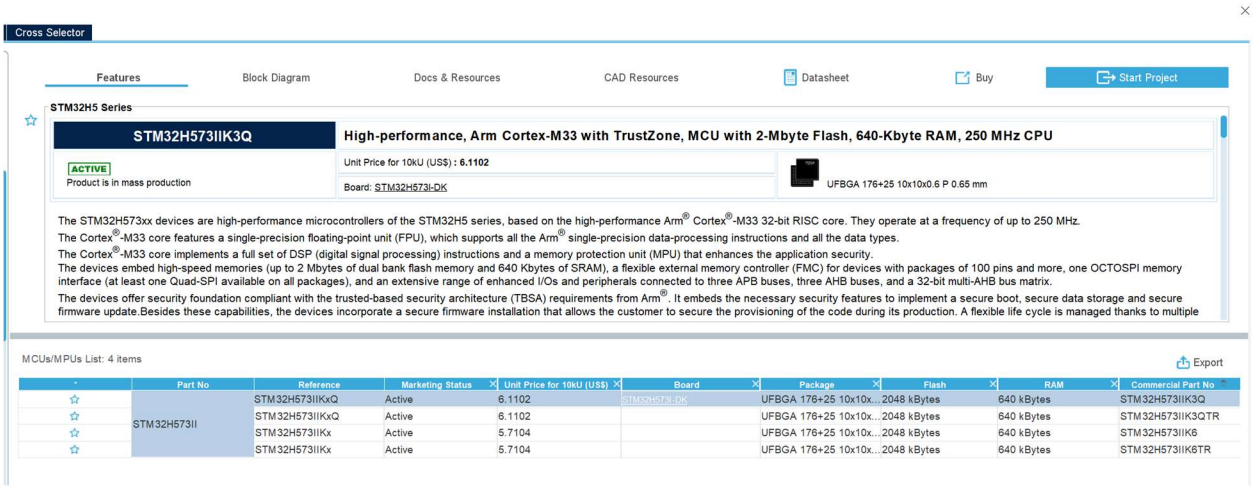
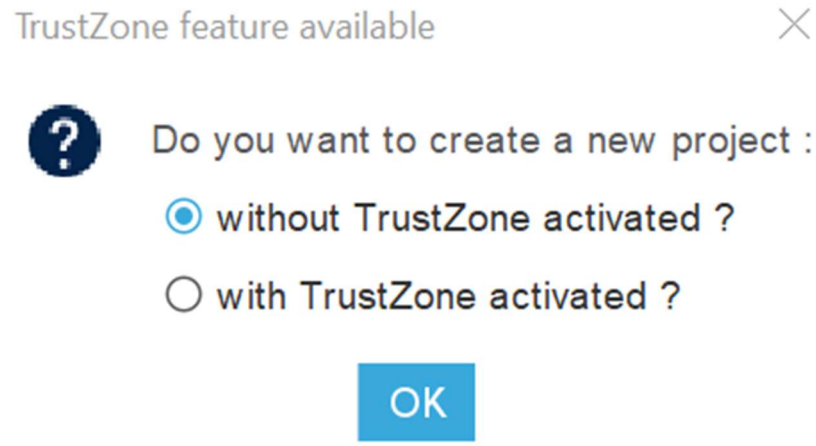
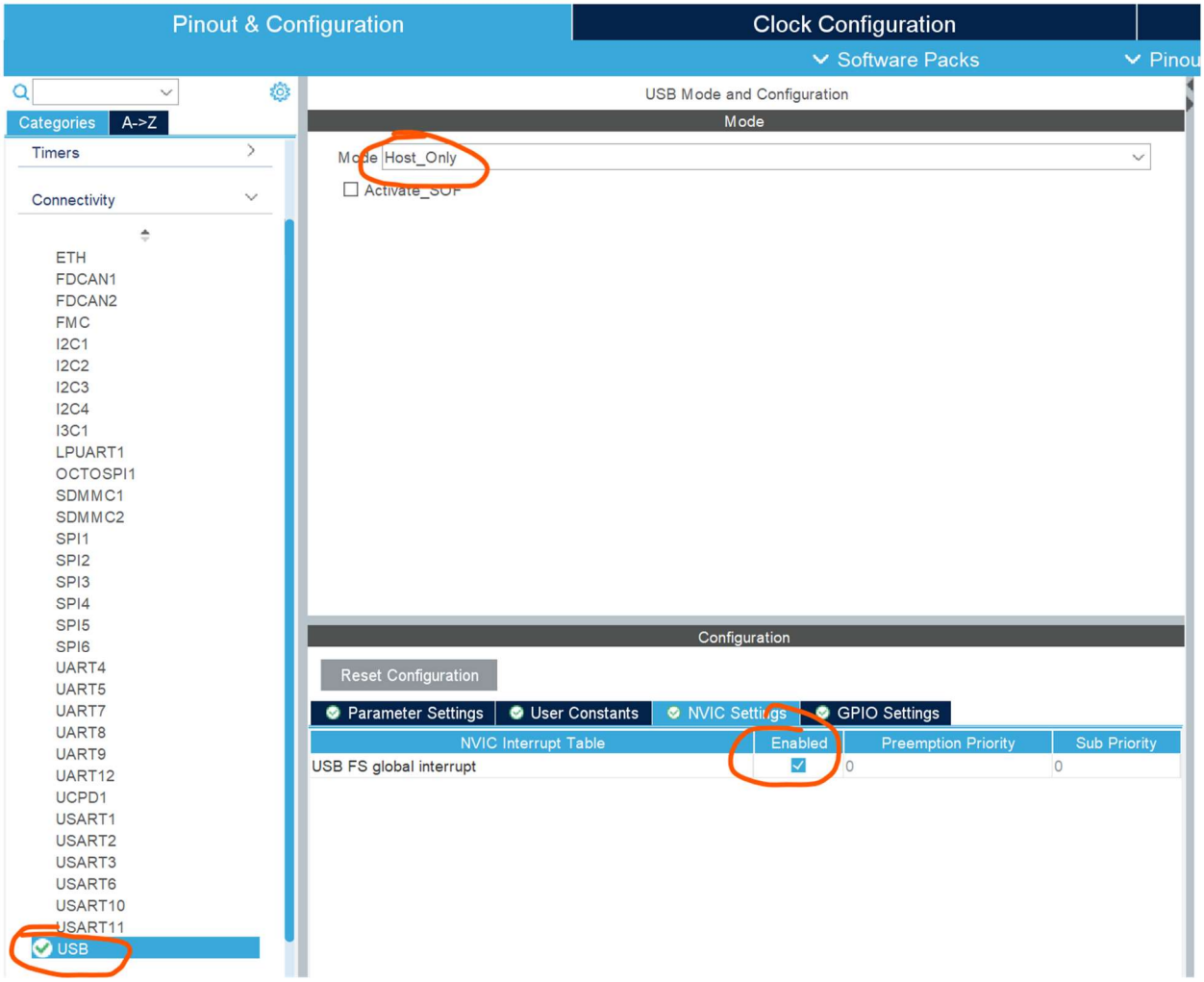
Project Creation Instructions

Blank project

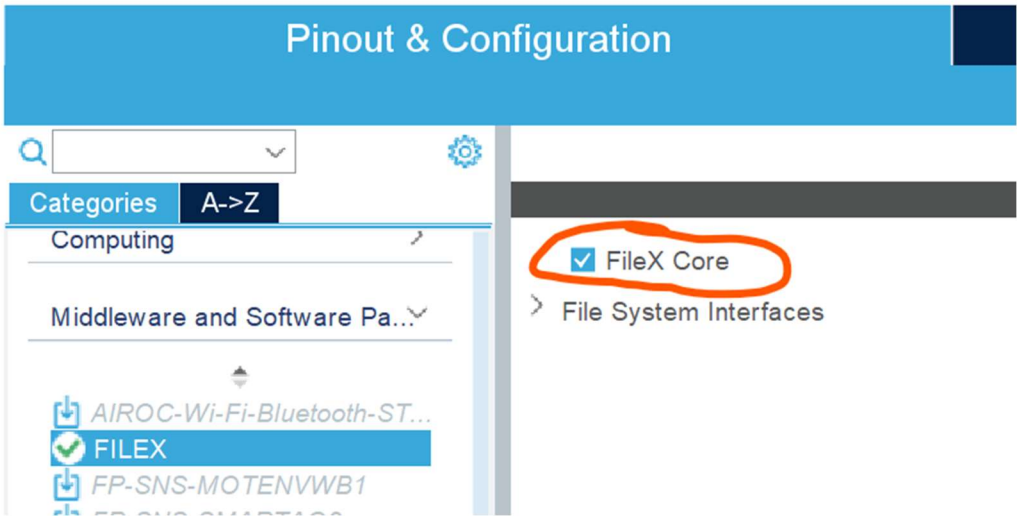




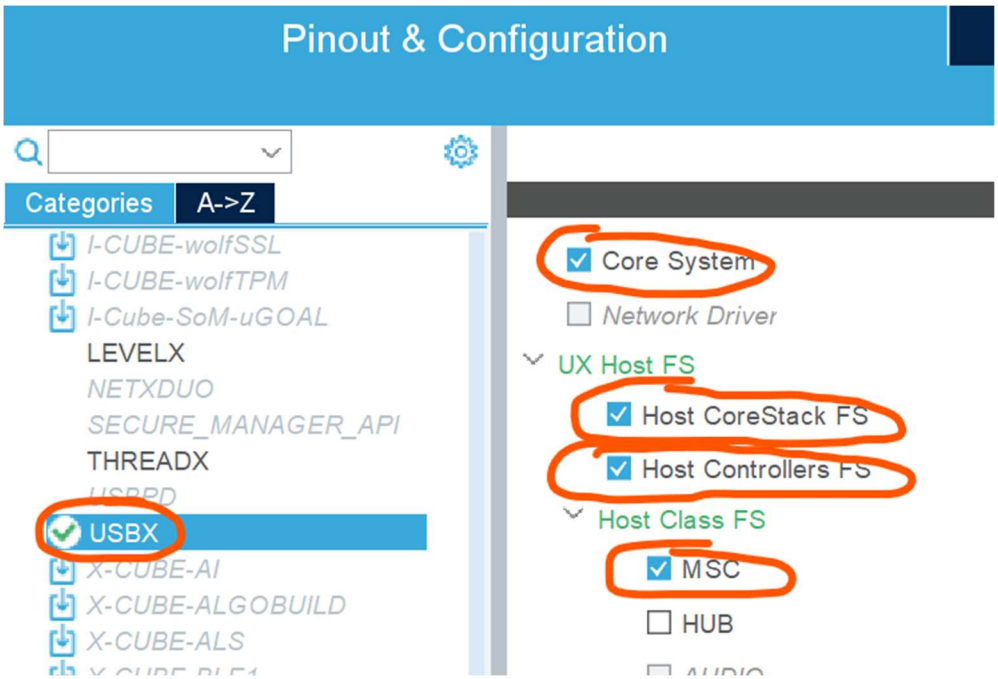
Enable USB, Host only, with global interrupt enabled



Enable FileX core

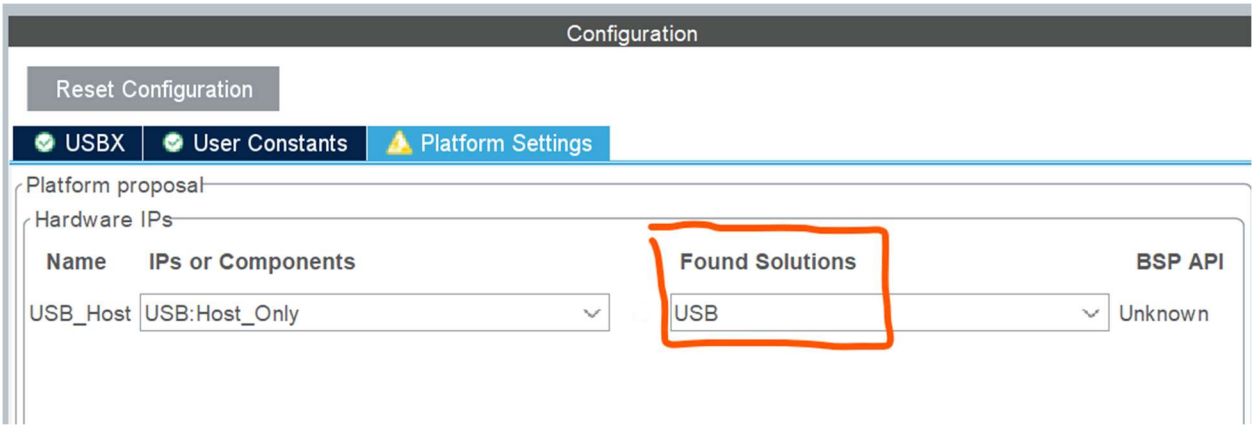


USBX



Set USB driver:

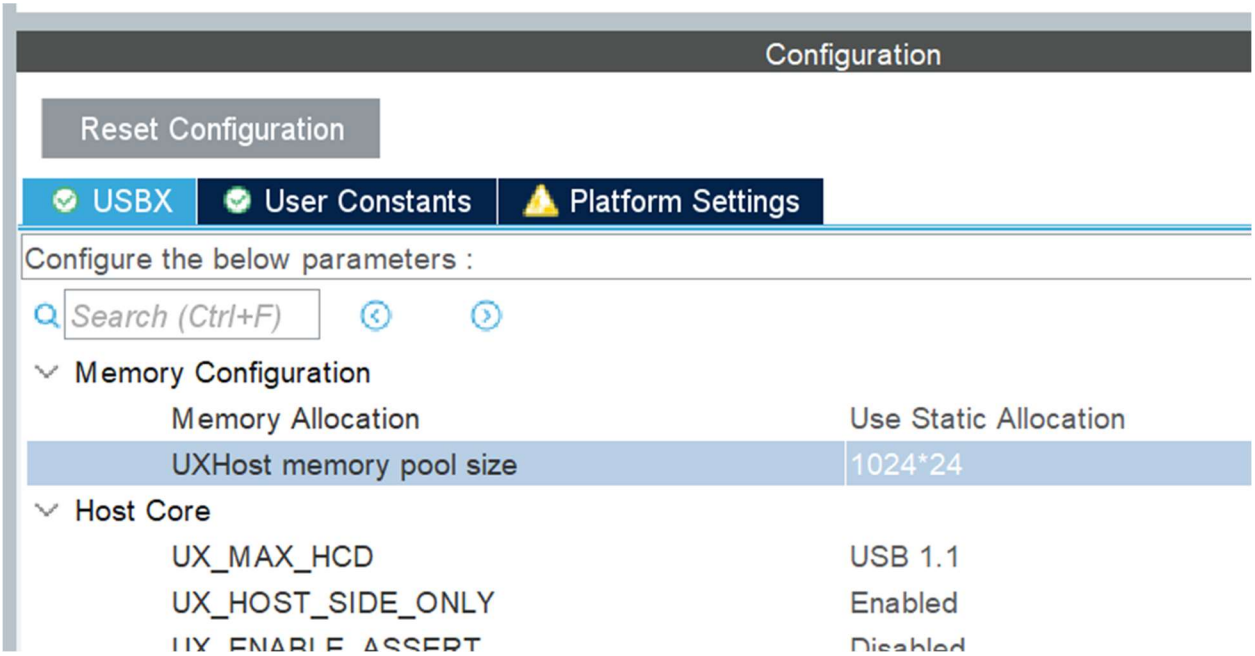
(leave VBUS unset)



Increase memory pool

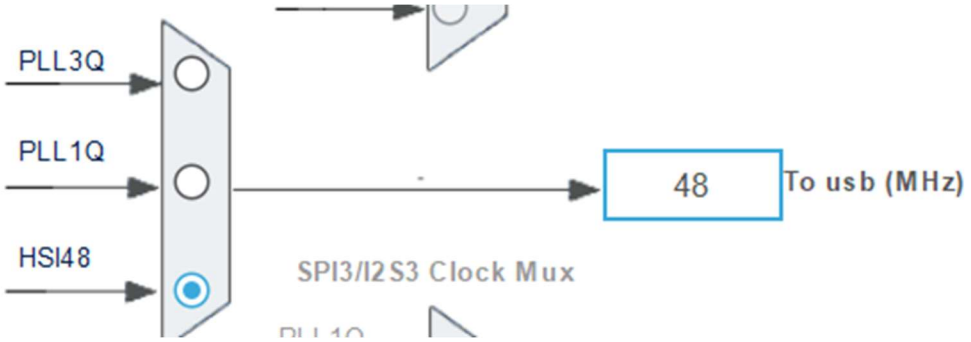
(24KB is what most of the ST examples use)

(note; this only does one part, second code mod done below)

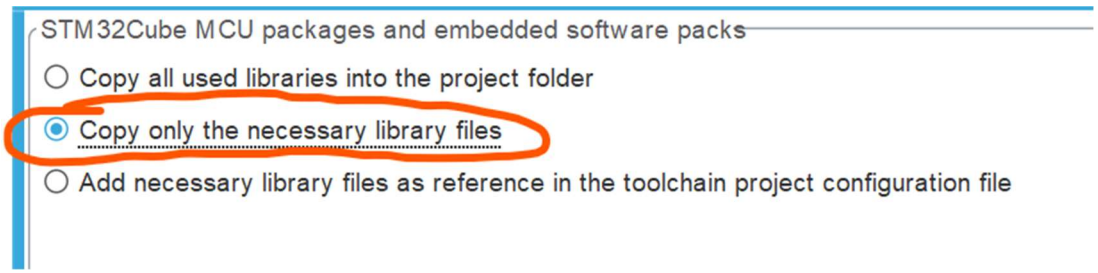


Confirm USB clock is 48Mhz

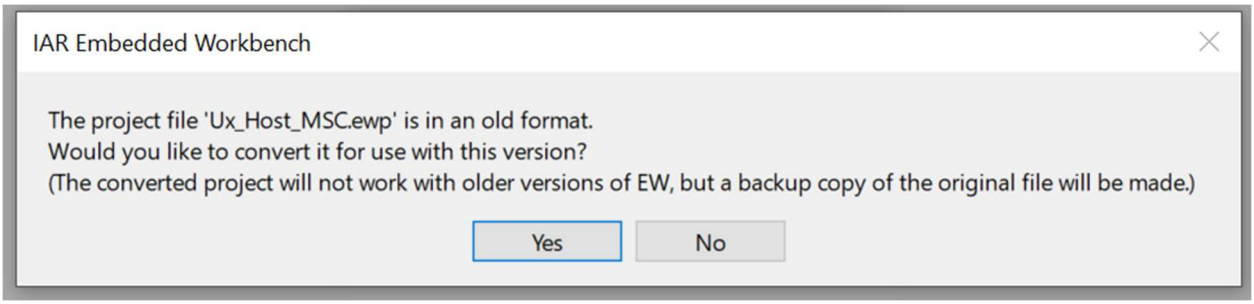
Other clocks can be “whatever” – adjust if needed



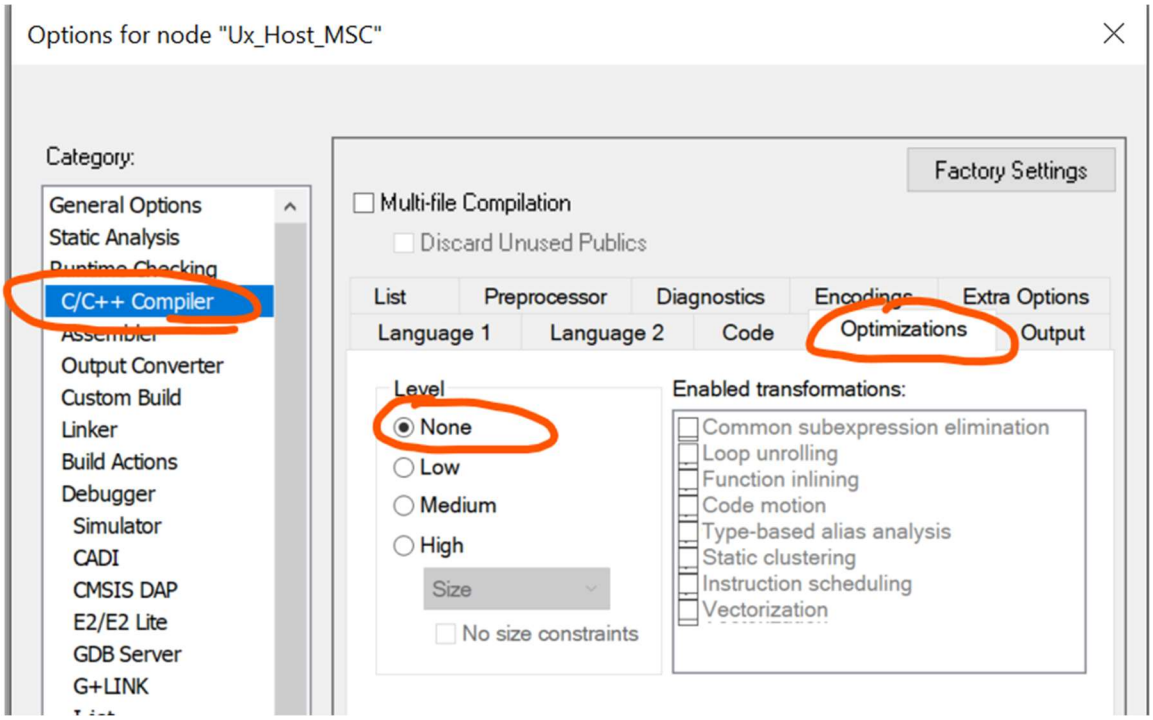
Generate



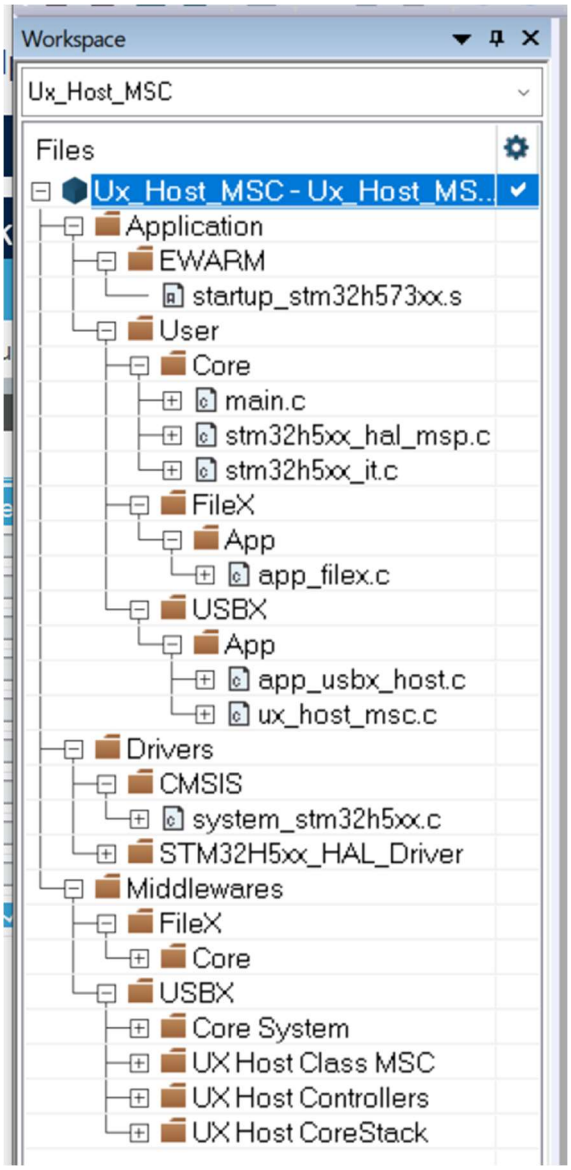
Project Import, IAR 9.50.1:



Improve debuggability while developing (recommend going back to optimizations after debug)

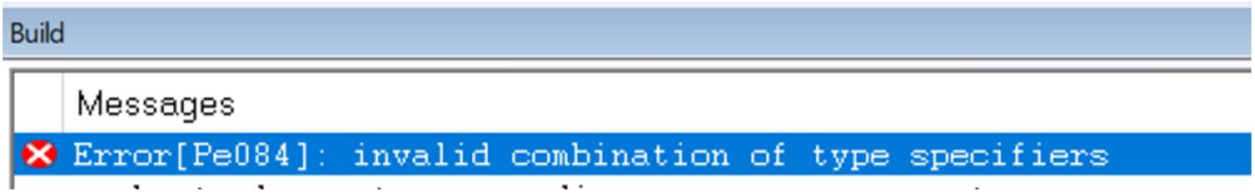


Starting tree:



Fix compilation error in autogenerated code…

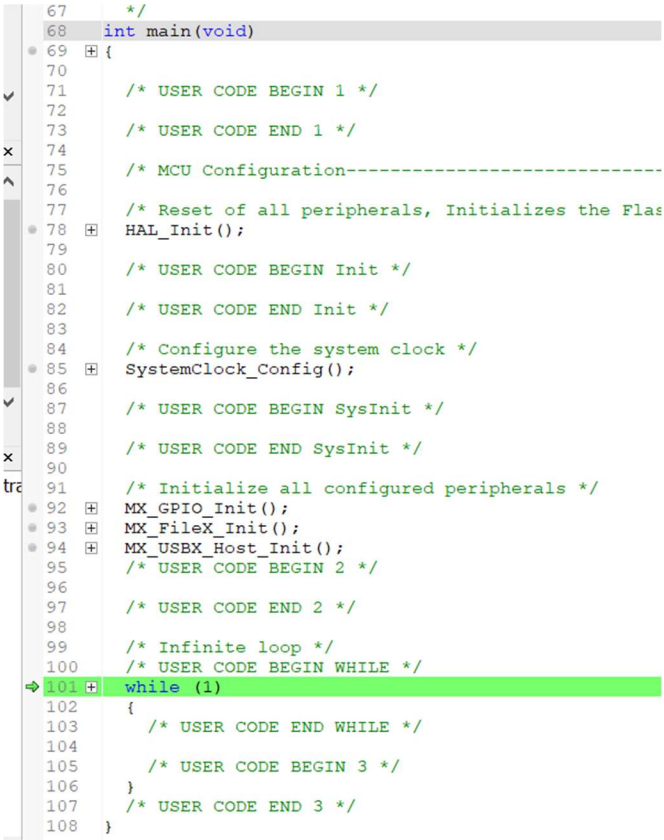
This error:



Is addressed by making this change: in ux_port.h

Checkpoint – Compile without Errors

Build and Run – reach infinite while loop in main.c



Normally there is a '_ux_host_stack_enum_thread_entry' thread that runs to ensure that two parts of the software (multiple threads) can't perform concurrent enumeration and cause data races.

But we are using standalone mode, so we will call the hook explicitly. This is valid only in non-threaded mode. _ux_host_stack_tasks_run().

Edits to main:

```
/* USER CODE END Header */
/* Includes ------------------------------------------
#include "main.h"
#include "app_filex.h"
#include "app_usbx_host.h"

/* Private includes ----------------------------------
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */
```

```
/* USER CODE END Header */
/* Includes ------------------------------------
#include "main.h"
#include "app_filex.h"
#include "app_usbx_host.h"

/* Private includes ----------------------------
/* USER CODE BEGIN Includes */
#include "ux_hcd_stm32.h"
#include "ux_host_class_storage.h"
#include "ux_host_stack.h"
/* USER CODE END Includes */
```

```
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_FileX_Init();
  MX_USBX_Host_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
```

```
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_FileX_Init();
  MX_USBX_Host_Init();
  /* USER CODE BEGIN 2 */

  // in a threaded ThreadX app this would be in 'USBX_APP_Host_Init' and immediately follow MX_USBX
  {
      MX_USB_HCD_Init(); // low-level driver
      ux_host_stack_hcd_register(_ux_system_host_hcd_stm32_name,
          _ux_hcd_stm32_initialize, (ULONG)USB_DRD_FS,
          (ULONG)&hhcd_USB_DRD_FS);
  }
  // end ThreadX app 'USBX_APP_Host_Init'


  // in a threaded ThreadX app this would be in 'app_ux_host_thread_entry'
  {
      // start USB host (to implement true OTG you wouldn't want to do this by default)
      // but if you are only a host (or are OTG only effectively supporting host, then this is ok)
      // for instance, in host mode you'd call HAL_HCD_Start; and in device mode call HAL_HCD_Stop
      HAL_HCD_Start(&hhcd_USB_DRD_FS);
  }
  // end ThreadX app 'app_ux_host_thread_entry'

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */


    // in a threaded ThreadX app this would be in 'ux_host_stack_enum_thread_entry'
    {
        _ux_host_stack_tasks_run();
    }
    // end ThreadX app 'ux_host_stack_enum_thread_entry'

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
```

Build and run, hit a hard fault; adjust buffers to avoid stack hard faults. These numbers came from ST example app. Tune later.

```
/* Exported constants --------------------------------------------------------*/
#define UX_HOST_APP_MEM_POOL_SIZE  1024*24
#define USBX_HOST_MEMORY_STACK_SIZE     1024

/* USER CODE BEGIN EC */

/* USER CODE END EC */
```

```
/* Exported constants --------------------------------------------------------*/
#define UX_HOST_APP_MEM_POOL_SIZE  1024*28
#define USBX_HOST_MEMORY_STACK_SIZE     1024*24

/* USER CODE BEGIN EC */

/* USER CODE END EC */
```

Checkpoint - Enumeration

Put a breakpoint in ux_host_event_callback and with a MSC device connected, the callback should be executed.

app_usbx_host.c

File Scope | ux_host_event_callback

```
169    * @param  current_instance: Pointer to class instance.
170    * @retval status
171    */
172   UINT ux_host_event_callback(ULONG event, UX_HOST_CLASS *current_class, VOID *current_instance)
173   {
174     UINT status = UX_SUCCESS;
175
176     /* USER CODE BEGIN ux_host_event_callback0 */
177
178
179     /* USER CODE END ux_host_event_callback0 */
180
181     switch (event)
182     {
183       case UX_DEVICE_INSERTION:
184
185         /* USER CODE BEGIN UX_DEVICE_INSERTION */
186
187         /* Get current Storage Class */
188         if (current_class -> ux_host_class_entry_function == ux_host_class_storage_entry)
189         {
190           if (storage == UX_NULL)
191           {
192             /* Get current Storage Instance */
193             storage = (UX_HOST_CLASS_STORAGE *)current_instance;
194
195             printf("\nUSB Mass Storage Device Plugged");
196             printf("PID: %#x ", (UINT)storage -> ux_host_class_storage_device -> ux_device_descriptor.idProduct);
197             printf("VID: %#x ", (UINT)storage -> ux_host_class_storage_device -> ux_device_descriptor.idVendor);
198
199             /* Get the storage media */
200             storage_media = (UX_HOST_CLASS_STORAGE_MEDIA *)current_class -> ux_host_class_media;
```