Processor Expert Kinetis SDK USB Stack Integration User Guide

Rev 11/2014

freescale™

Contents

Section number **Title** **Page**

Chapter 1
Introduction

Chapter 2
Processor Expert SDK USB layers description

Chapter 3
Creating PEx USB project

Chapter 1
Introduction

It is a complex task to writing an USB application, but Processor Expert simplifies this task. Processor Expert supports set of components that provides properties to configure USB and layered acces correspodning to USB specification including class components. Each component handles corresponding static files from Kinetis SDK, compiler settings and generates code according to the selected properties. This document describes how to write USB applications in Processor Expert.

Processor Expert Software is a development system to create, configure, optimize, migrate, and deliver software components that generate source code for Freescale silicon. For Processor Expert web page: www.freescale.com/processorexpert

The Kinetis software development kit (SDK) is an extensive suite of robust peripheral drivers, stacks, middleware and example applications designed to simplify and accelerate application development on any Kinetis MCU. For Kinetis SDK web page: www.freescale.com/ksdk.

Kinetis SDK source structure contains complete API to acces Freescale MCUs and also USB stack (HOST/DEVICE/OTG modes) implementation source files. Kinetis SDK USB stack is divided to layers and for these layers were created Processor Expert (PEx) components. The main function of every PEx USB component is adding source file code to the project (linked or standalone mode) and creating USB stack configuration files.

Processor Expert is fully integrated into Kinetis Design Studio. The Kinetis Design Studio IDE is a complimentary integrated development environment for Kinetis MCUs that enables robust editing, compiling and debugging of your designs. Based on free, open-source software including Eclipse, GNU Compiler Collection (GCC), GNU Debugger (GDB), and others.

For Kinetis Design Studio web page: www.freescale.com/kds

Kinetis Design Studio is released as only base product, this means it does not contain SDK support by default and it is necessary to add SDK support by installing corresponding service pack (eclipse update), this update is available in each SDK in tools directory.

1.1 PEx USB stack structure

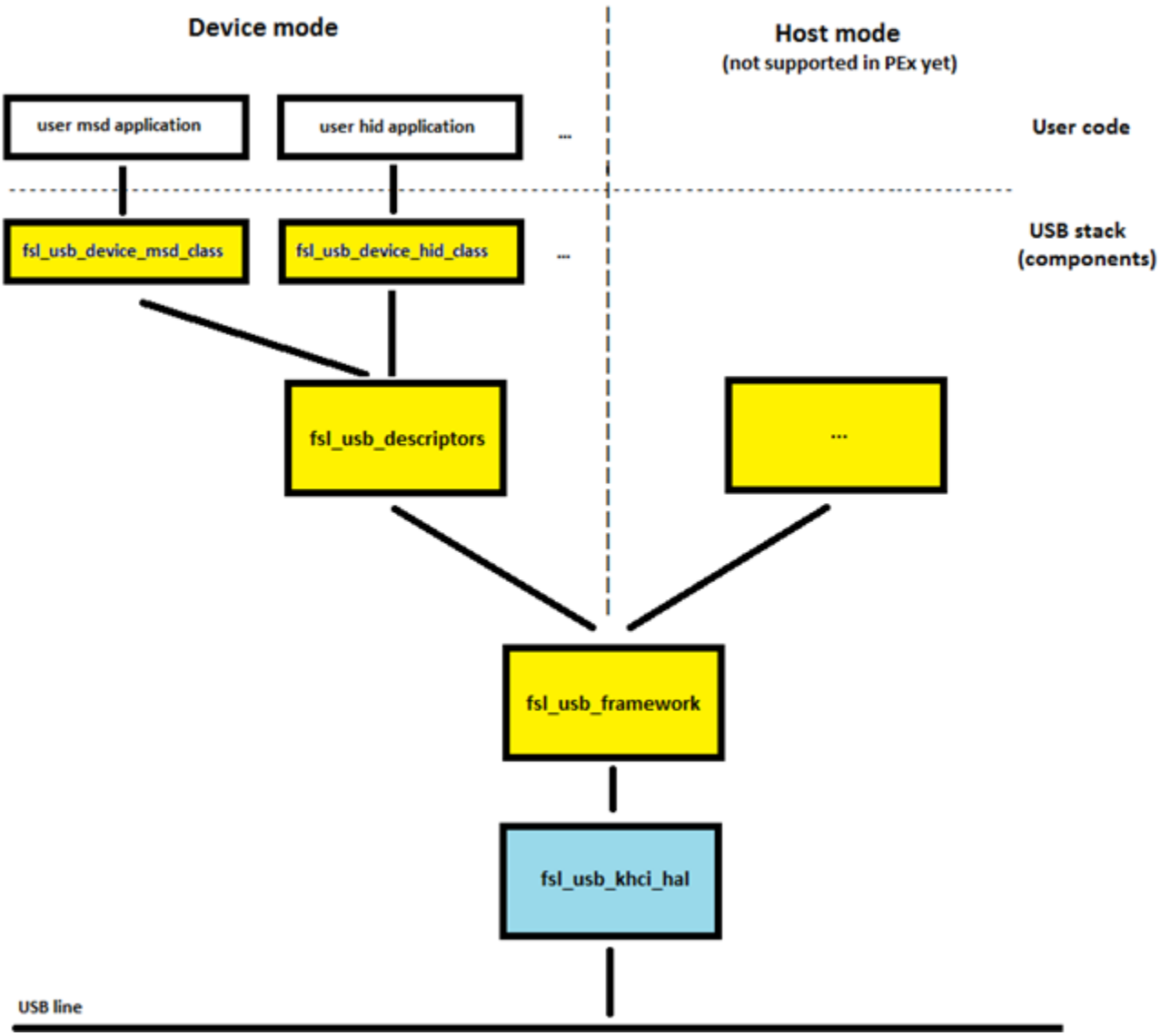

Figure 1-1. USB stack structure

Chapter 2
Processor Expert SDK USB layers description

2.1 fsl_usb_device_msd_class component

Component allows:

- USB device mass storage Subclass and Protocol code configuration. Mass storage device class driver supports only SCSI transparent command subclass code and BBB (bulk only transport) protocol code.

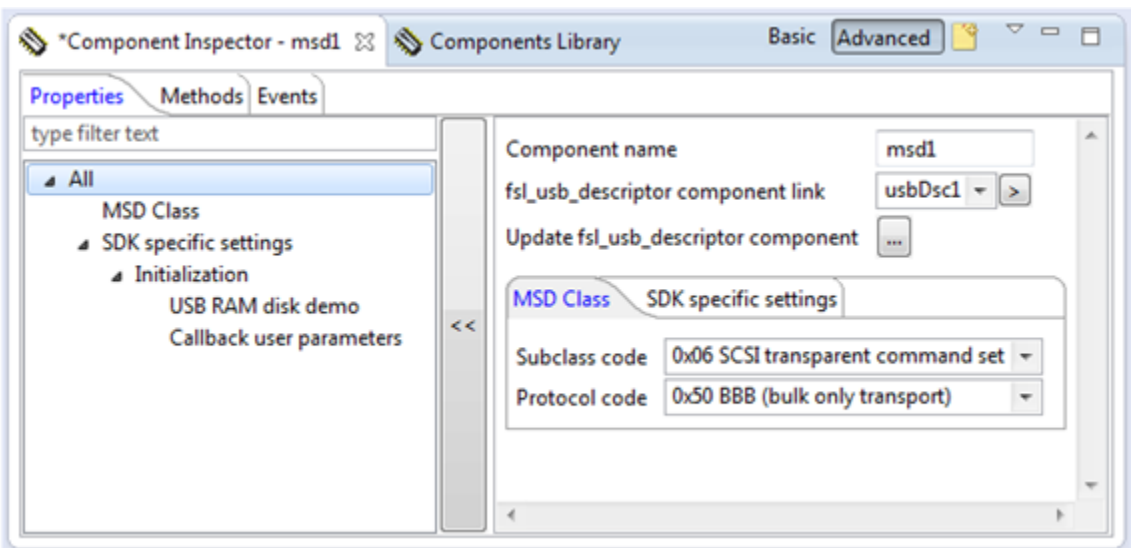

Figure 2-1. MSD Subclass and Protocol code configuration

- Create MSD USB stack configuration structures and variables.

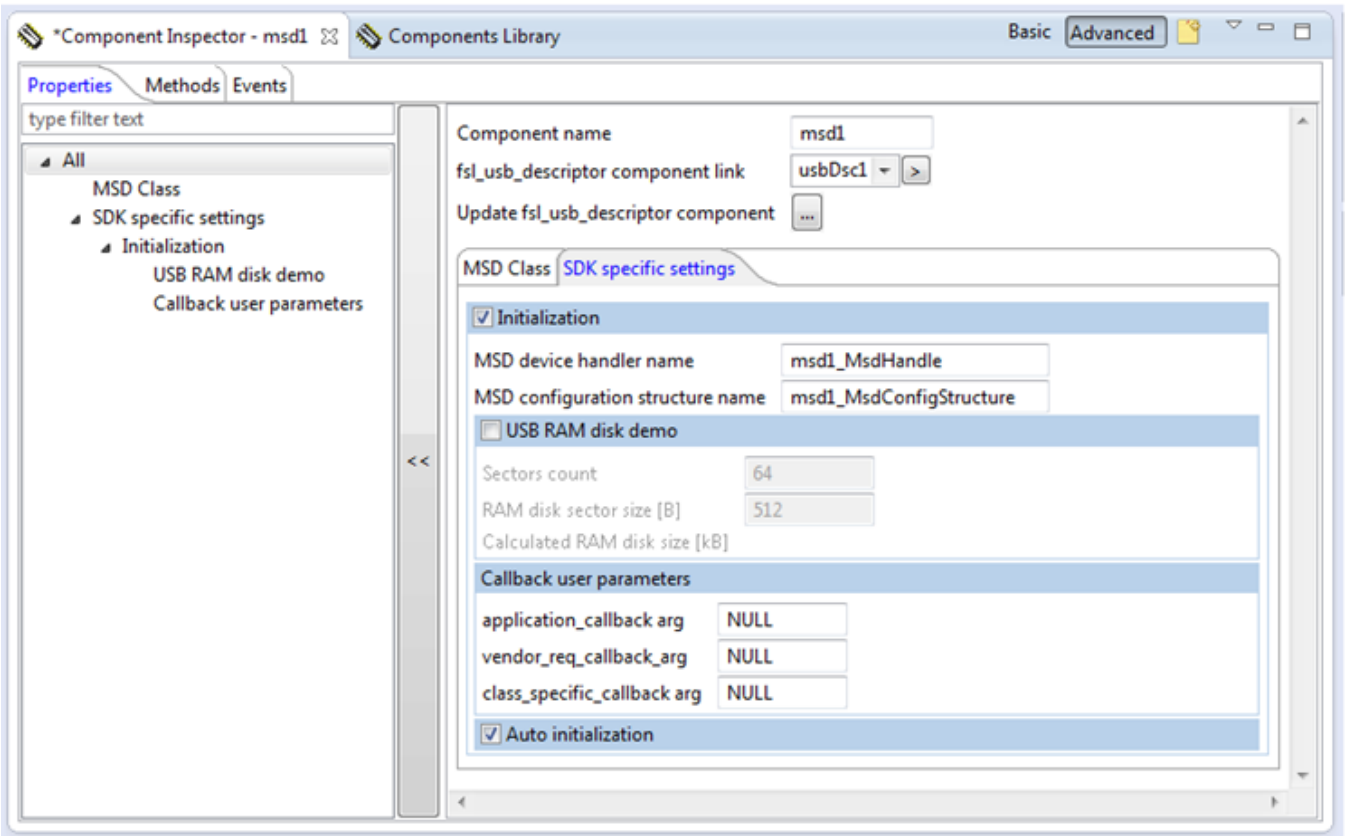

Figure 2-2. Properties for MSD class initialization

- Create C module (default name of C module is *msd1_msd.c* which is stored in **Sources** folder) with USB MSD callbacks. This module contains USB RAM disk demo code which is possible to activate by USB RAM disk demo property, USB RAM disk parameters (sector size, count) are possible to set by properties in component.

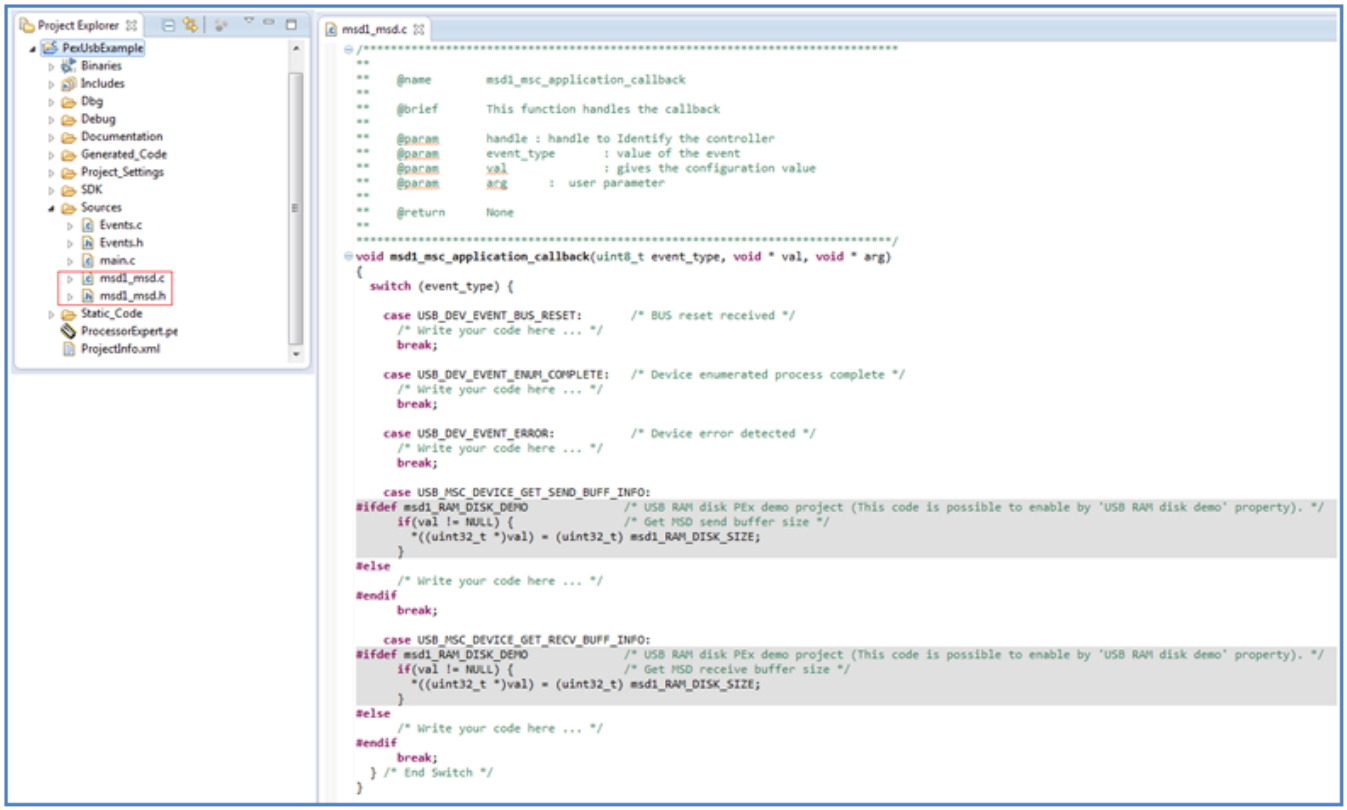

Figure 2-3. MSD class callbacks API (with RAM disk demo code) generated by Processor Expert

- Access MSD API functions (API functions are taken over *usb_class_msc.h* file)

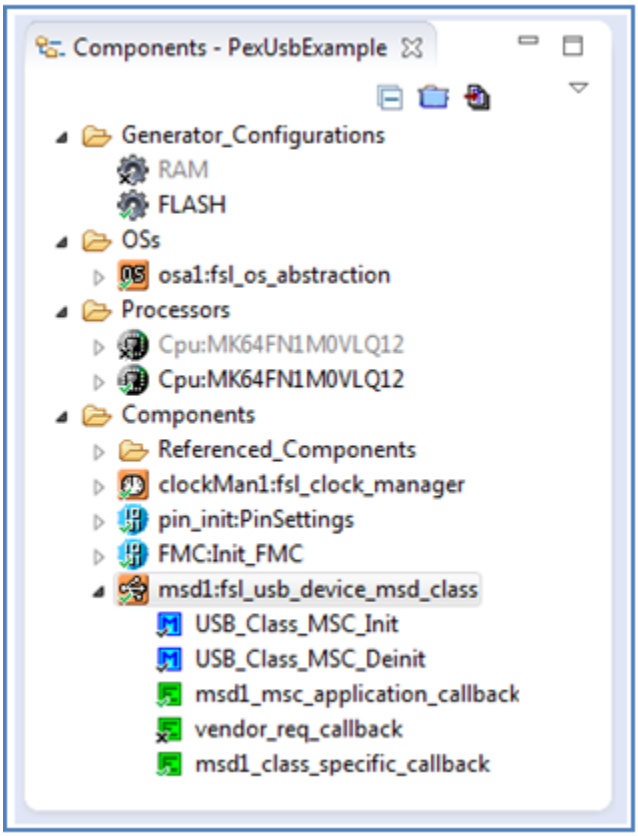

Figure 2-4. MSD class API and callback functions

- Add MSD class requirements to *fsl_usb_descriptor* component (class type, desired endpoints).
- Add Kinetis SDK USB stack MSD driver files to the project.

2.2 fsl_usb_device_hid_class component

Component allows:

- USB device HID Subclass and Protocol code configuration.

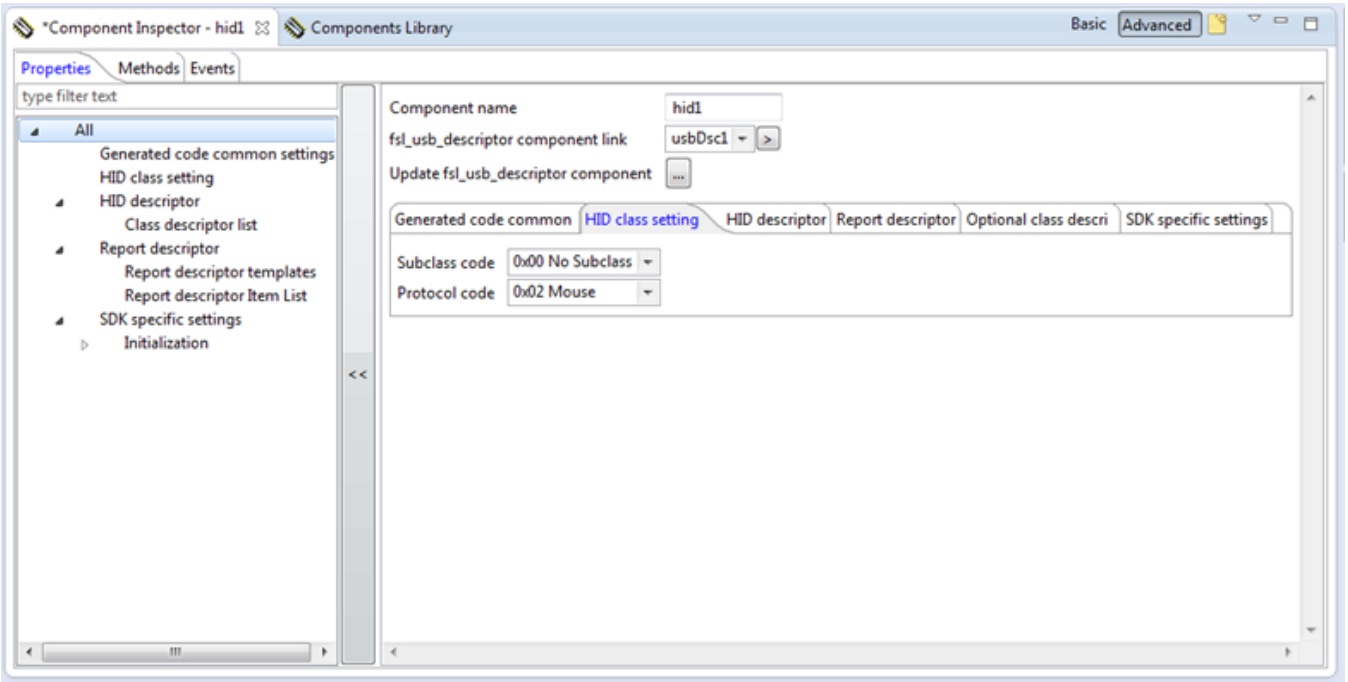

Figure 2-5. HID Subclass and Protocol code configuration

- HID report descriptor definition. *fsl_usb_device_hid_class* component contains two predefined report descriptor template: Standard mouse and keyboard.

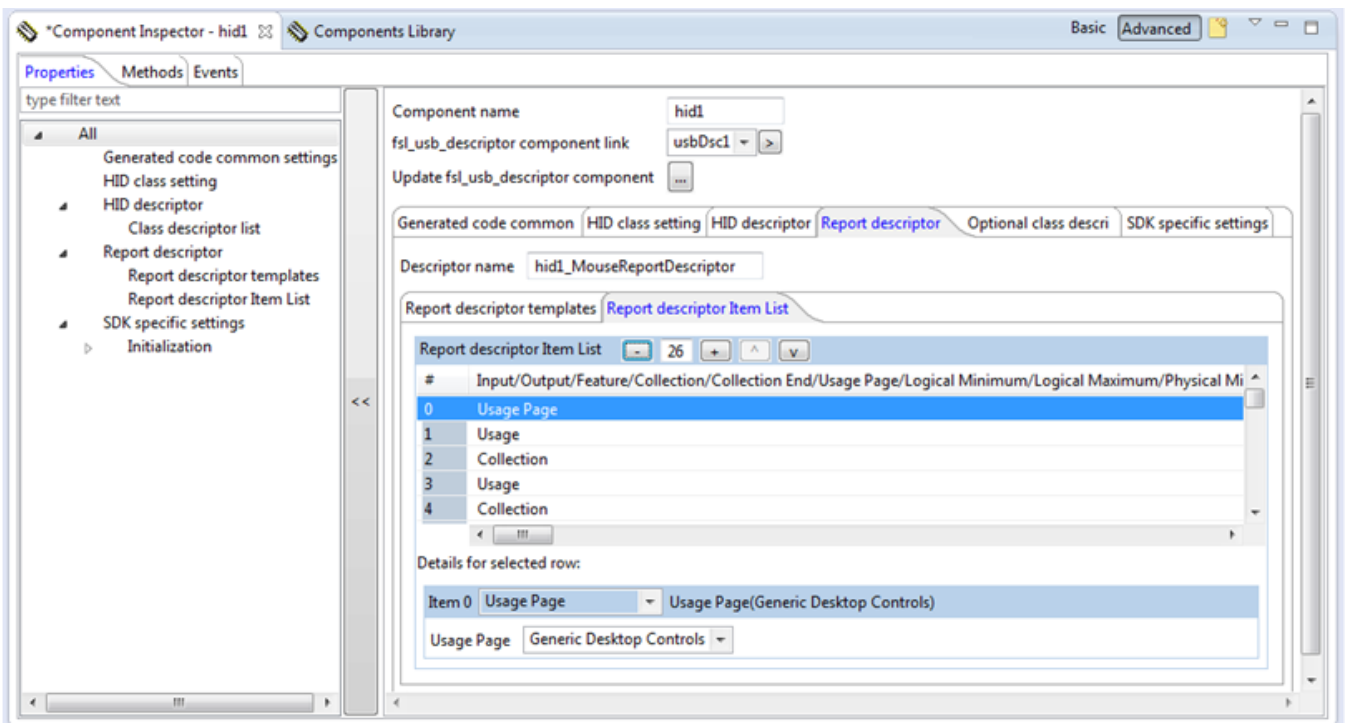

Figure 2-6. HID Report descriptor definition

- Create HID USB stack configuration structures and variables.

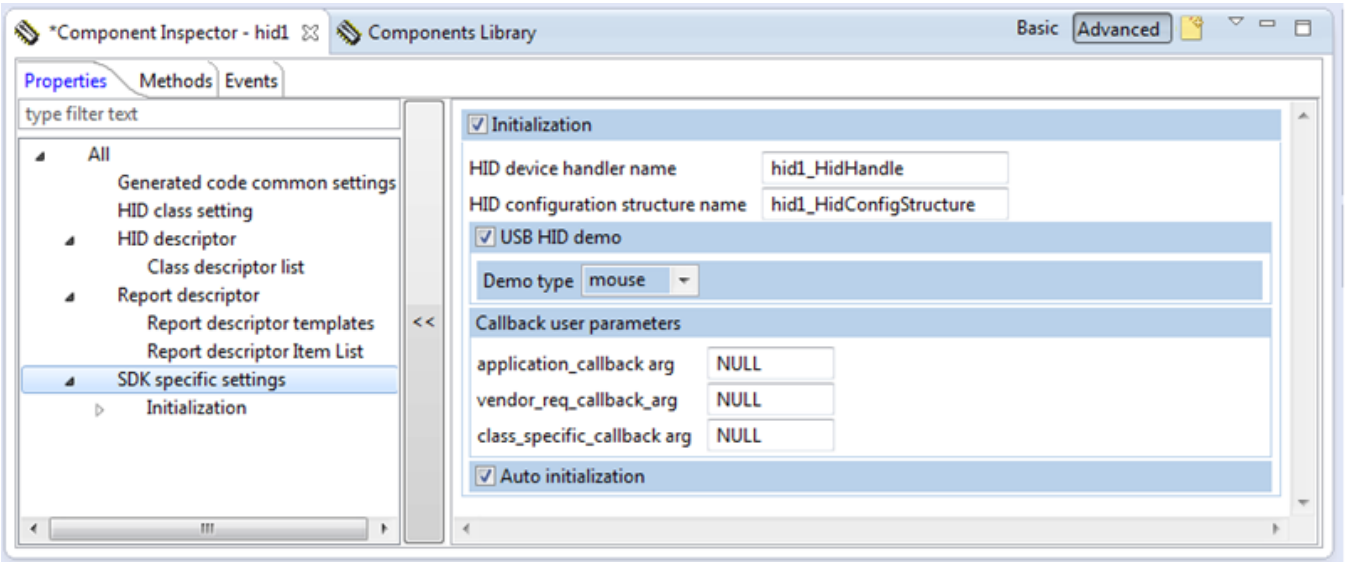

Figure 2-7. Properties for HID class initialization

- Create C module (default name of C module is *hid1_hid.c* which is stored in **Sources** folder) with USB HID callbacks. This module contains USB HID demo code which is possible to activate by **USB HID demo** property, USB HID demo type is possible to set by **Demo type** property in component. Selecting **Demo type** (mouse/keyboard) is automatically changed HID report descriptor and HID protocol code (mouse/keyboard).
- Access to HID Class API functions (API functions are taken over *usb_class_hid.h* file).

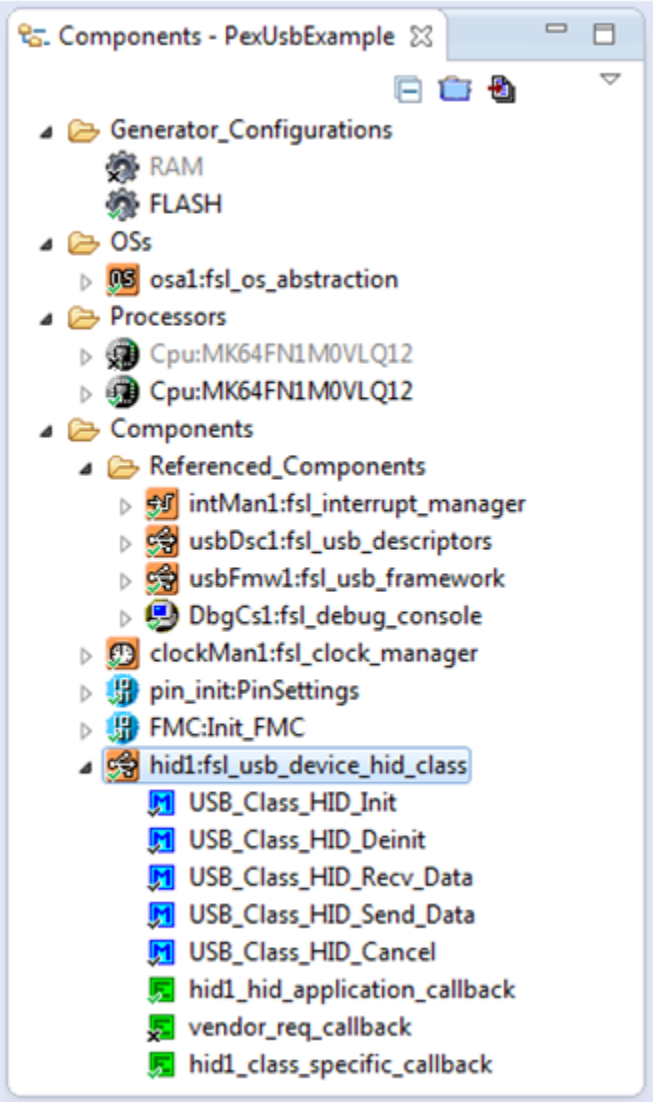

Figure 2-8. HID Class API and callback functions

- Add HID class requirements to *fsl_usb_descriptor* component (class type, desired endpoint). *fsl_usb_device_hid_class* requires only one input-endpoint. If application requires data from Host device (output-endpoint), output-endpoint must be added manually in *fsl_usb_descriptors* component.
- Add Kinetis SDK USB stack HID driver files to the project.

2.3 fsl_usb_descriptors component

The *fsl_usb_descriptors* component is used for USB device mode. Basic function of the *fsl_usb_descriptors* component is collecting requirements (for example, USB device class type, Pipes numbers, and type) of all *fsl_usb_device_XXX_class* components, define endpoint sizes and configure using SDK USB class drivers. On the basis of collected

information, *fsl_usb_descriptors* component creates USB standard and class descriptors structures and creates standard USB function (for example, *GetDescriptors*, *SetInterface*, *SetConfiguration* etc.).

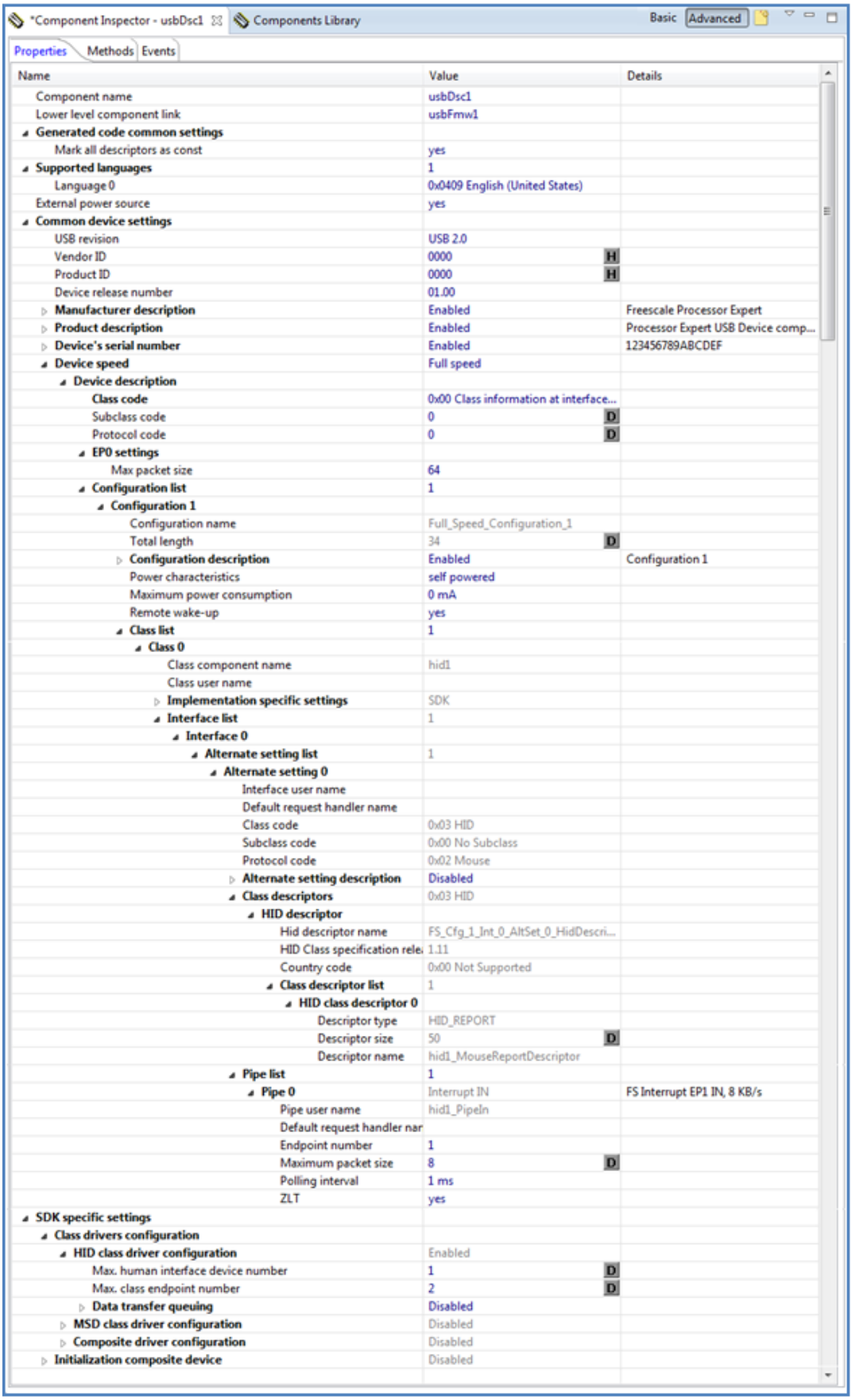

Figure 2-9. fsl_usb_descriptors component

Component allows:

- Create C module (default name of C module is *usbDsc1.c* which is stored in **Generated_Code** folder). *usbDsc1.c* file contains description of all USB device class component - Device/Configuration/Strings descriptors, structures which describe used endpoints and class types, standard USB device function (*GetDescriptors*, *SetInterface*, *SetConfiguration* etc.).

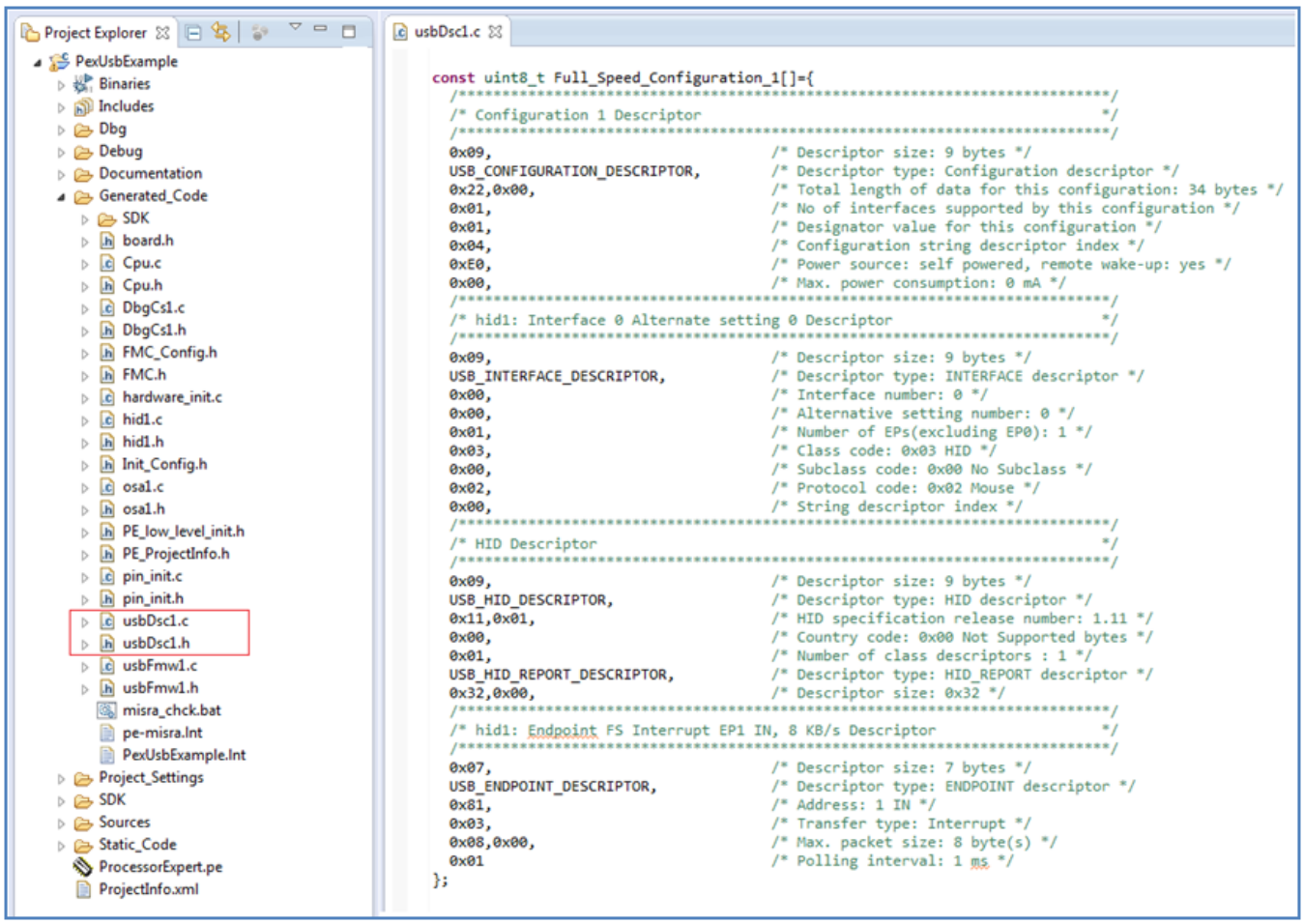

Figure 2-10. Content of usbDsc1.c file, description of USB device

- Access to Composite class API functions (API functions are taken over *usb_class_composite.h* file). Composite class API is enabled automatically when more that one *fsl_usb_device_XXX_class* components are available in project (composite USB device mode).

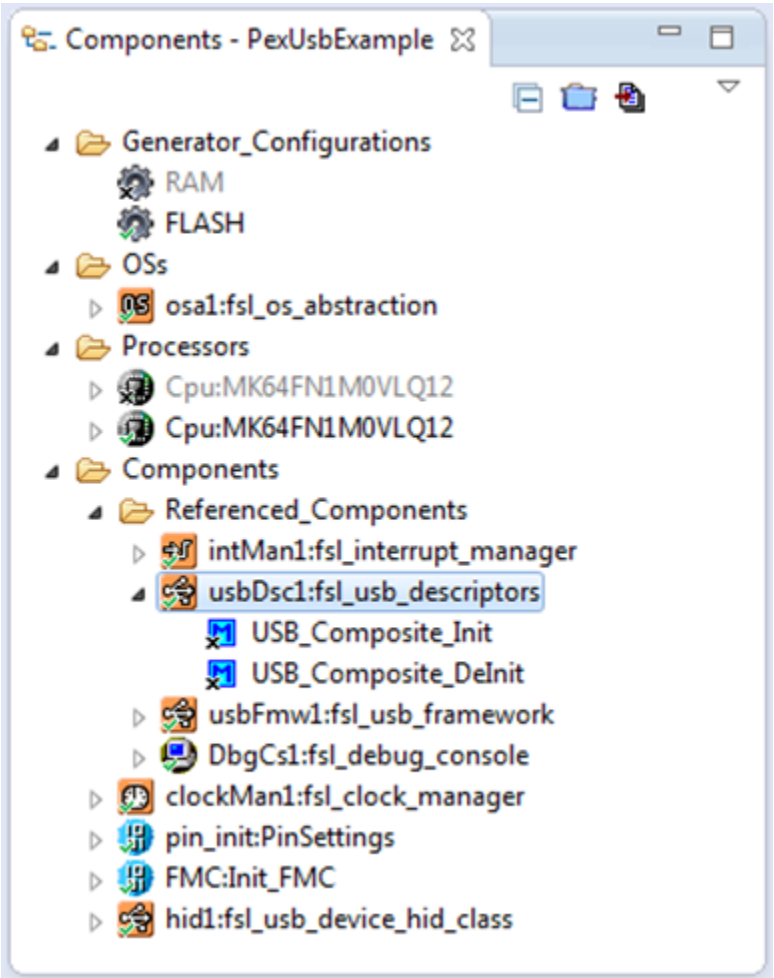

Figure 2-11. Composite class API functions

- Add requirements to *fsl_usb_framework* component (USB mode endpoint count, etc.).
- Add Kinetis SDK USB stack composite driver files to the project (for USB composite mode), create kinetis SDK class configuration files.

2.4 fsl_usb_framework component

The *fsl_usb_framework* component covers bottom layer of the USB interface that transmits and receives packets. The *fsl_usb_framework* component supports three modes: DEVICE, HOST, and OTG. According to the selected mode *fsl_usb_framework* component adds SDK USB stack source files to the project and generates configuration and BSP files. BSP file contains code for USB module timing and interrupt settings.

- The *fsl_usb_framework* component is possible to use as standalone component in a project to create user USB stack (upper USB stack layers can be created by user).

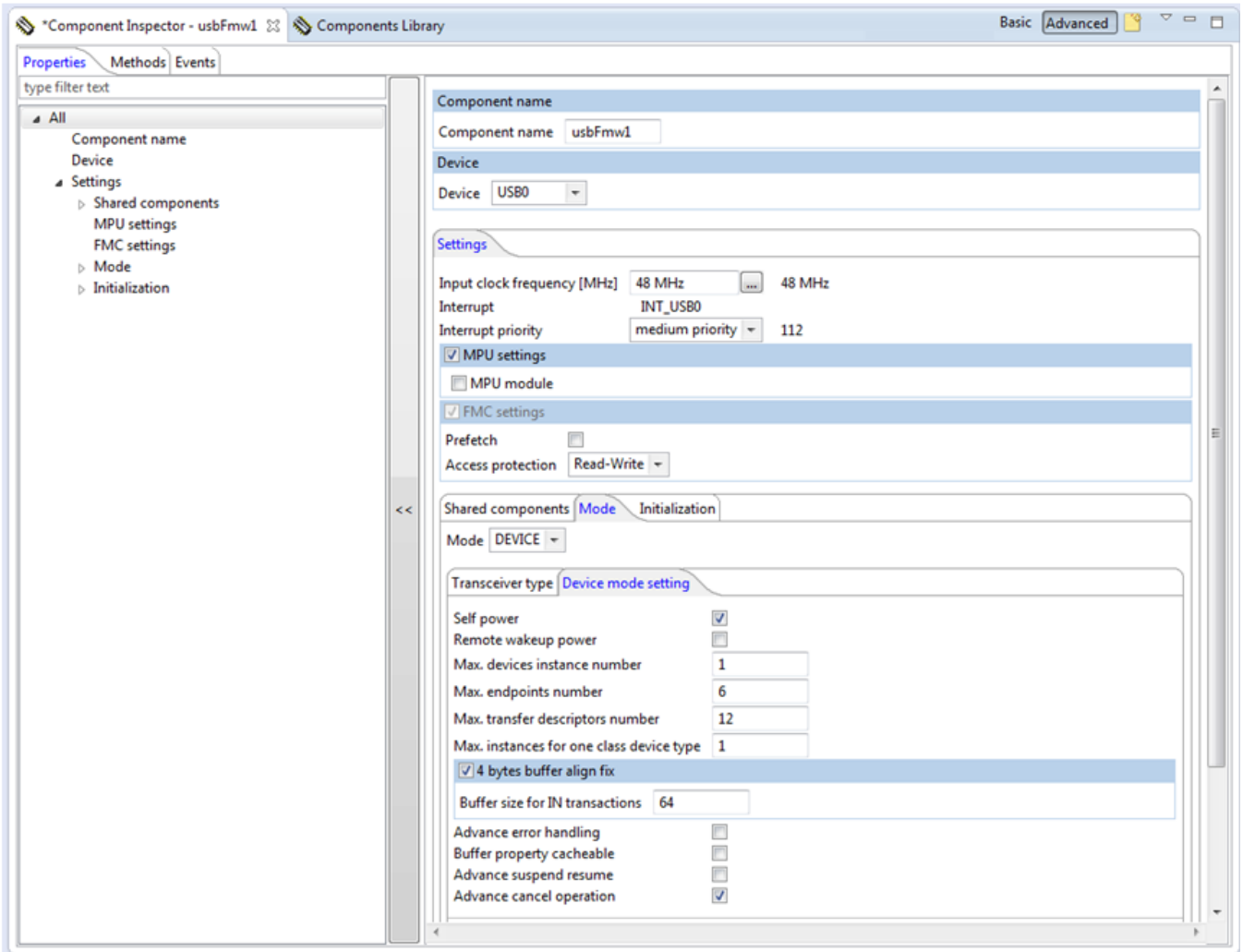

Figure 2-12. USB DEVICE mode configuration

- Create configuration structures and variables for USB DEVICE or HOST mode (for standalone component using).

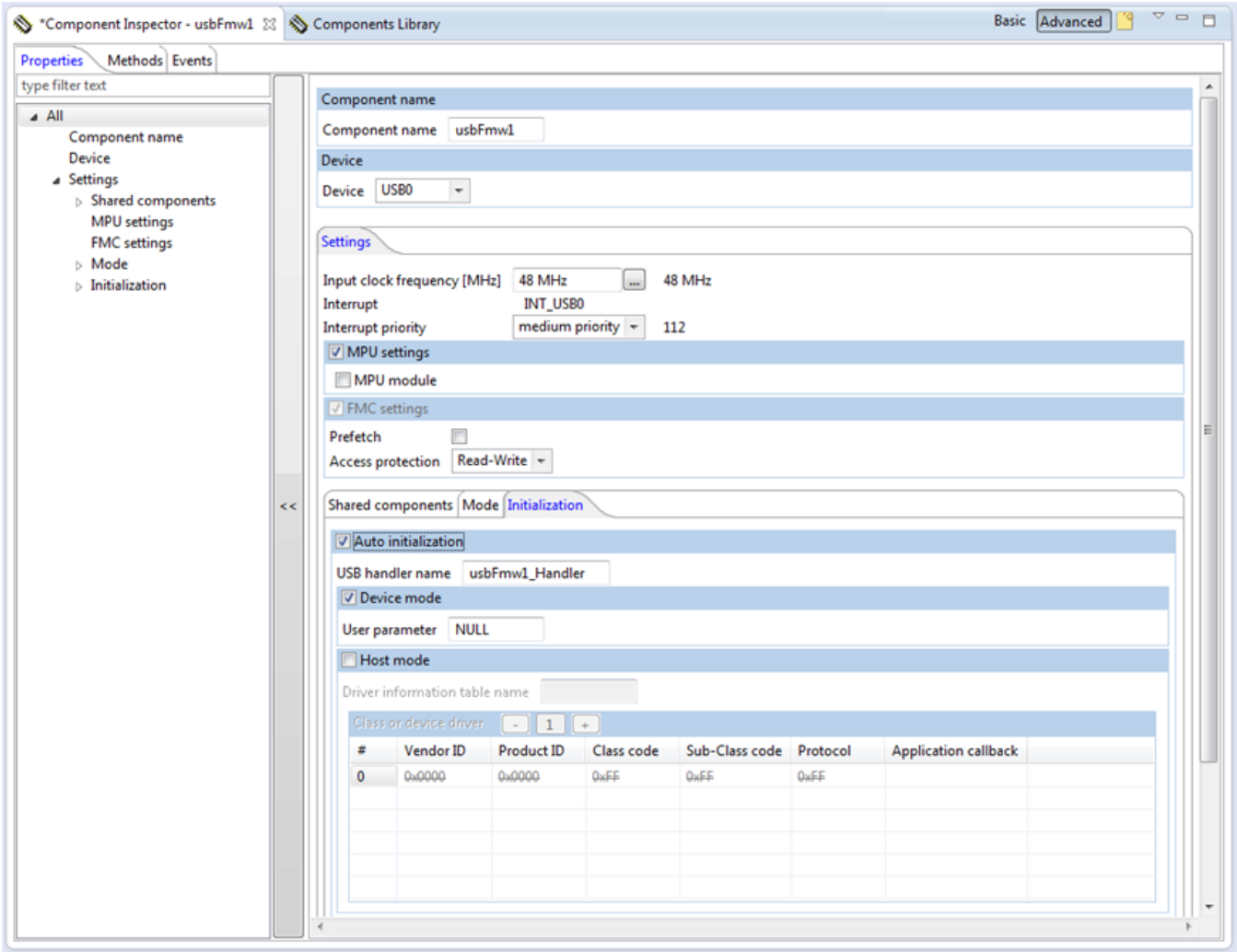

Figure 2-13. Properties for DEVICE mode initialization

- Access to *usb_framework* API functions (API functions are taken over *usb_device_stack_interface.h* file for DEVICE mode, for HOST mode over *usb_host_stack_interface.h* file).

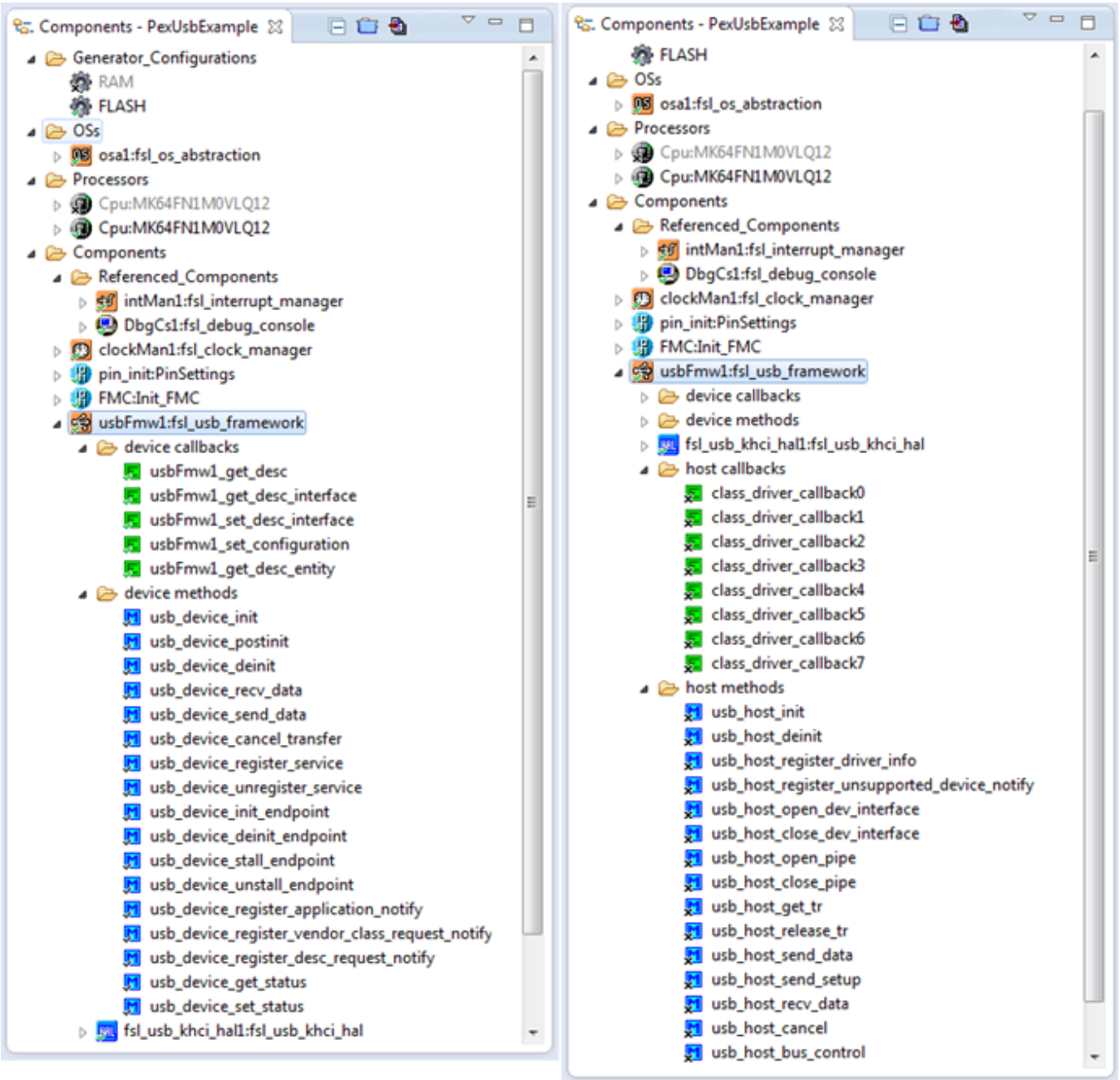

Figure 2-14. USB HOST/DEVICE mode API and callback functions

2.5 fsl_usb_khci_hal component

It provides basic I/O macros for access to the USB periphery (USB periphery read/write register access operation). Standalone use of *fsl_usb_khci_hal* component is not allowed. The *fsl_usb_khci_hal* layer uses modules from higher layers of Kinetis SDK USB stack.

Chapter 3
Creating PEx USB project

To create PEx SDK project:

1. Create Processor Expert SDK project with selected MCU MK64FN1M0 (TWR-K64F120M).

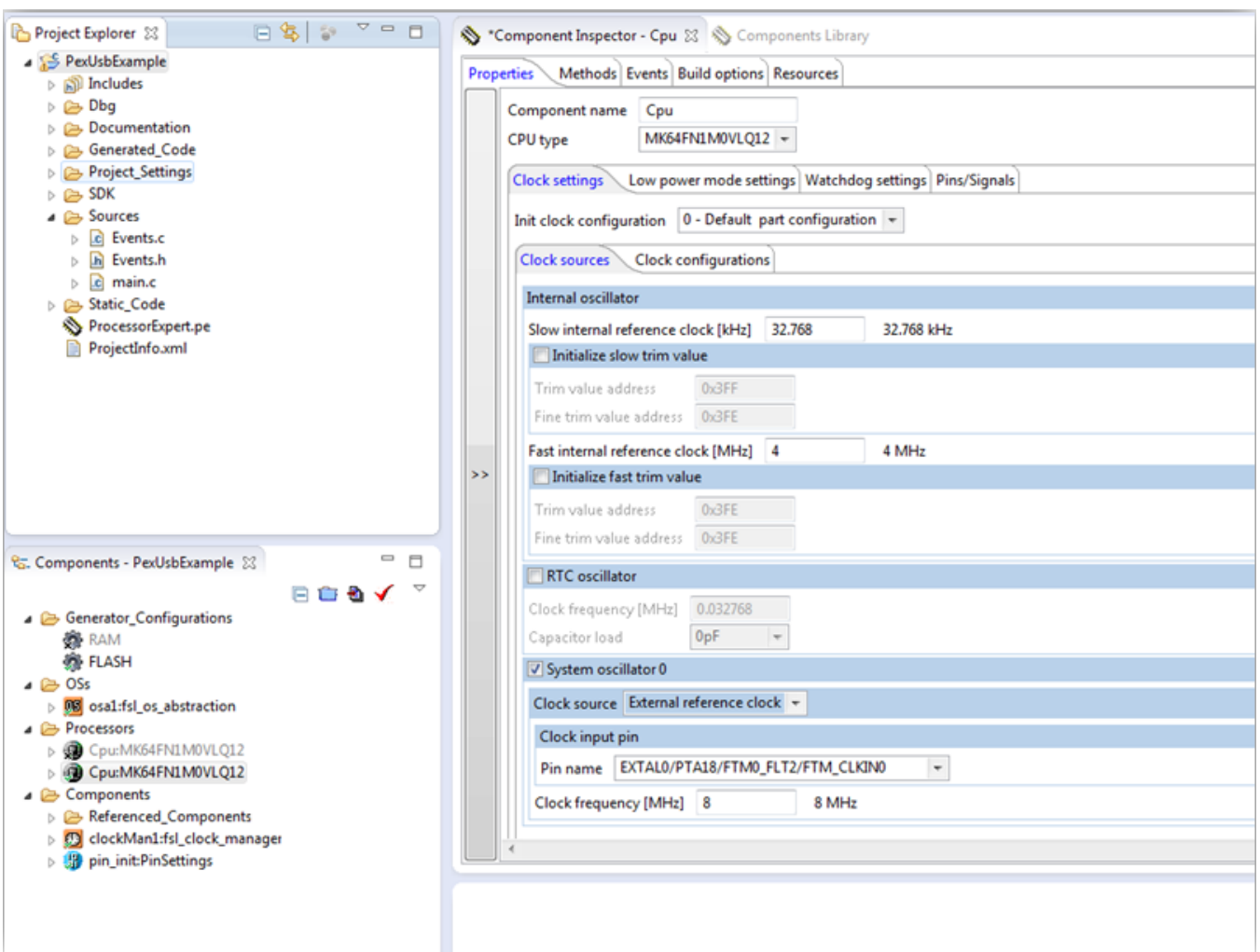

Figure 3-1. Created PEx project - default settings

2. USB full speed (FS) OTG controller requires 48MHz input clock source which depends on CPU clock setting (in case, if USB module is clocked from internal clock source). Default (after reset) Cpu clock settings is from internal oscillator and with using FLL module => Default Cpu core clock value is about 20.97152 MHz. This frequency value is not possible to use for USB controller, default Cpu clock settings must be changed.
    - On Cpu component, enable **System oscilator 0** property, select Clock source = External reference clock, set Clock frequency [MHz] to 50 (TWR-K64F120M board contains 50MHz external oscillator, see scheme for TWR-MK64F120M board).

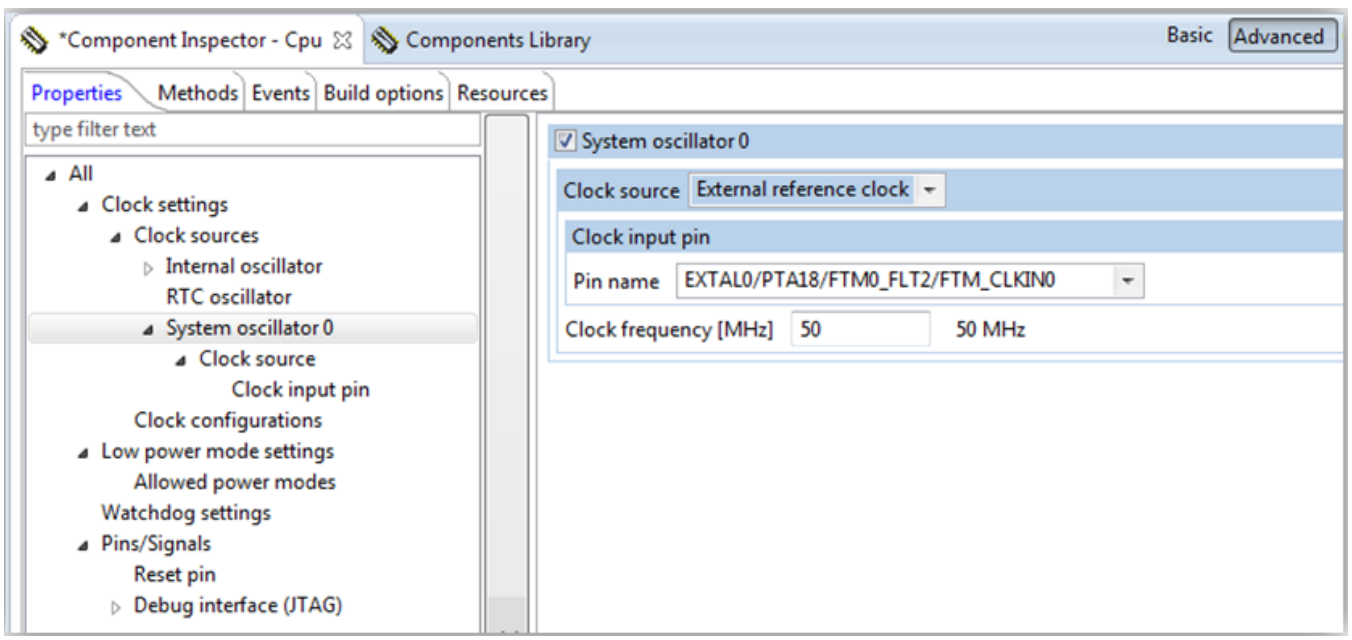

Figure 3-2. System oscillator 0 configuration

- On Cpu component, select **PEE MCG** mode and set **PLL output clock [MHz]** to 120.

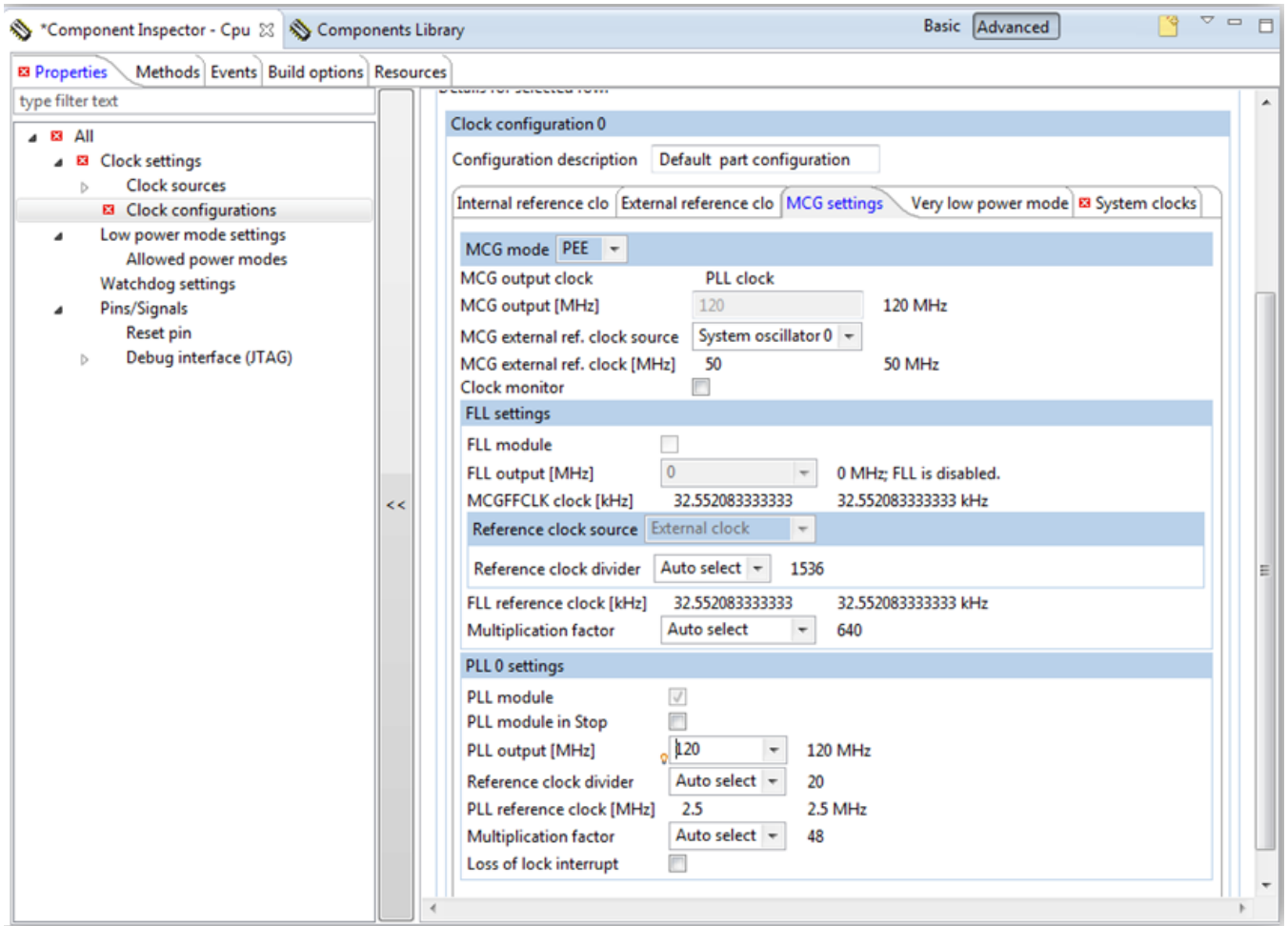

Figure 3-3. MCG module configuration

- Set Cpu clocking: **Core clock** to 120 MHz, **Bus clock** to 60 MHz, **External bus clock** to 30MHz, **Flash clock** to 15MHz. Internal clock source for USB module is from PLL/FLL clock output, for USB controller applies: 120MHz/(5/2) = 48MHz.

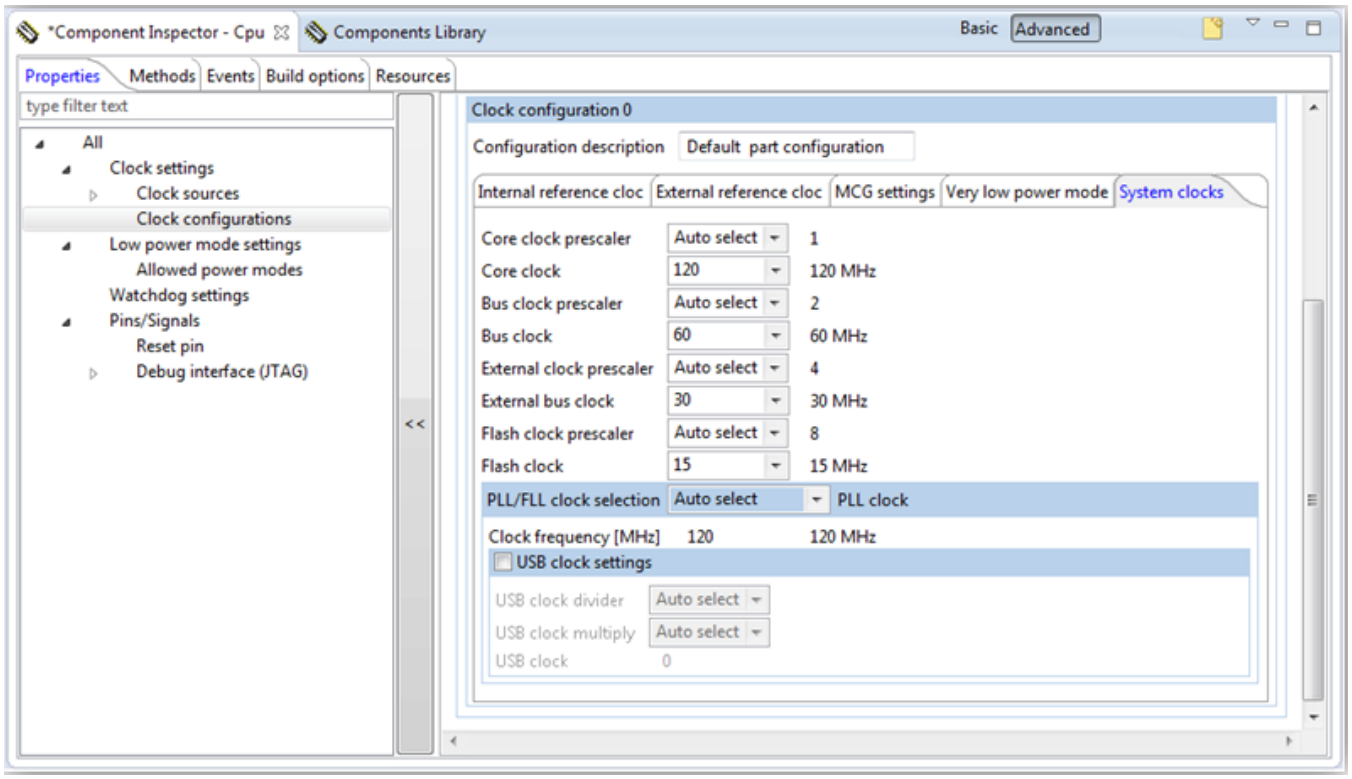

Figure 3-4. Cpu "System clocks" configuration

3. USB OTG controller (BUS master) can also access Flash memory (BUS slave) is through crossbar switch (see Reference Manual for selected Cpu). The crossbar switch connects bus masters and bus slaves using a crossbar switch structure.

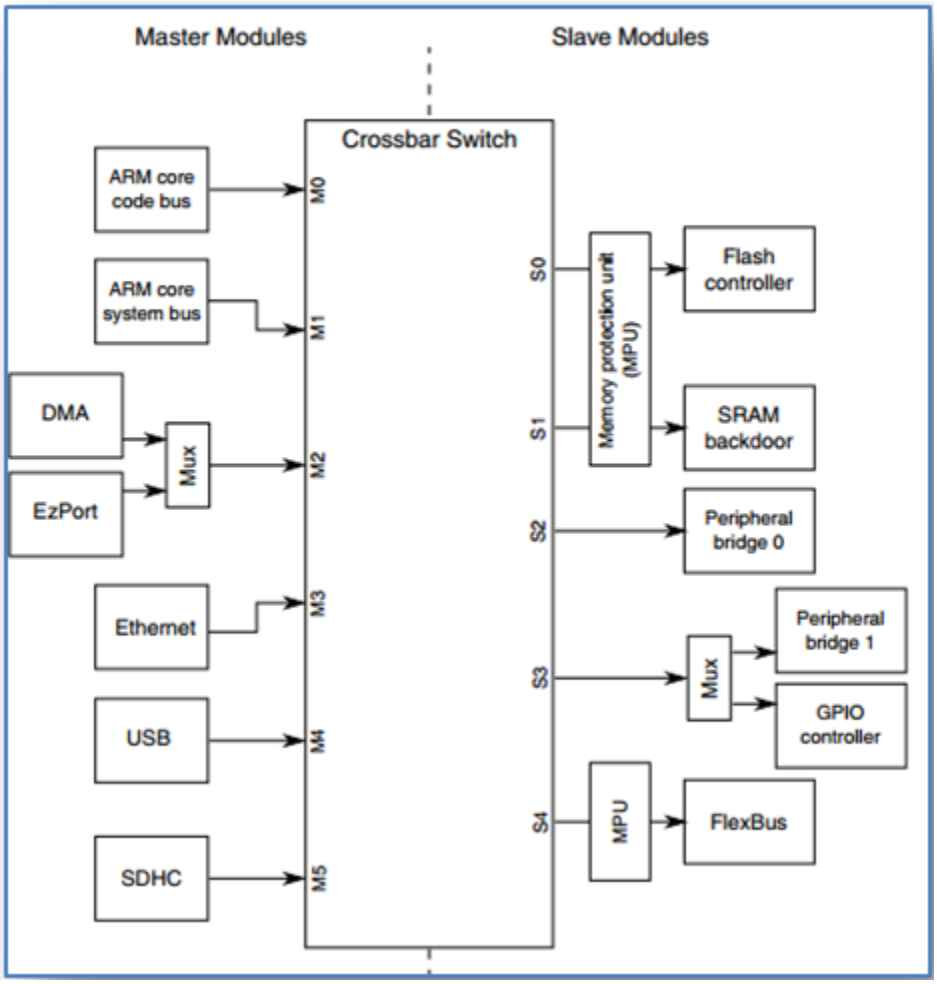

Figure 3-5. Crossbar switch integration

USB stack drivers can access to variables which can are stored in the Flash memory (for example, USB device/configuration descriptors,…). By default, USB controller has no access to flash memory. Therefore, access must be permitted in Flash Memory Controller (FMC). You can use *Init_FMC* component to change default settings.

Add *Init_FMC* component to the project from the component library:

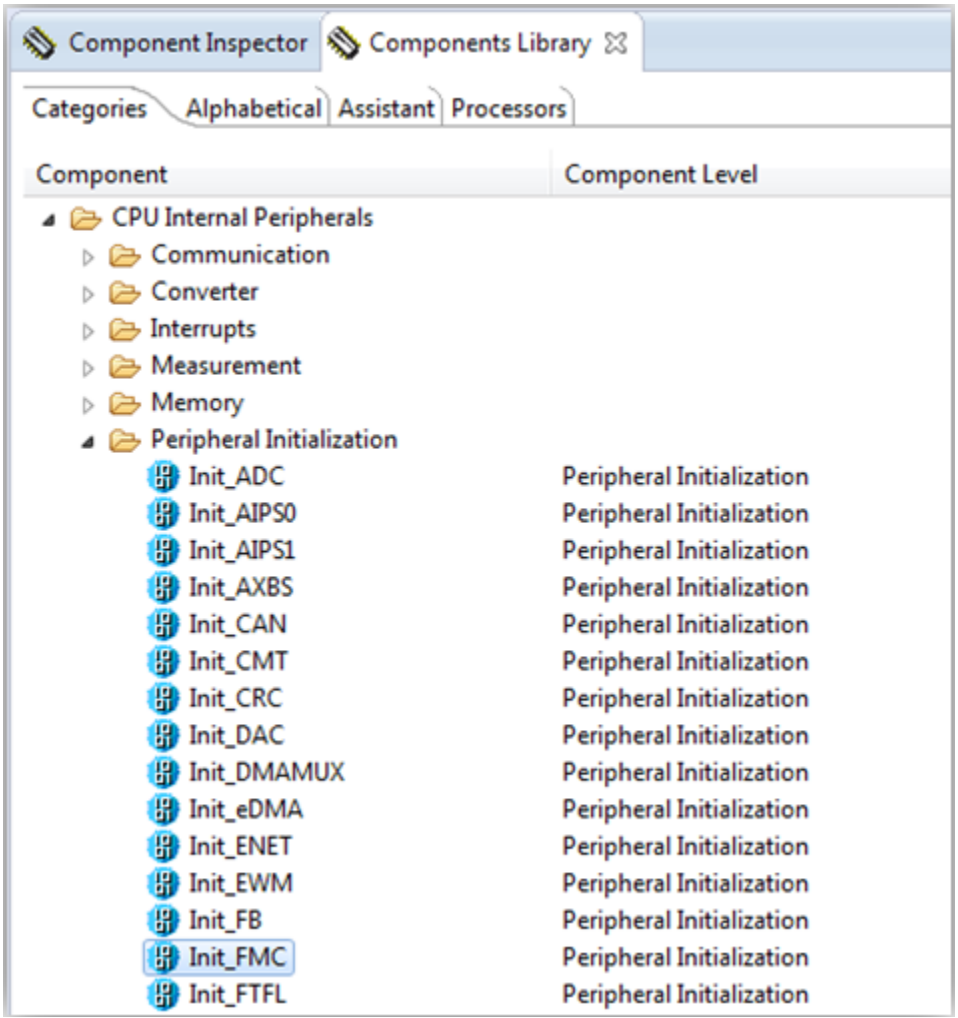

Figure 3-6. Init_FMC component in Components Library view

In *Init_FMC* component, set Access protection for Master 4 (USB_OTG module) to Read only. Now, USB controller has Read only access to flash memory.

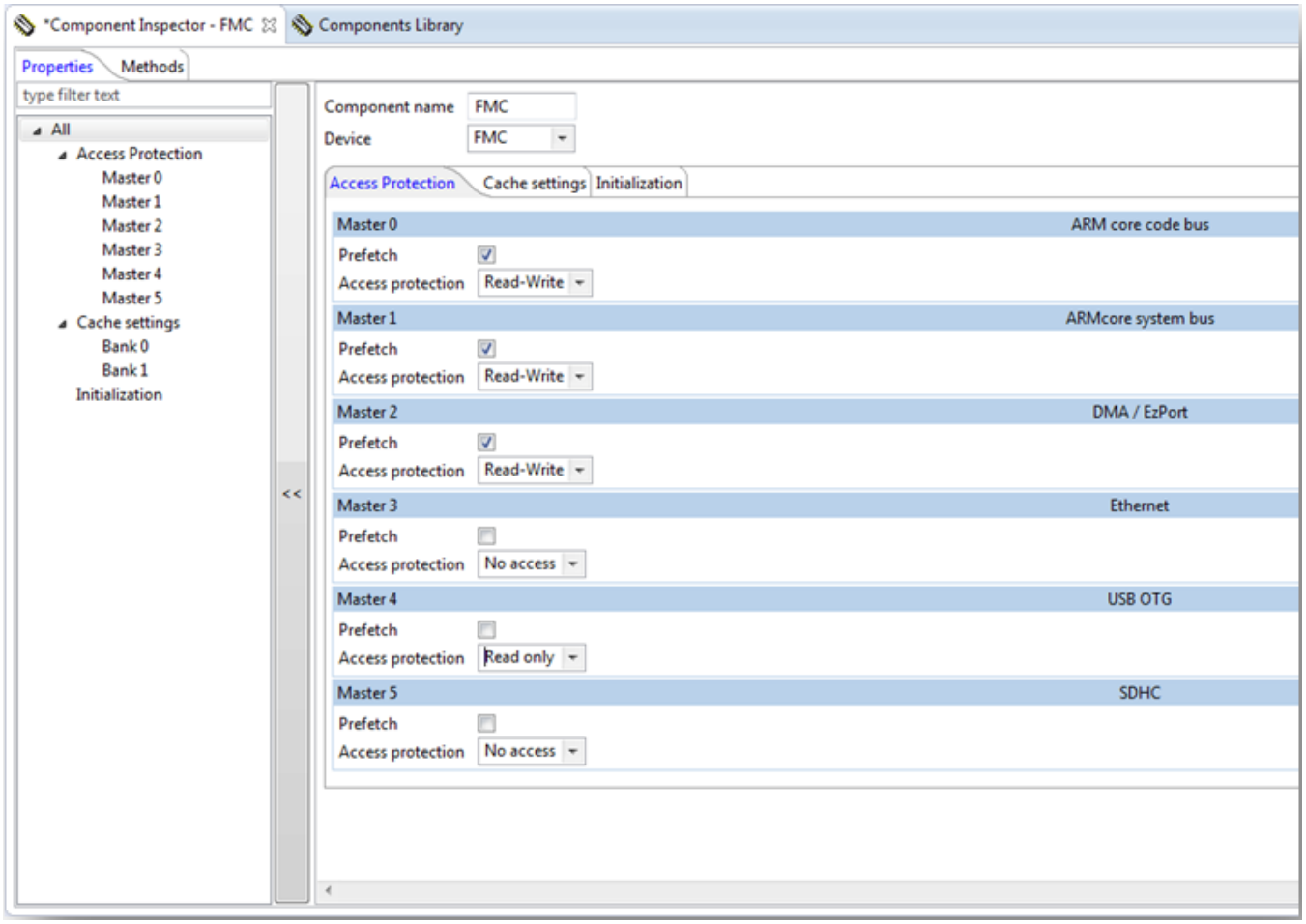

Figure 3-7. Access protection configuration

4. USB FS/LS transceiver module requires voltage source (3.3V) which is provided by USB regulator output. Input of USB regulator is connected to Vregin pin (on MCU package). This PIN can be connected to USBvbus line (USB transceiver is powered from USB host device, for example, PC).

For correct USB device project functionality on TWR-MK64F120M board, ensure that pins 1-2 of J19 are connected.

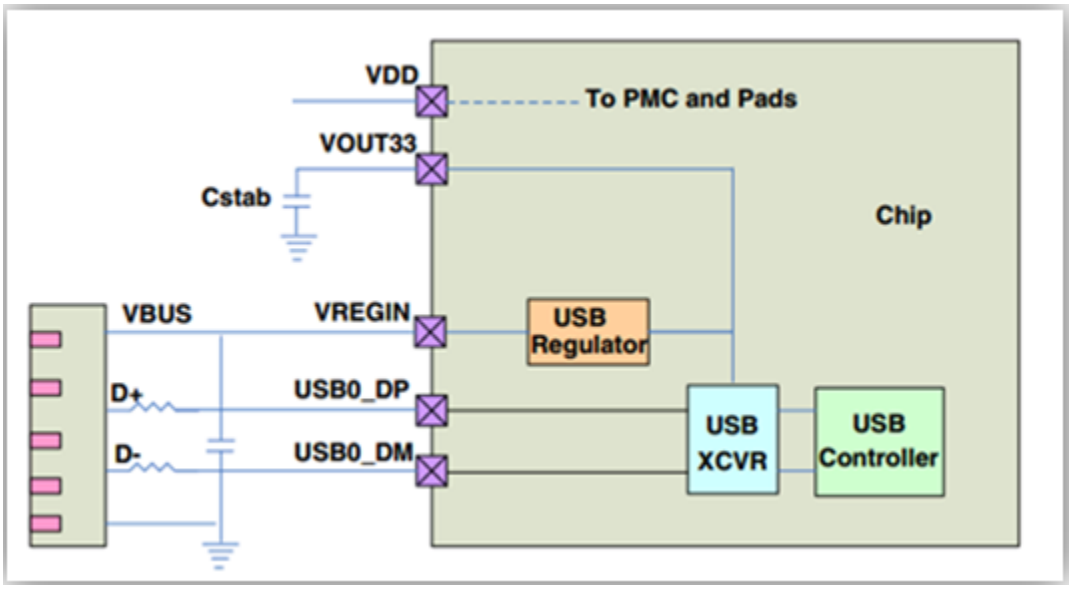

Figure 3-8. USB regulator, VBUS use case

3.1 Creating USB mass storage project

The simplest way to create PEx USB project is simply to insert USB class component (*fsl_usb_device_hid/msd_class*). In this case, all required lower USB stack layers components are automatically added to the project by Processor Expert.

1. For USB mass storage project, add the *fsl_usb_device_msd_class* component from the **Components Library** view to the current project (see Creating PEx USB project).

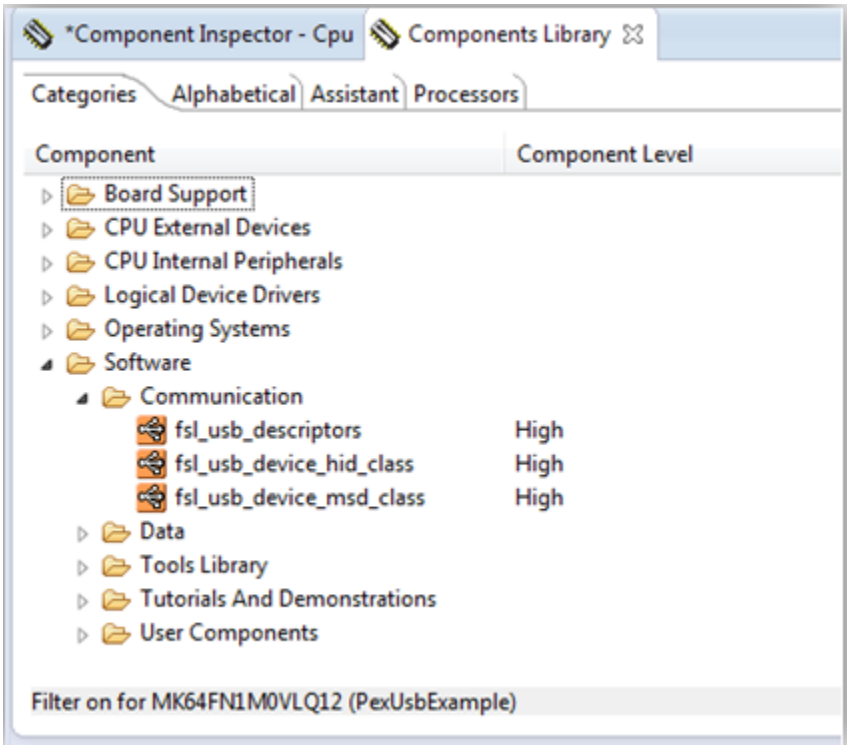

Figure 3-9. USB components in Component Library view

Function of *fsl_usb_device_msd_class* driver requires presence of other USB layers: *fsl_usb_descriptors*, *fsl_usb_framework* (see PEx USB stack structure) and other SDK drivers (*fsl_clock* and *interrupt_manager* etc). All these components are automatically added to the project.

2. Configure *fsl_debug_console* component. Select **Device** as **UART5** and **Baud rate** as **115200**. Also, select Rxd & Txd pins.

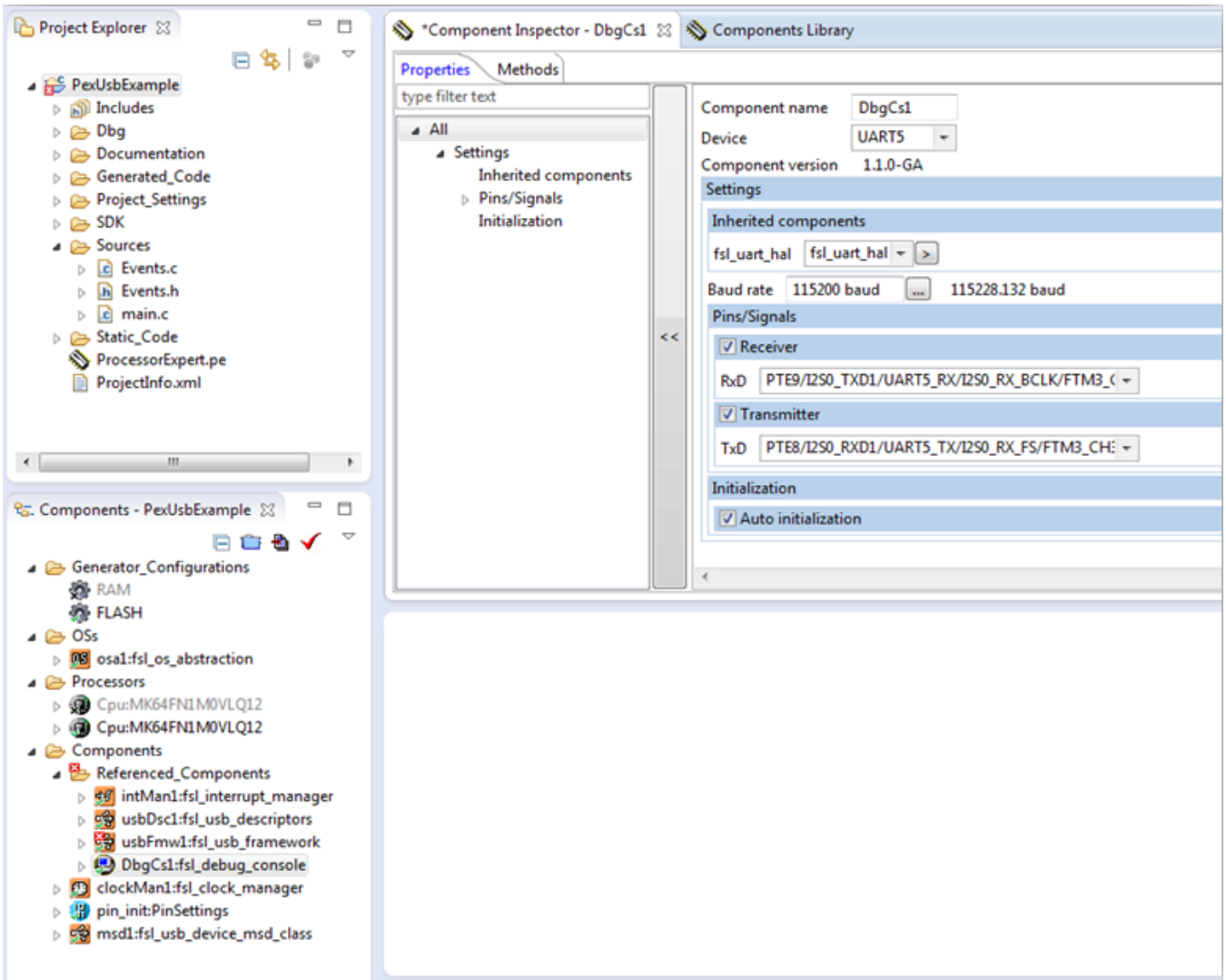

Figure 3-10. fsl_debug_console component configuration

3. In **MSD Class** tab, configure **Subclass code** and **Protocol code**. Only SCSI transparent command and BBB (bulk only transport) are currently supported. Select *fsl_usb_device_msd_class* component's Subclass and Protocol code settings.

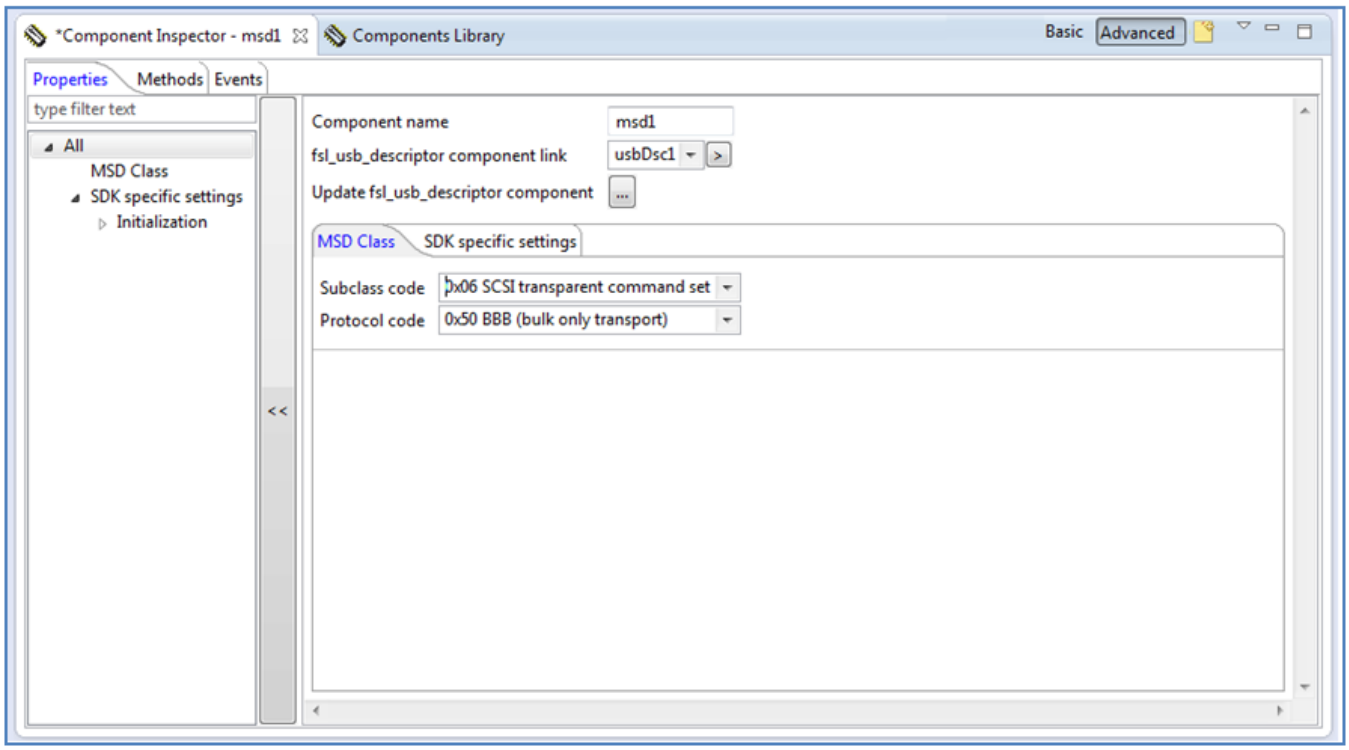

Figure 3-11. MSD Subclass and Protocol code configuration

4. In **SDK specific settings** tab:
   - Select **Initialization** and **Auto initialization** checkboxes. They should be selected by default.
   - Select **USB RAM disk demo** to use predefined demo code for USB RAM disk demo. Configure **Sectors count** and **Sector size** parameters of RAM disk (if default values are not suitable).

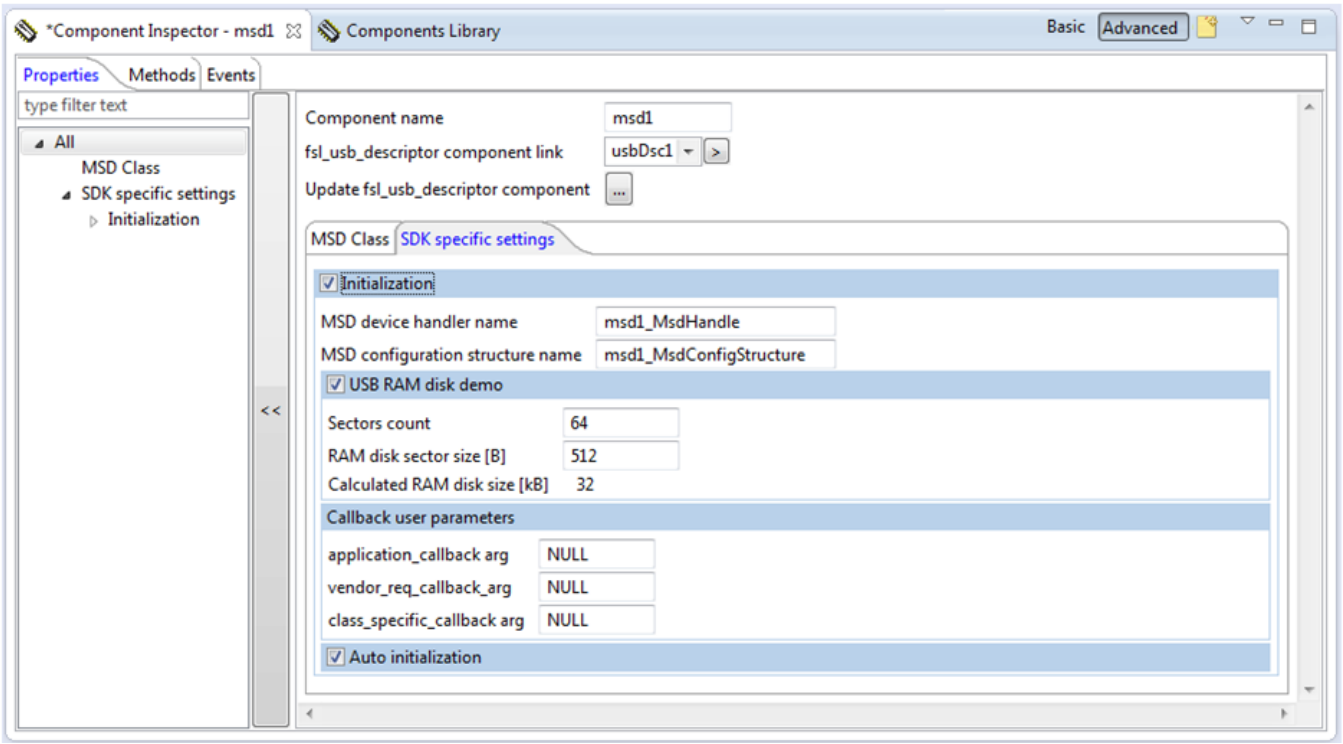

Figure 3-12. Properties for MSD Class initialization

5. Click **Code generation** button. As code is generated, all required files are created/ added in the project.

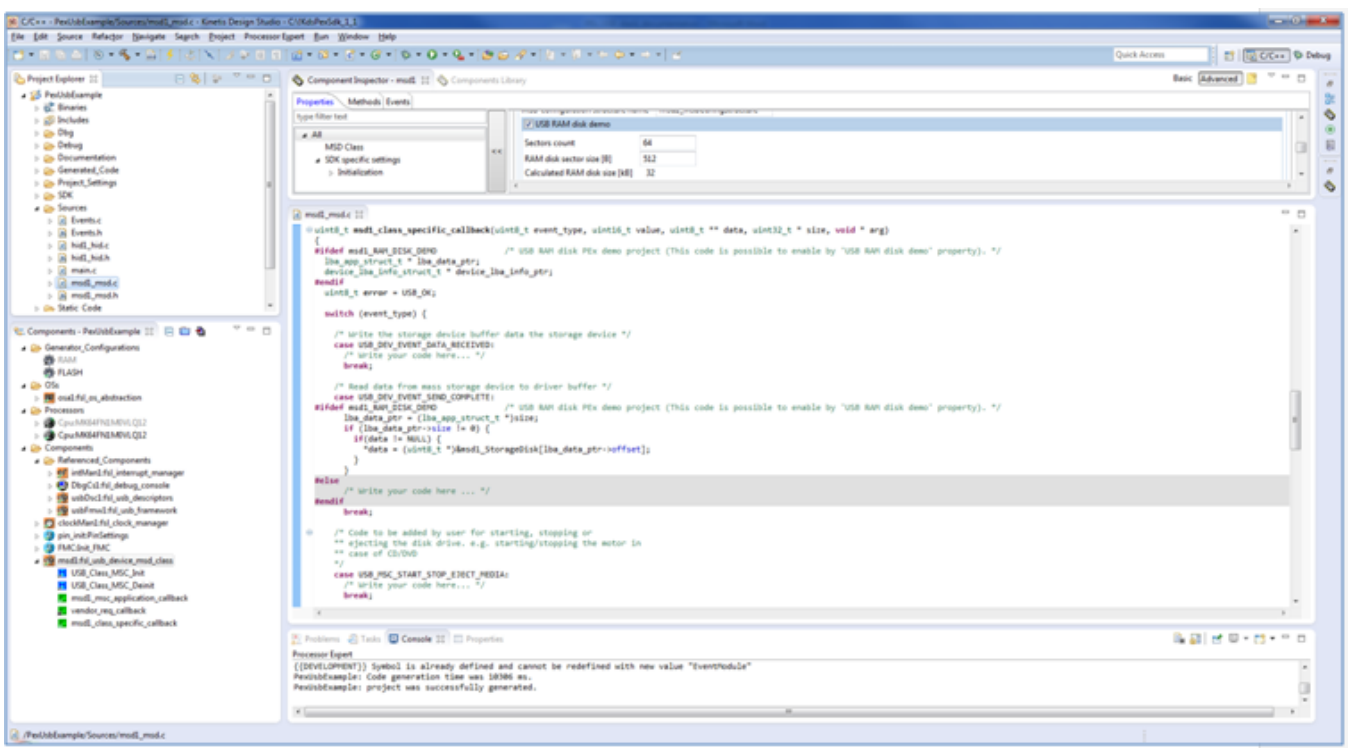

Figure 3-13. MSD class callbacks API (with RAM disk demo code) generated by Processor Expert

6. Build and load USB project into TWR-MK64F120M board.
7. Plug-in the micro USB connector (J17) of TWR-MK64F120M to PC.
8. The windows will prompt you to format the disk.

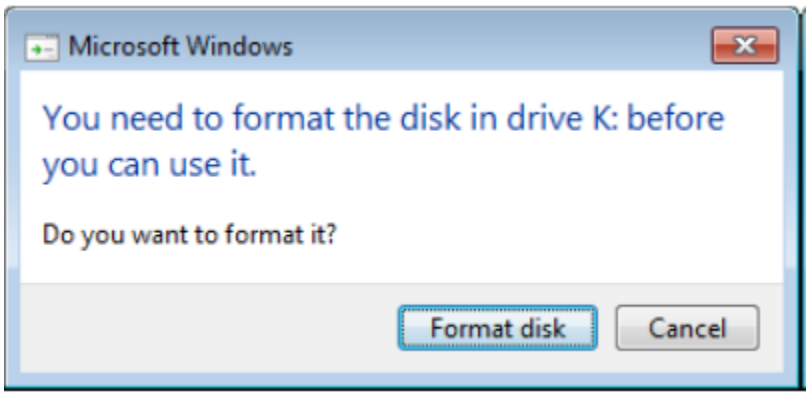

Figure 3-14. OS Windows requires to format disk

9. When the format is completed, the computer will display the capacity of removable disk.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Revision 11/2014