

Project Creation Instructions

Blank project

Cross Selector

Features | Block Diagram | Docs & Resources | CAD Resources | Datasheet | Buy | Start Project

STM32H5 Series

STM32H573IIK3Q

High-performance, Arm Cortex-M33 with TrustZone, MCU with 2-Mbyte Flash, 640-Kbyte RAM, 250 MHz CPU

ACTIVE
Product is in mass production

Unit Price for 10kU (US\$) : 6.1102

Board: STM32H573I-DK

UFPGA 176+25 10x10x0.6 P 0.65 mm

The STM32H573xx devices are high-performance microcontrollers of the STM32H5 series, based on the high-performance Arm® Cortex®-M33 32-bit RISC core. They operate at a frequency of up to 250 MHz. The Cortex®-M33 core features a single-precision floating-point unit (FPU), which supports all the Arm® single-precision data-processing instructions and all the data types. The Cortex®-M33 core implements a full set of DSP (digital signal processing) instructions and a memory protection unit (MPU) that enhances the application security. The devices embed high-speed memories (up to 2 Mbytes of dual bank flash memory and 640 Kbytes of SRAM), a flexible external memory controller (FMC) for devices with packages of 100 pins and more, one OCTOSPI memory interface (at least one Quad-SPI available on all packages), and an extensive range of enhanced I/Os and peripherals connected to three APB buses, three AHB buses, and a 32-bit multi-AHB bus matrix. The devices offer security foundation compliant with the trusted-based security architecture (TBSA) requirements from Arm®. It embeds the necessary security features to implement a secure boot, secure data storage and secure firmware update. Besides these capabilities, the devices incorporate a secure firmware installation that allows the customer to secure the provisioning of the code during its production. A flexible life cycle is managed thanks to multiple

MCUs/MPUs List: 4 items

	Part No	Reference	Marketing Status	Unit Price for 10kU (US\$)	Board	Packaging	Flash	RAM	Commercial Part No
☆		STM32H573IIKxQ	Active	6.1102	STM32H573I-DK	UFPGA 176+25 10x10x... 2048 kBytes	640 kBytes	640 kBytes	STM32H573IIK3Q
☆	STM32H573II	STM32H573IIKxQ	Active	6.1102		UFPGA 176+25 10x10x... 2048 kBytes	640 kBytes	640 kBytes	STM32H573IIK3QTR
☆		STM32H573IIKx	Active	5.7104		UFPGA 176+25 10x10x... 2048 kBytes	640 kBytes	640 kBytes	STM32H573IIK6
☆		STM32H573IIKx	Active	5.7104		UFPGA 176+25 10x10x... 2048 kBytes	640 kBytes	640 kBytes	STM32H573IIK6TR

Export

TrustZone feature available



Do you want to create a new project :

- without TrustZone activated ?
- with TrustZone activated ?

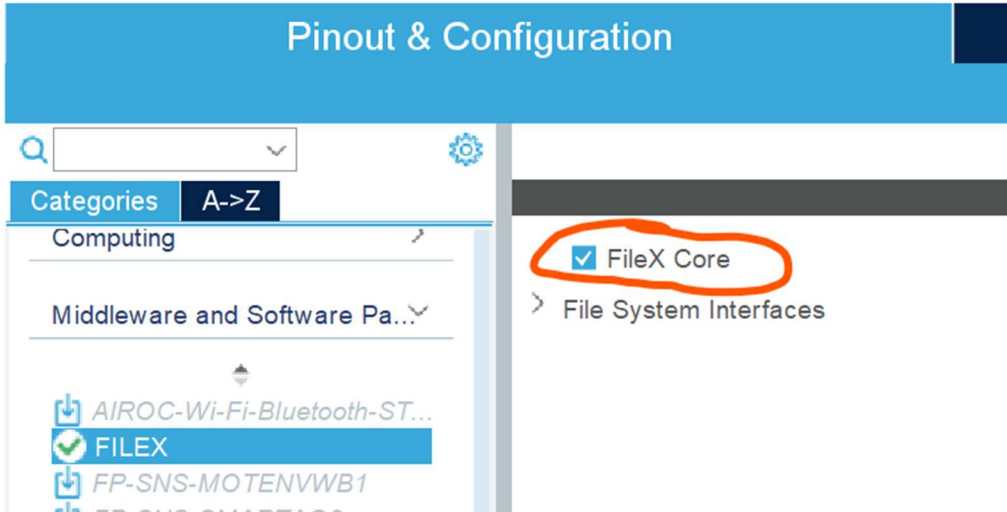


Enable USB, Host only, with global interrupt enabled

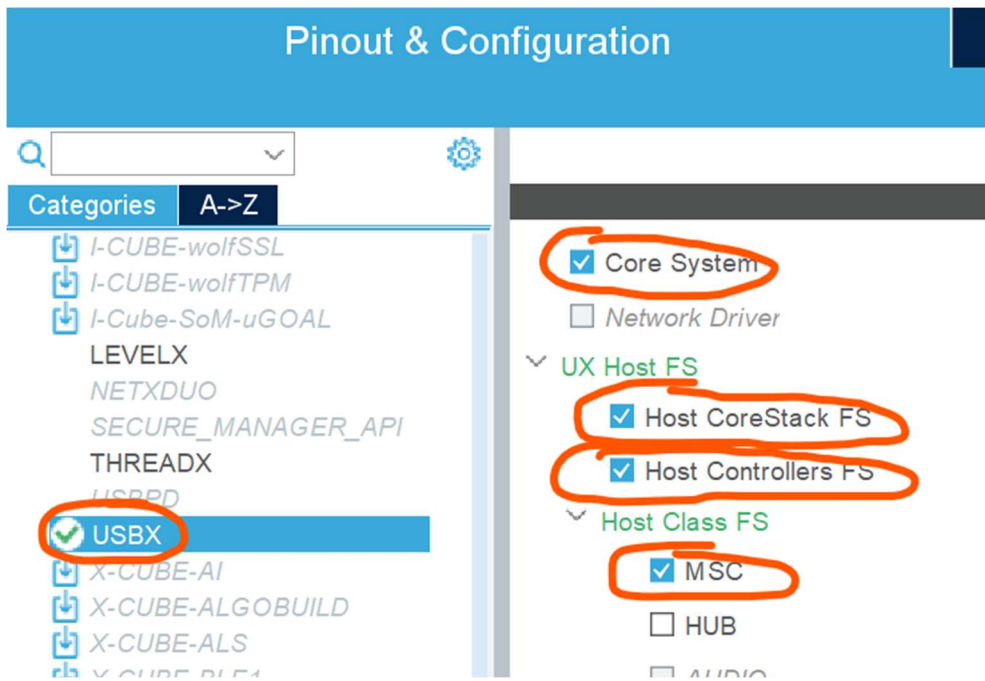
The screenshot displays the STM32CubeIDE configuration interface. The top navigation bar includes 'Pinout & Configuration' and 'Clock Configuration'. The main window is titled 'USB Mode and Configuration'. On the left, a 'Categories' sidebar lists various peripheral groups, with 'USB' selected and circled in orange. The main area shows the 'Mode' dropdown set to 'Host_Only' (circled in orange) and an unchecked 'Activate_SOF' checkbox. Below this is the 'Configuration' section, which includes a 'Reset Configuration' button and tabs for 'Parameter Settings', 'User Constants', 'NVIC Settings', and 'GPIO Settings'. The 'NVIC Settings' tab is active, showing a table with the following data:

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
USB FS global interrupt	<input checked="" type="checkbox"/>	0	0

Enable FileX core



USBX



Set USB driver:

(leave VBUS unset)

The screenshot shows the 'Configuration' window with the 'Platform Settings' tab selected. Under the 'Hardware IPs' section, there is a table with columns: Name, IPs or Components, Found Solutions, and BSP API. The 'USB_Host' row shows 'USB:Host_Only' in the 'IPs or Components' column, 'USB' in the 'Found Solutions' column (highlighted with an orange box), and 'Unknown' in the 'BSP API' column.

Name	IPs or Components	Found Solutions	BSP API
USB_Host	USB:Host_Only	USB	Unknown

Increase memory pool

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

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

The screenshot shows the 'Configuration' window with the 'Platform Settings' tab selected. The 'Memory Configuration' section is expanded, showing a table with columns: Memory Allocation and Use Static Allocation. The 'UXHost memory pool size' row shows '1024*24' in the 'Use Static Allocation' column.

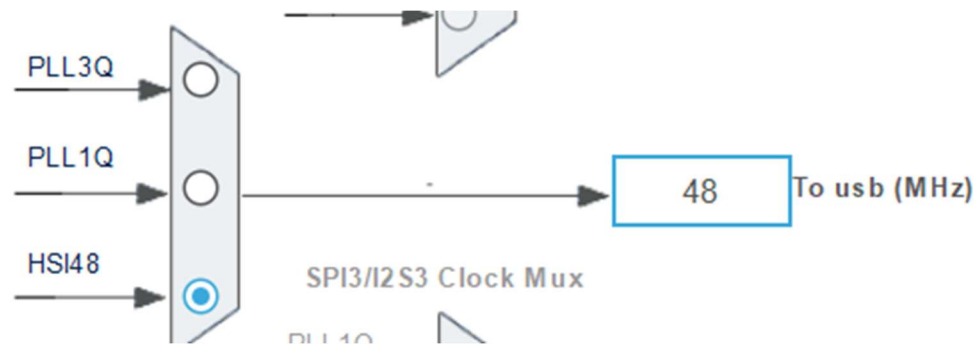
Memory Allocation	Use Static Allocation
UXHost memory pool size	1024*24

Below the memory configuration, the 'Host Core' section is also expanded, showing the following settings:

UX_MAX_HCD	USB 1.1
UX_HOST_SIDE_ONLY	Enabled
UX_ENABLE_ASSERT	Disabled

Confirm USB clock is 48Mhz

Other clocks can be “whatever” – adjust if needed

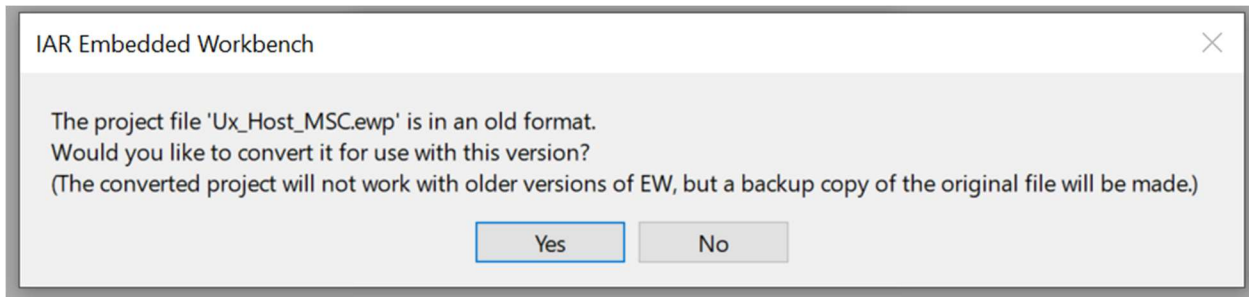


Generate

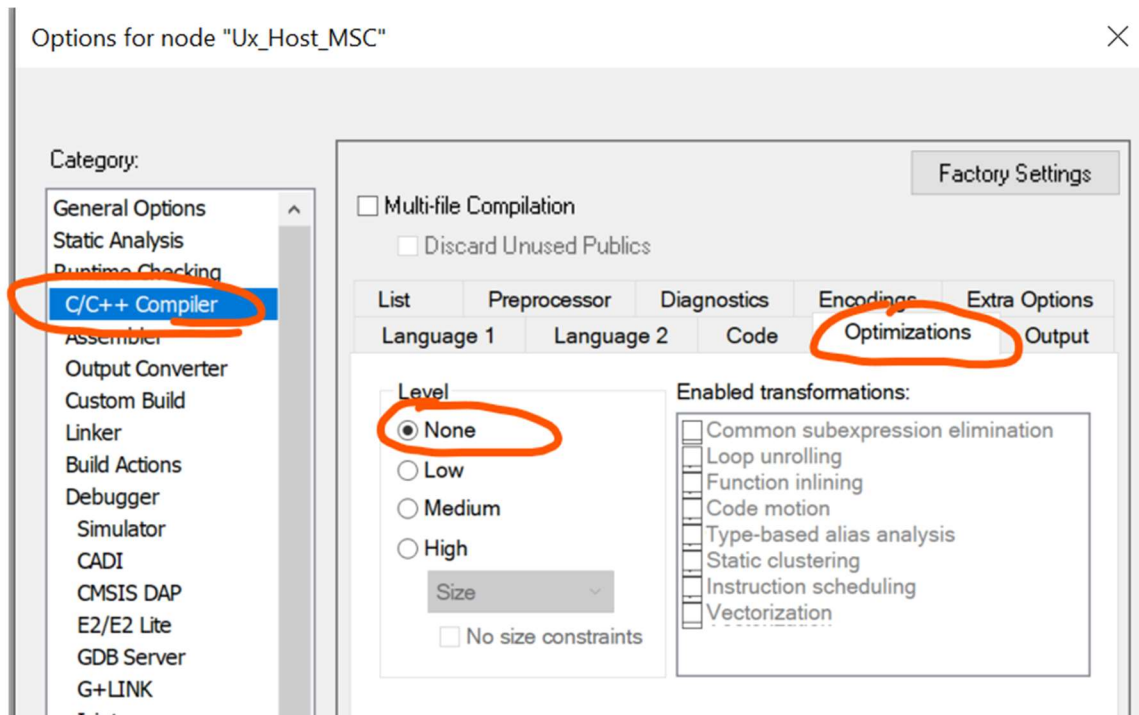
STM32Cube MCU packages and embedded software packs

- Copy all used libraries into the project folder
- Copy only the necessary library files
- Add necessary library files as reference in the toolchain project configuration file

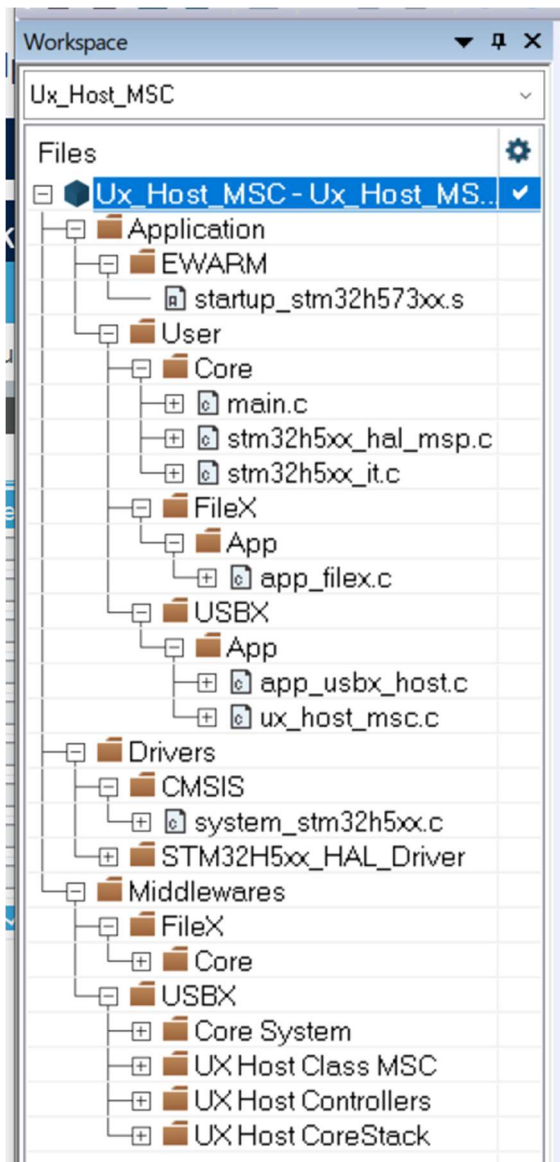
Project Import, IAR 9.50.1:



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



Starting tree:



Fix compilation error in autogenerated code...

This error:

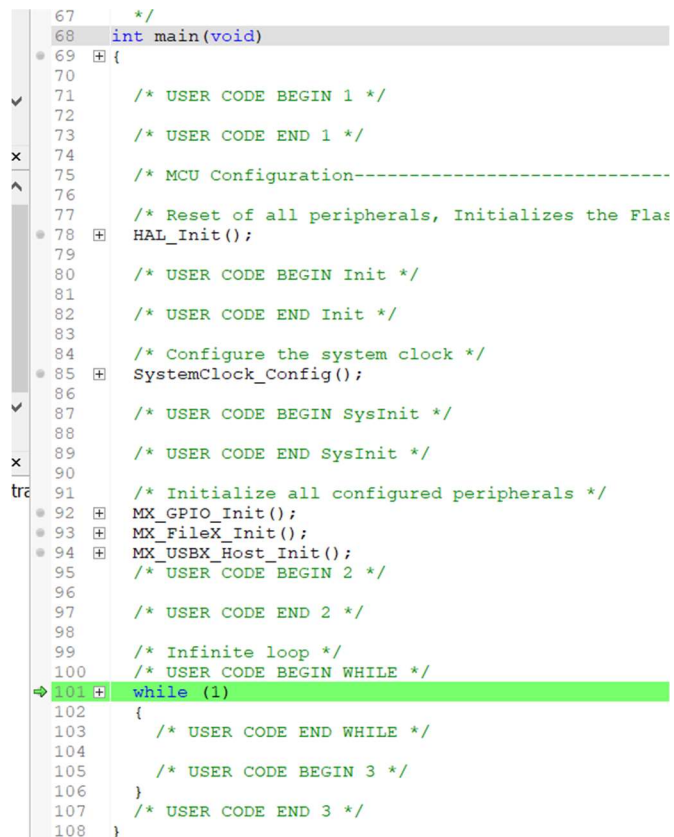


Is addressed by making this change: in ux_port.h



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:

<pre>/* 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 */</pre>	<pre>/* 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 */</pre>
--	--

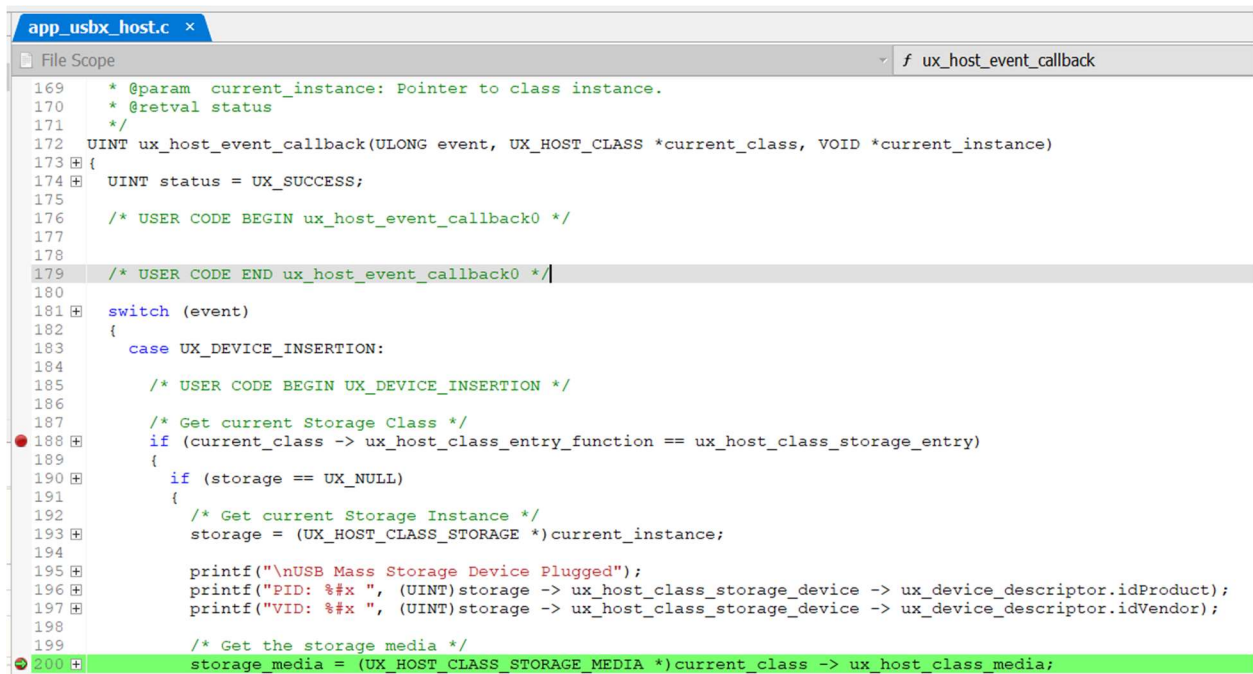
<pre>/* USER CODE END SysInit */ /* Initialize all configured peripherals */ MX_GPIO_Init(); MX_FileX_Init(); MX_USBX_Host_Init(); /* USER CODE BEGIN 2 */ /* USER CODE END 2 */ /* Infinite loop */ /* USER CODE BEGIN WHILE */ while (1) { /* USER CODE END WHILE */ /* USER CODE BEGIN 3 */ } /* USER CODE END 3 */ } /**</pre>	<pre>/* 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)hxcd_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(hxcd_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 */ } /**</pre>
--	---

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

<pre>/* 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 */</pre>	<pre>/* 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 */</pre>
---	--

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