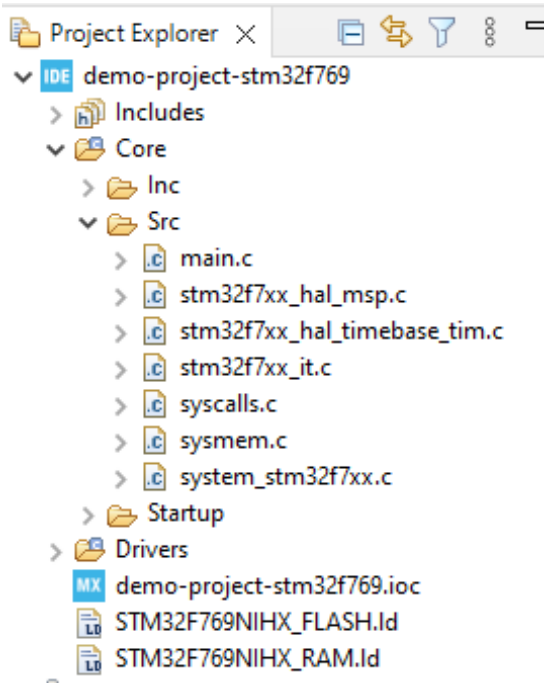
Security: Disable RNG

Computing: Disable CRC

Middleware and SW Packs: Disable FreeRTOS

Save your configuration: *CTRL+S.* STM32CubeIDE will prompt if you want to generate code, hit *Yes*.

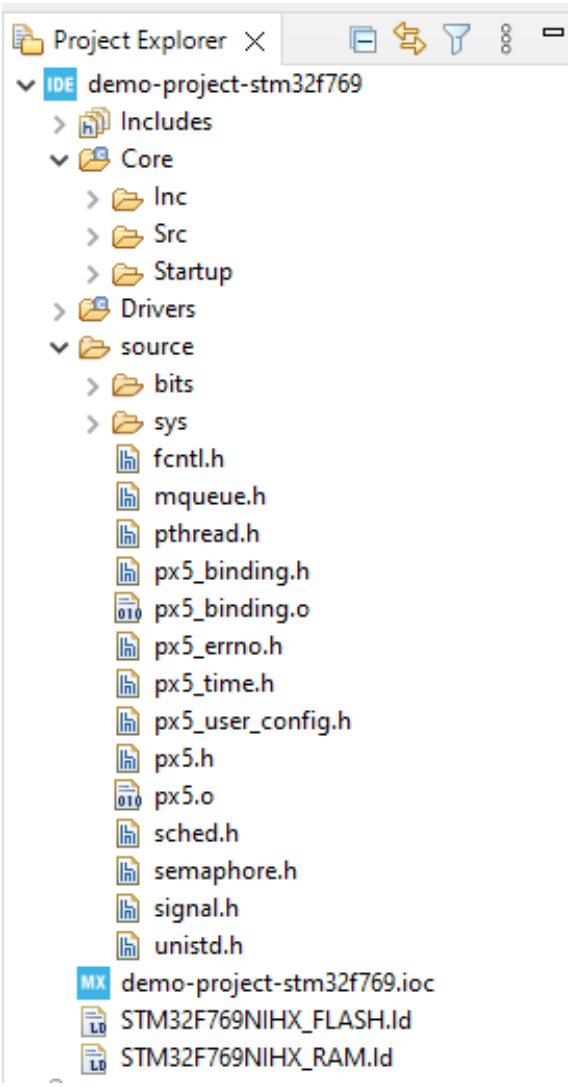
On project explorer you should find the basic project structure as follows:



Step 2: Add PX5 RTOS source code

For this step you'll need to have access to PX5 RTOS files, which you can obtain from any sample on the PX5 RTOS website with a compatible architecture (like Cortex M7 sample if you're using a Cortex M7 MCU).

Drag and drop the PX5 RTOS source folder under your project's root directory to copy all files and directories. Your project should now look like the following:



Step 3: Modify the linker file

Two changes are needed in order to provide memory for PX5 RTOS process. Open the linker file under your project root directory, in this case the **STM32F769NIHX_FLASH.ld** file.

Add the following line under the existing min_stack_size definition, which is normally on line 42.

```
_Min_Proc_Stack_Size = 0x400; /* required amount of PX5
RTOS process stack */
```

This part of the linker file should look like the following after the modification:

```
38 /* Highest address of the user mode stack */
39 _estack = ORIGIN(RAM) + LENGTH(RAM); /* end of "RAM" Ram type memory */
40
41 _Min_Heap_Size = 0x200; /* required amount of heap */
42 _Min_Stack_Size = 0x400; /* required amount of stack */
43 _Min_Proc_Stack_Size = 0x400;  /* required amount of PX5 RTOS process stack */
```

Still on the linker file, replace the user_heap_stack content with the below:

```
._user_heap_stack :
  {
  . = ALIGN(8);
  PROVIDE ( end = . );
  PROVIDE ( _end = . );
  . = . + _Min_Heap_Size;
  _Proc_Stack_Base = .;
  . = . + _Min_Proc_Stack_Size;
  _Proc_Stack_Limit = .;
  . = . + _Min_Stack_Size;
  . = ALIGN(8);
} >RAM
```

This part of the linker file should look like the following after the modification:

```
166   /* User_heap_stack section, used to check that there is enough "RAM" Ram  type memory left */
167   ._user_heap_stack :
168     {
169     . = ALIGN(8);
170     PROVIDE ( end = . );
171     PROVIDE ( _end = . );
172     . = . + _Min_Heap_Size;
173     _Proc_Stack_Base = .;
174     . = . + _Min_Proc_Stack_Size;
175     _Proc_Stack_Limit = .;
176     . = . + _Min_Stack_Size;
177     . = ALIGN(8);
178   } >RAM
```

Step 4: Modify the startup file

The start-up file requires a modification so the application will use the process stack. Open the startup file under core/startup/**startup_stm32F769nihx.s** file.

Find the Reset_Handler and add the following after the set stack pointer instruction.

```
  /* PX5 RTOS, switch to use PSP and set the stack top
to it. */

  ldr r0, =_Proc_Stack_Limit

  msr psp, r0

  mov r1, #2

  msr control, r1

  mov sp, r0
```

This part of the startup file should look like the following after the modification:

```
57    .section  .text.Reset_Handler
58  .weak  Reset_Handler
59  .type  Reset_Handler, %function
60Reset_Handler:
61  ldr   sp, =_estack      /* set stack pointer */
62
63  /* PX5 RTOS, switch to use PSP and set the stack top to it. */
64  ldr r0, =_Proc_Stack_Limit
65  msr psp, r0
66  mov r1, #2
67  msr control, r1
68  mov sp, r0
```

Step 5: Modify the ISR file

Some changes are needed in order to provide a single, periodic timer interrupt to drive all of PX5 RTOS time related services. Open the ISR file under core/src/**stm32f7xx_it.c** file.

Add the following function declaration before the *SysTick_Handler* function:

```
void px5_timer_interrupt_process(void);
```

Add the following function call within the *SysTick_Handler* function:

```
px5_timer_interrupt_process();
```

This part of the ISR file should look like the following after the modification:

```
180
181 void px5_timer_interrupt_process(void);
182
183 /**
184   * @brief This function handles System tick timer.
185   */
186 void SysTick_Handler(void)
187 {
188   /* USER CODE BEGIN SysTick_IRQn 0 */
189     px5_timer_interrupt_process();
190   /* USER CODE END SysTick_IRQn 0 */
191
192   /* USER CODE BEGIN SysTick    q_IRQn 1 */
193
194   /* USER CODE END SysTick_IRQn 1 */
195 }
```

Still on this file, comment out or remove these two functions to avoid duplicate declaration:

- `void PendSV_handler(void)`
- `void SVC_Handler(void)`

Step 6: Remove the syscalls.c file

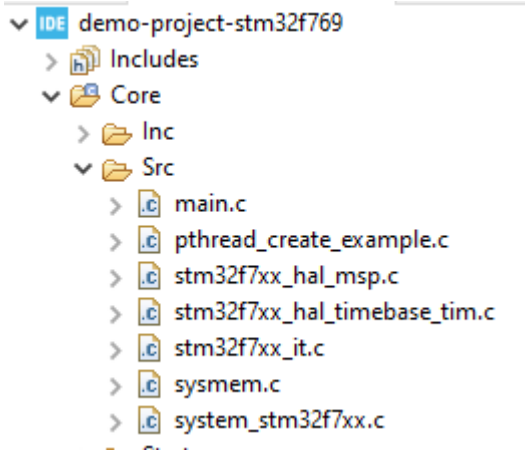
Find the **syscalls.c** file under core/src and remove it from the project. Right click on the file and select *delete*. Confirm the action.

Step 7: Add a PX5 RTOS example file

For this step you'll need to have access to PX5 RTOS files, which you can obtain from any sample on the PX5 RTOS website with a compatible architecture (like Cortex M7 sample if you're using a Cortex M7 MCU).

Using the PX5 RTOS example folder, drag and drop the **pthread_create_example.c** under core/src and confirm the copy of this file.

Your project directory should now look like the following:



Step 8: Modify main.c file

Since the example file already contains a main function, we need to rename the existing one. Open the file under core/src/**main.c**

Rename the main function call from *int main(void)* to *void platform_setup(void)*

Then, remove the while loop from the same function.

Step 9: Modify HAL time base file

Regarding the timer interrupt period, most applications use a 1ms interrupt interval. However, the application can use any timer interrupt frequency, providing that the PX5 RTOS is built with the correct value of PX5_TICKS_PER_SECOND (default is 1,000 which represents a 1ms timer interrupt interval).

Open the HAL time base file under core/src/**stm32f7xx_hal_timebase_tim.c** and add the following code under the *HAL_InitTick* function.

```
  /* Configure the SysTick to have interrupt in 1ms
time basis */

  HAL_SYSTICK_Config(SystemCoreClock/1000);



  /* Configure the SysTick IRQ priority */

  HAL_NVIC_SetPriority(SysTick_IRQn, TickPriority, 0);
```

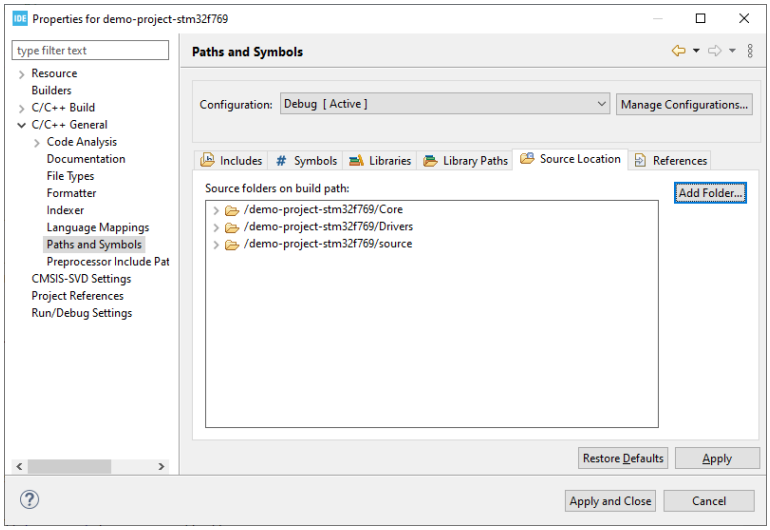
This part of the HAL time base file should look like this after the modification:

```
41  HAL_StatusTypeDef HAL_InitTick(uint32_t TickPriority)
42  {
43    RCC_ClkInitTypeDef    clkconfig;
44    uint32_t              uwTimclock, uwAPB1Prescaler = 0U;
45
46    uint32_t              uwPrescalerValue = 0U;
47    uint32_t              pFLatency;
48    HAL_StatusTypeDef     status;
49
50    /* Configure the SysTick to have interrupt in 1ms time basis */
51    HAL_SYSTICK_Config(SystemCoreClock/1000);
52
53    /* Configure the SysTick IRQ priority */
54    HAL_NVIC_SetPriority(SysTick_IRQn, TickPriority, 0);
55
56    /* Enable TIM6 clock */
57    __HAL_RCC_TIM6_CLK_ENABLE();
```

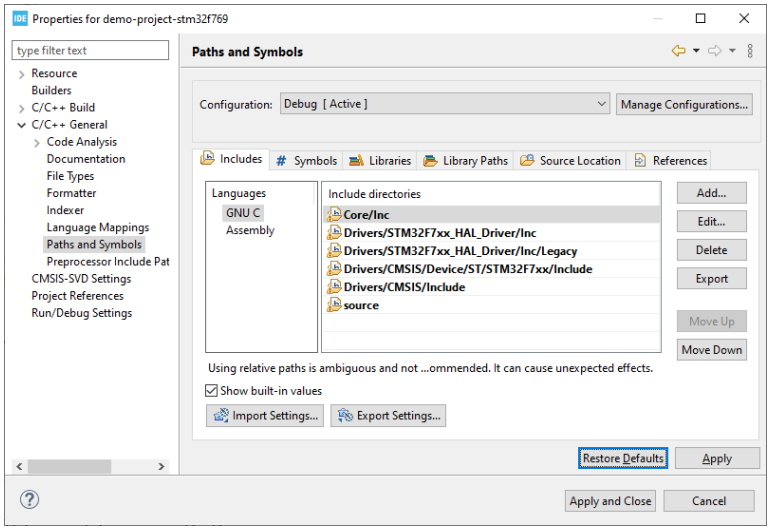
Step 10: Update Project's Paths and Symbols

The last step is to update the project's paths and symbols to add the PX5 RTOS source folder. To do so, right click project name, select *properties*.

Then, select *Paths and Symbols* under *C/C++ General*. Select the *Source Location* tab, click on the button *Add Folder*, select *source* and hit *ok*.



Still on this window, select the *Includes* tab, click on the *add* button, type *source* and hit *ok*, then *apply*.

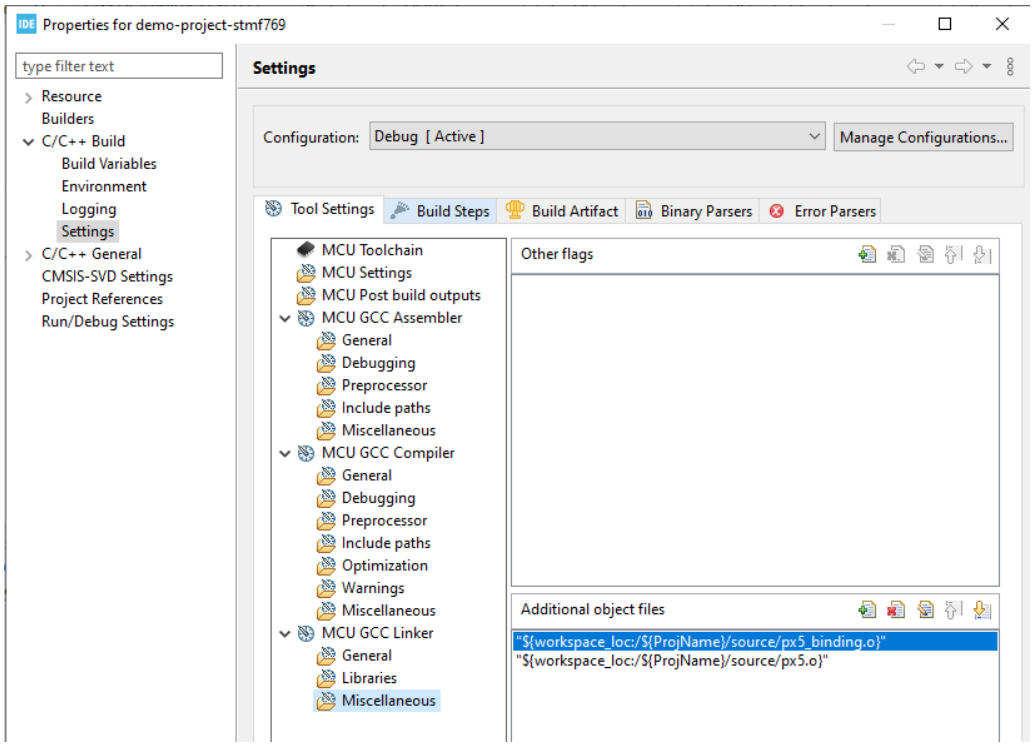


In this last step, we need to add the two object files from PX5 RTOS to the linker options: open the *Settings* tab under *C/C++ Build*.

Then, select the *Tools Settings* tab, expand *MCU GCC Linker*, and click on *Miscellaneous*.

The bottom screen displays “additional object files”, click in the add path file icon (the one with a green plus sign), select *Workspace,* find the source directory and then select both files:

- px5.o
- px5_binding.o



Click *Apply* then *Apply and Close*. Your project is ready to be built and tested on your target device.

Enhance • Simplify • Unite

11440 West Bernardo Court • Suite 300
San Diego, CA 92127, USA

Phone: +1 (858) 753-1715
Website: px5rtos.com