**Freescale Semiconductor**
vfvdcg436

Document Number: PEXKSDKUSBSI
Rev. 0, 07/2015

# Processor Expert Kinetis SDK USB Stack Integration

**Contents**



## 1 Introduction

Processor Expert helps in simplifying the complex task of writing USB applications. Processor Expert supports set of components that provides properties to configure USB and layered access corresponding to the USB specifications including the class components. Each components handles the corresponding static files from Kinetis SDK, requested compiler settings, and generates code according to the selected properties. This document is lists the steps to quickly start writing 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 more information on Processor Expert, see 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 more information on Kinetis SDK, see www.freescale.com/ksdk.

The Kinetis SDK source structure contains the complete API to access Freescale microcontrollers and the USB stack (HOST/DEVICE/OTG modes) implementation source files. The Kinetis SDK USB stack is divided in layers and for these

layers the Processor Expert (PEx) components are created. The main function of each PEx USB component is to add source file code to the project (linked or standalone mode) and create the USB stack configuration files.

Processor Expert is fully integrated into Kinetis Design Studio (KDS). 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 a free, open-source software including Eclipse, GNU Compiler Collection (GCC), GNU Debugger (GDB), and others.

For more information on Kinetis Design Studio, see www.freescale.com/kds.

Kinetis Design Studio is released only as a base product and does not contain SDK support by default. It is necessary to add SDK support by installing corresponding service pack (Eclipse update). The Eclipse udpate is available in each SDK in tools directory.

# 2 PEx USB stack structure

The following figure illustrates the Processor Expert USB stack structure.

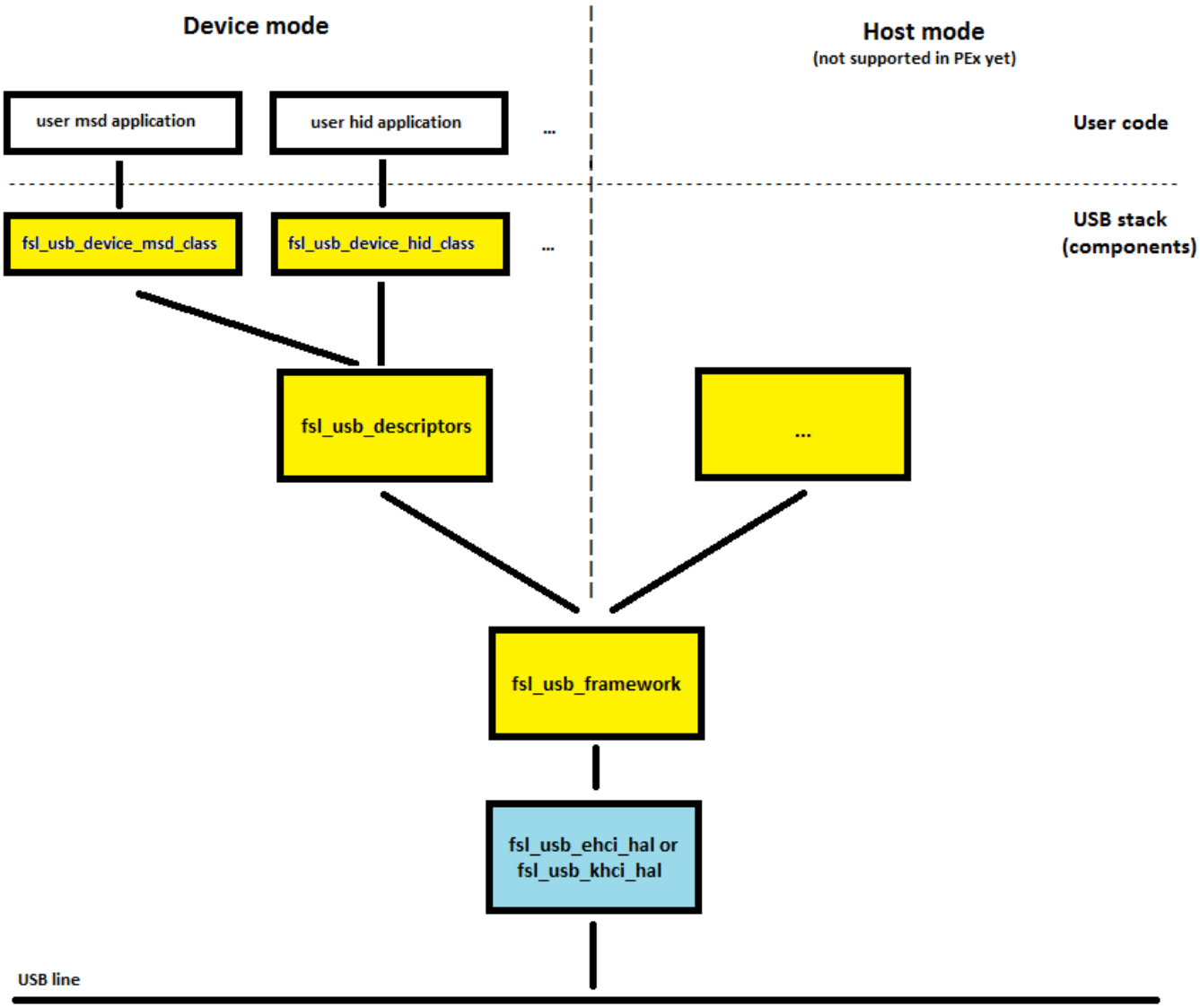


**Figure 1. USB stack structure**

# 3 PEx SDK USB layers description

## 3.1 fsl_usb_device_msd_class component

The component allows:

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

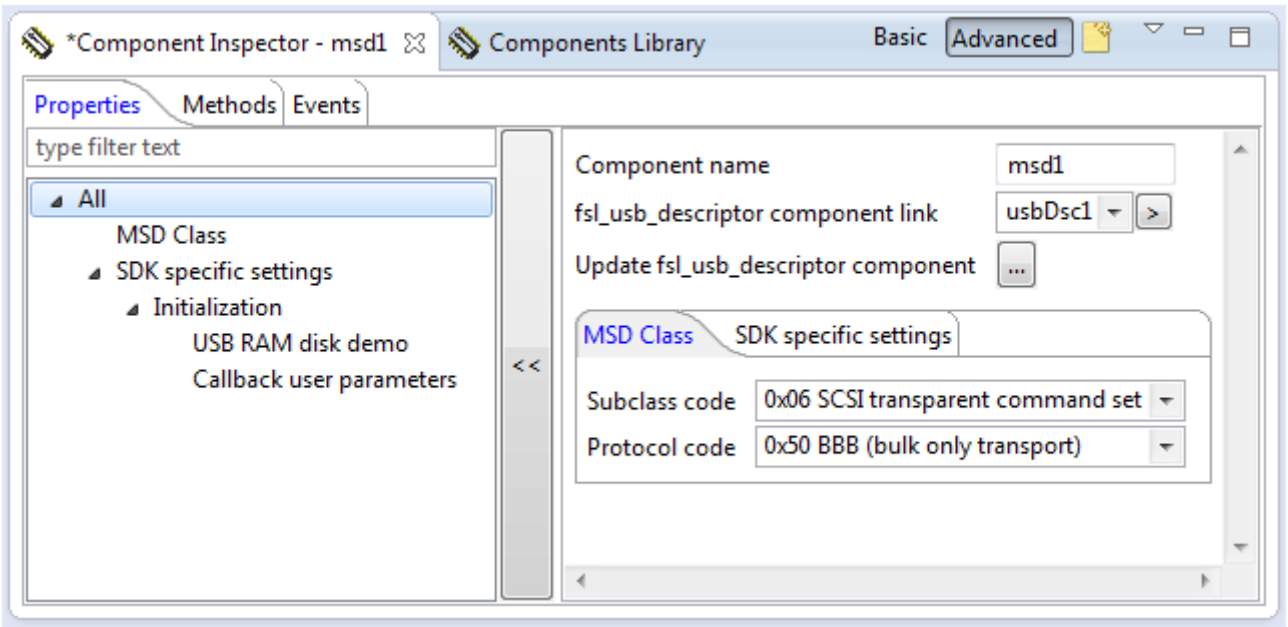


**Figure 2. MSD Subclass and Protocol code configuration**

- Creating MSD USB stack configuration structures and variables.

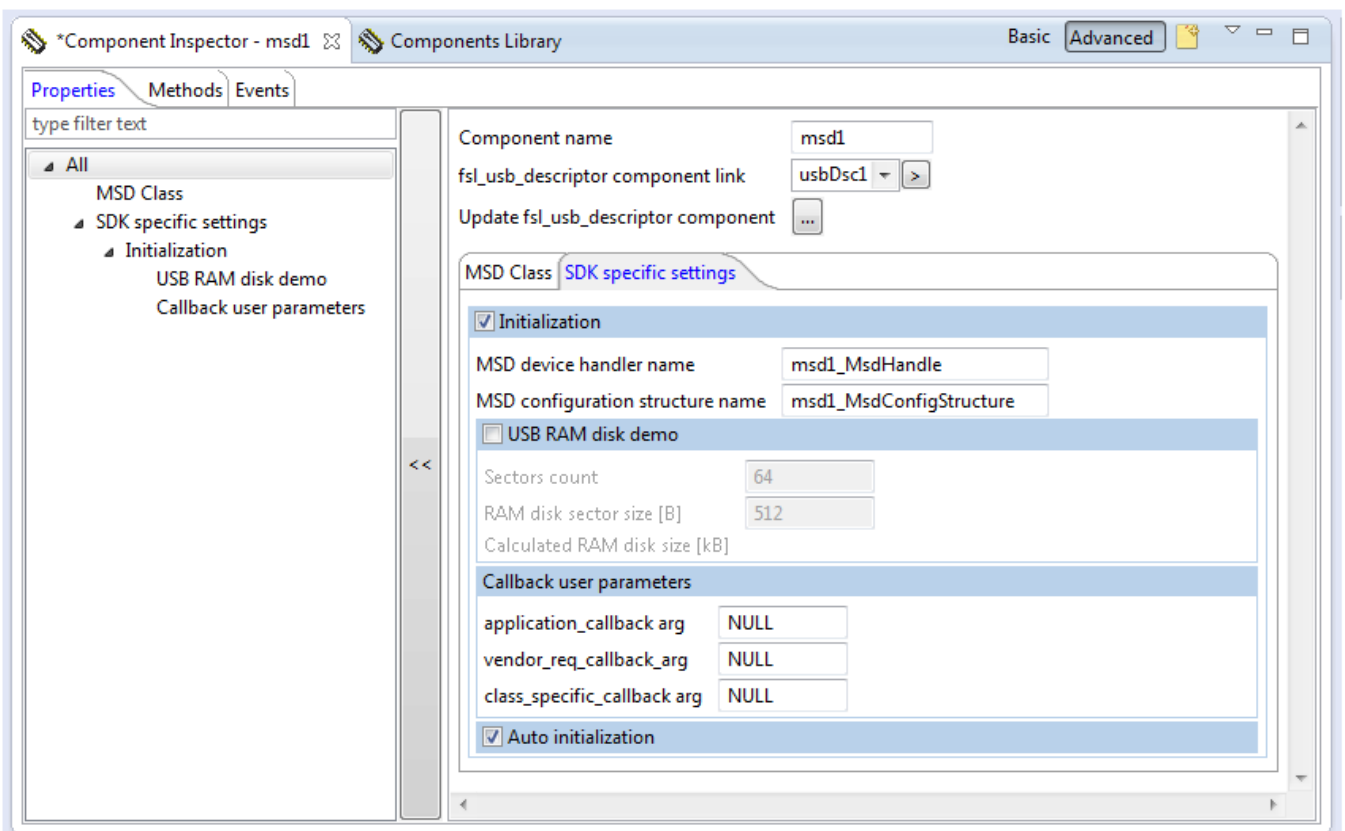


**Figure 3. Properties for MSD class initialization**

- Creating C module with USB MSD callbacks. The default name of C module is *msd1_msd.c* and is stored in Sources folder. This module contains USB RAM disk demo code which is activated by the **USB RAM disk demo** property. You can set the USB RAM disk parameters such as sector size and count in the component properties.

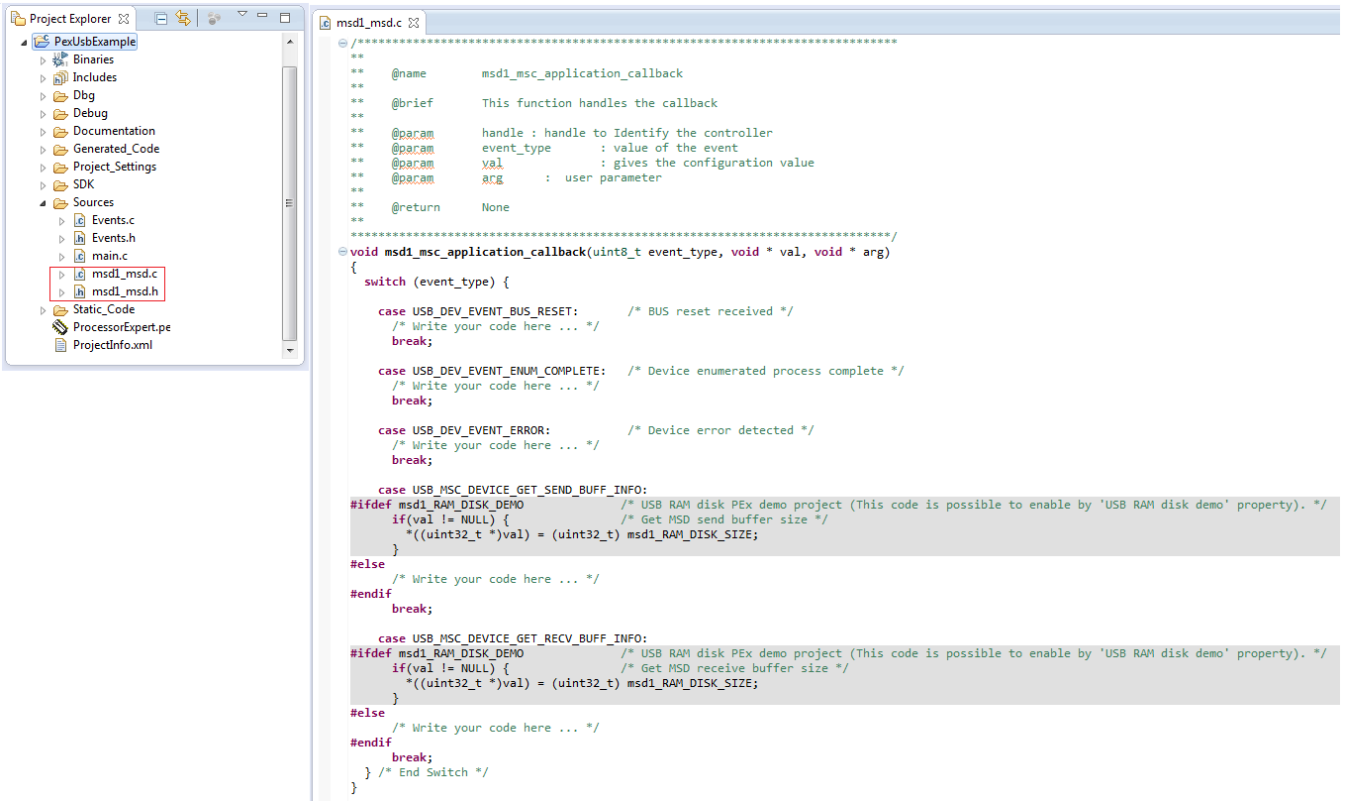


**Figure 4. MSD class callbacks API (with RAM disk demo code) generated by Processor Expert**

- Accessing to the MSD API functions (API functions are taken over by the *usb_class_msc.h* file).

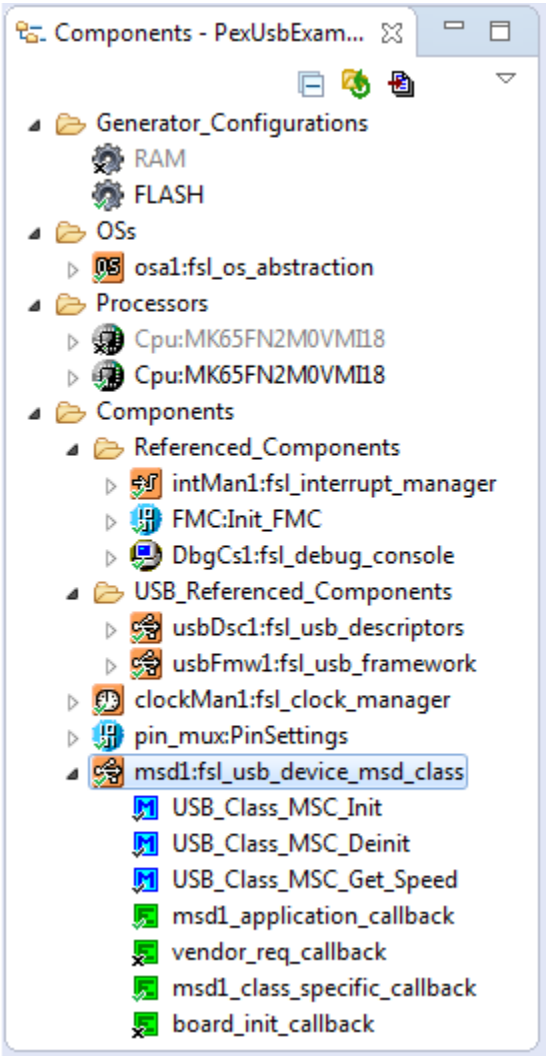


**Figure 5. MSD class API and callback functions**

- Adding MSD class requirements to *fsl_usb_descriptor* component such as class type, desired endpoints, and so on.
- Adding Kinetis SDK USB stack MSD driver files to the project.

## 3.2 fsl_usb_device_hid_class component

The component allows:

- USB device HID. **Subclass** and **Protocol** code configuration.

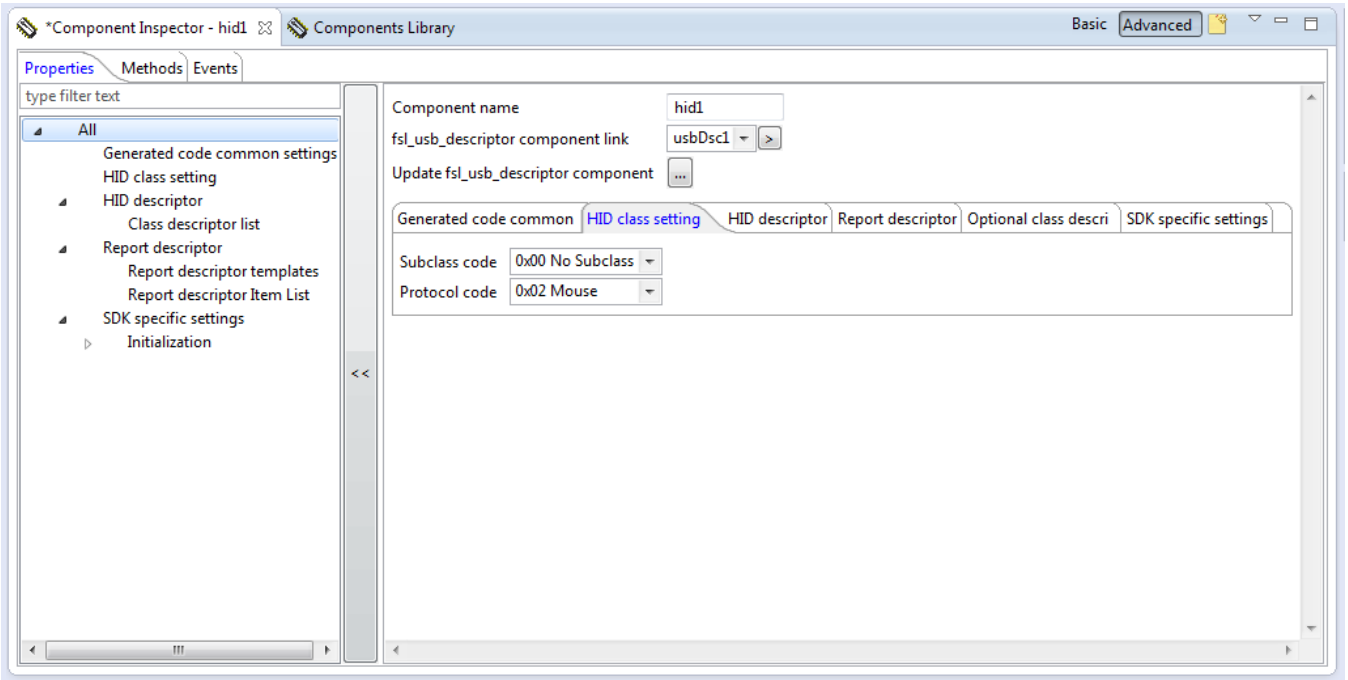


**Figure 6. HID Subclass and Protocol code configuration**

- HID report descriptor definition. *fsl_usb_device_hid_class* component contains 3 predefined report descriptor template: Standard mouse, keyboard, or thermometer.

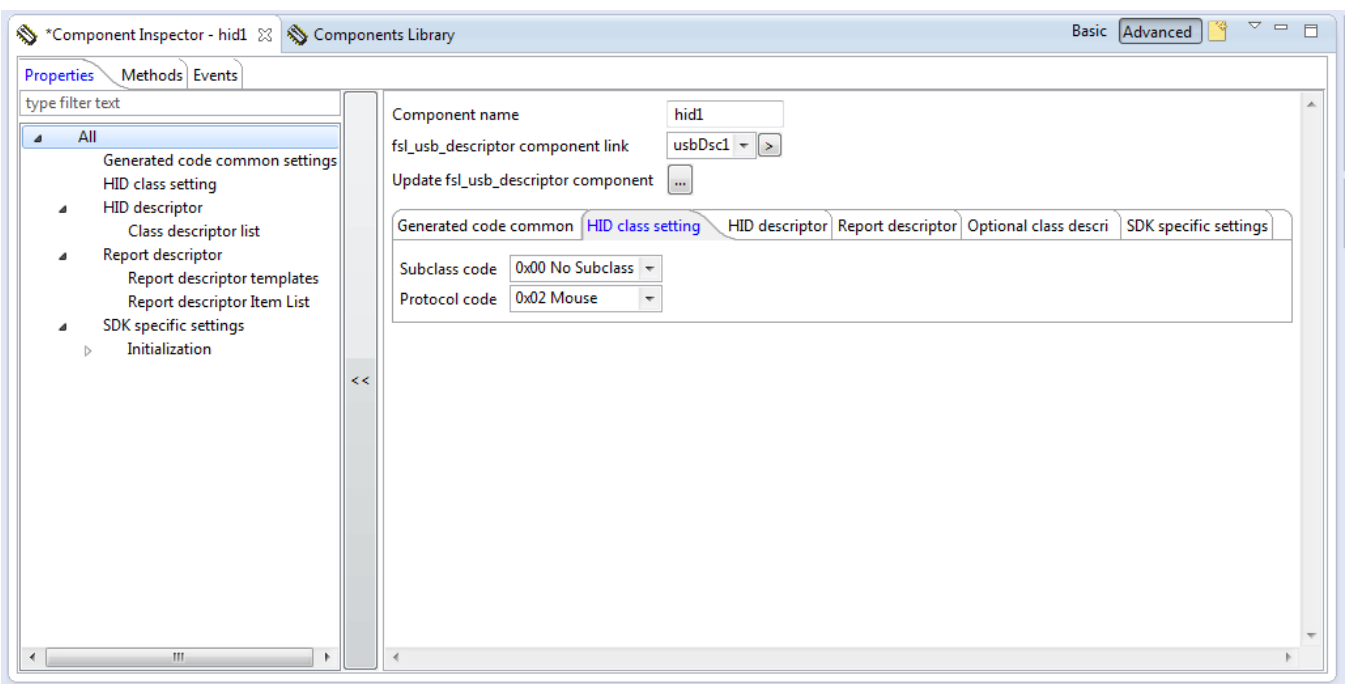


**Figure 7. HID report descriptor definition**

- Creating HID USB stack configuration structures and variables.

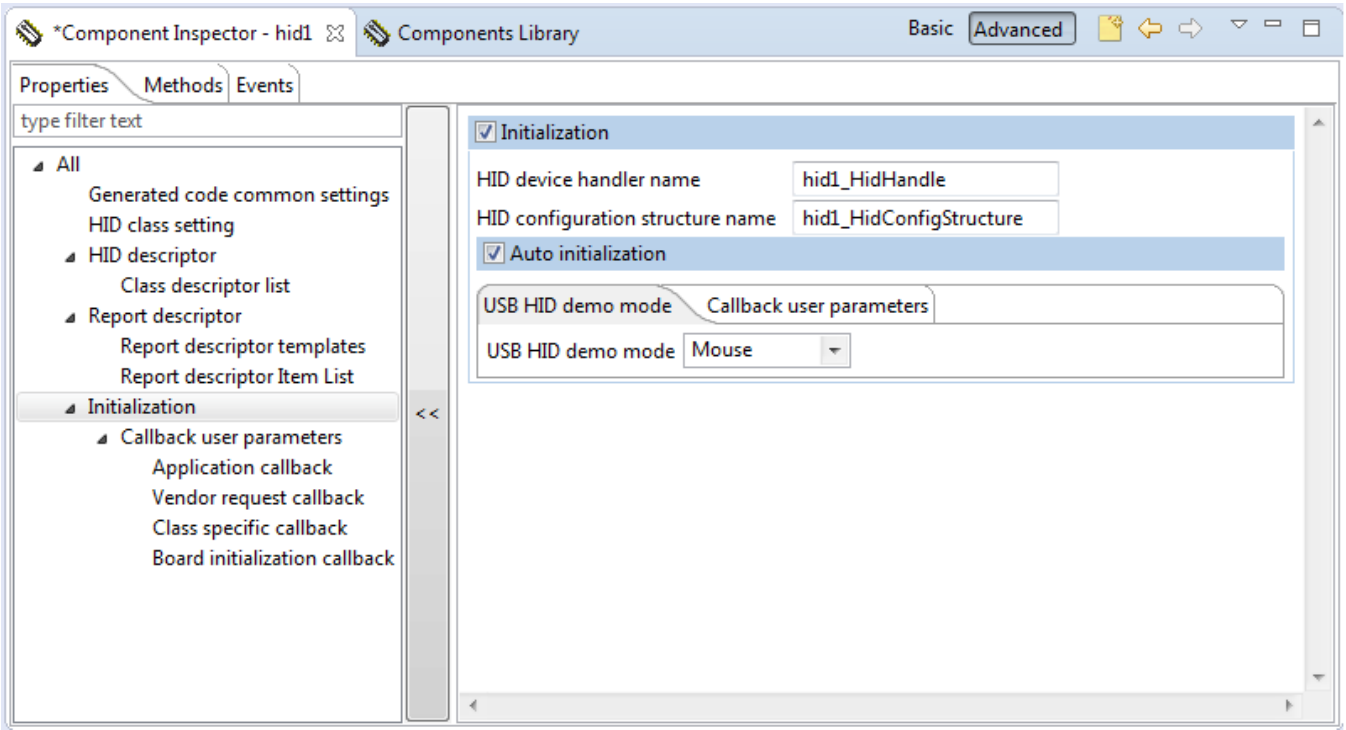


**Figure 8. Properties for HID class initialization**

- Creating C module. The default name of C module is *hid1_hid.c* and is specified by the **Event module name** property in the **Events** tab. The module is stored in the Sources folder. This module contains USB HID callbacks and USB HID demo code, which can be actovated by the **USB HID demo mode** property. The USB HID component contains **Mouse,Keyboard**, and **Thermometer** demos type. Selecting a particular demo type (mouse/keyboard/thermometer) automatically changes the HID report descriptor and the HID protocol code (mouse/keyboard).

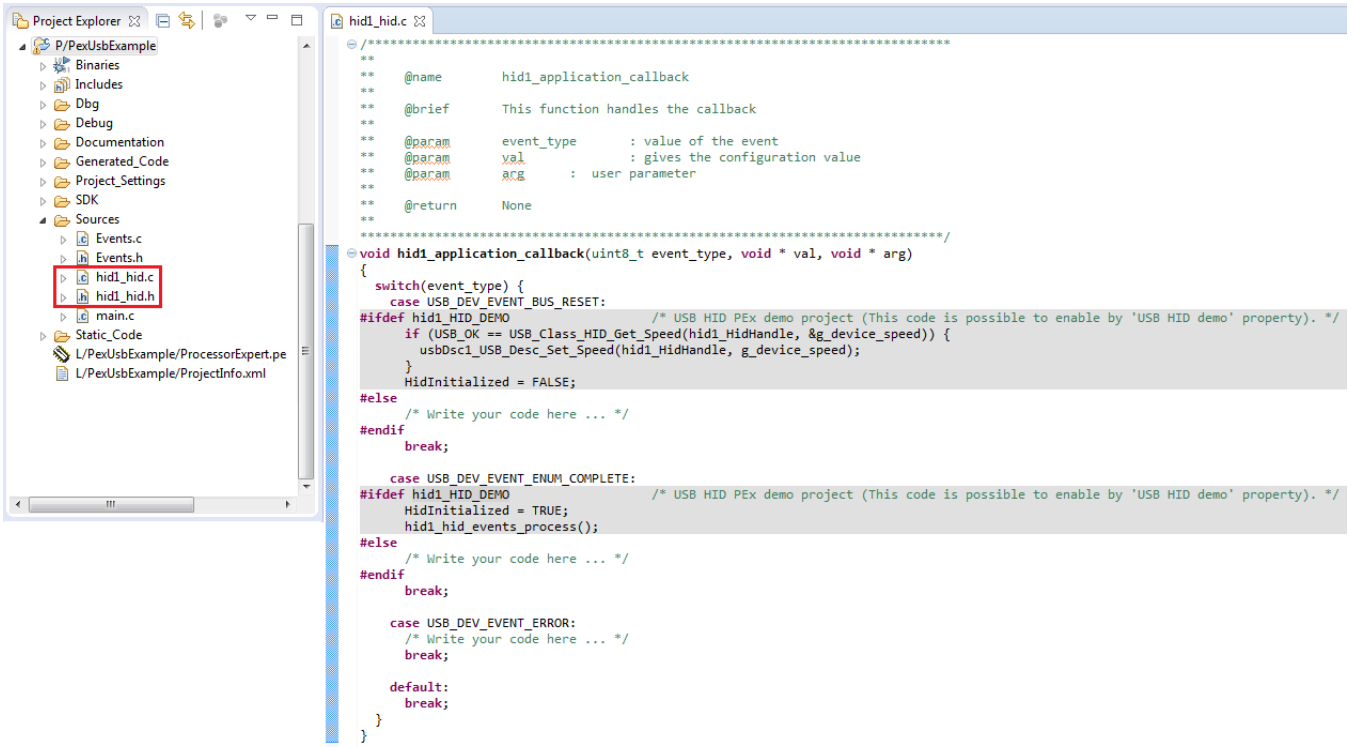


**Figure 9. HID class callbacks API (with HID demo code) generated by Processor Expert**

- Accessing the HID class API functions (API functions are taken over *usb_class_hid.h* file).

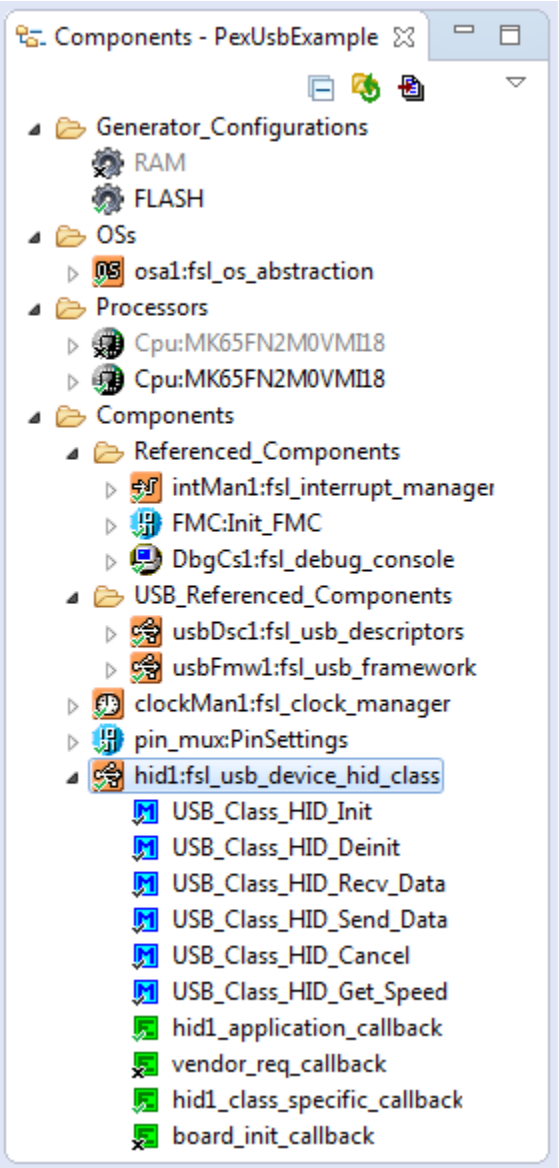


**Figure 10. HID class API and callback functions**

- Adding HID class requirements to *fsl_usb_descriptor* component such as class type, desired endpoint, and so on. *fsl_usb_device_hid_class* component requires only one input-endpoint (inputput-endpoint by default). If application requires receive data from HOST device (output-endpoint), you must manually add the output-endpoint in *fsl_usb_descriptors* component.
- Adding Kinetis SDK USB stack HID driver files to the project.

## 3.3 fsl_usb_descriptors component

*fsl_usb_descriptors* component is used for DEVICE USB mode.

The basic function of the *fsl_usb_descriptors* component is to collect requirements. For example, USB device class type, pipes numbers and type of all *fsl_usb_device_XXX_class* components, define enpoint sizes and configure using SDK USB class drivers. On the basis of the collected information, *fsl_usb_descriptors* component creates the USB standard and class descriptors structures and creates the standard USB function. For example, GetDescriptors, SetInterface, and SetConfiguration.

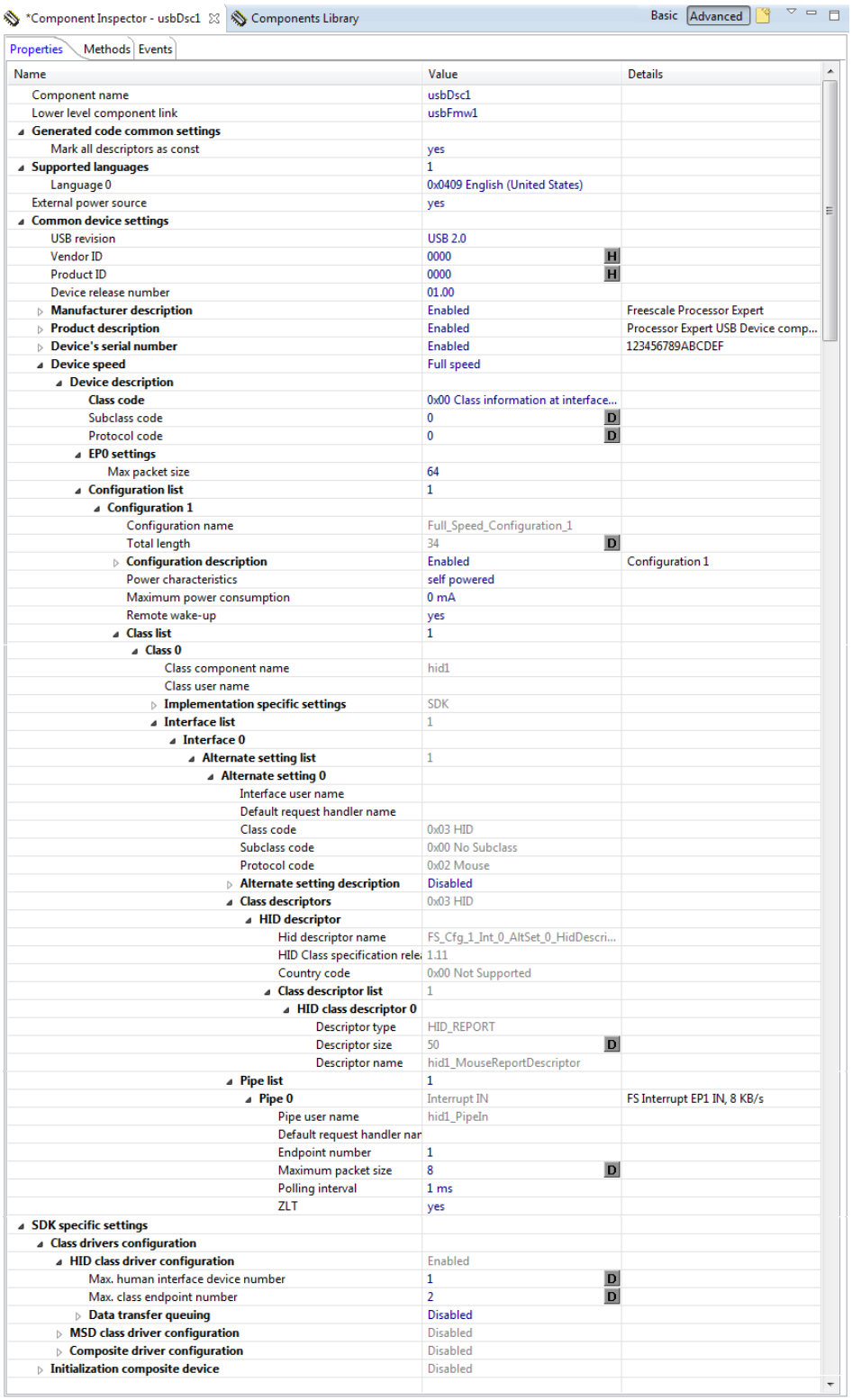


**Figure 11. fsl_usb_descriptors component**

Component allows:

- Creating C module. The default name of C module is *usbDsc1.c,* which is stored in the Generated_Code folder. The *usbDsc1.c* file contains description of all USB device class component such as Device/Configuration/Strings descriptors, structures which describe used endpoints and class types, and standard USB device function like GetDescriptors, SetInterface, SetConfiguration, and so on.

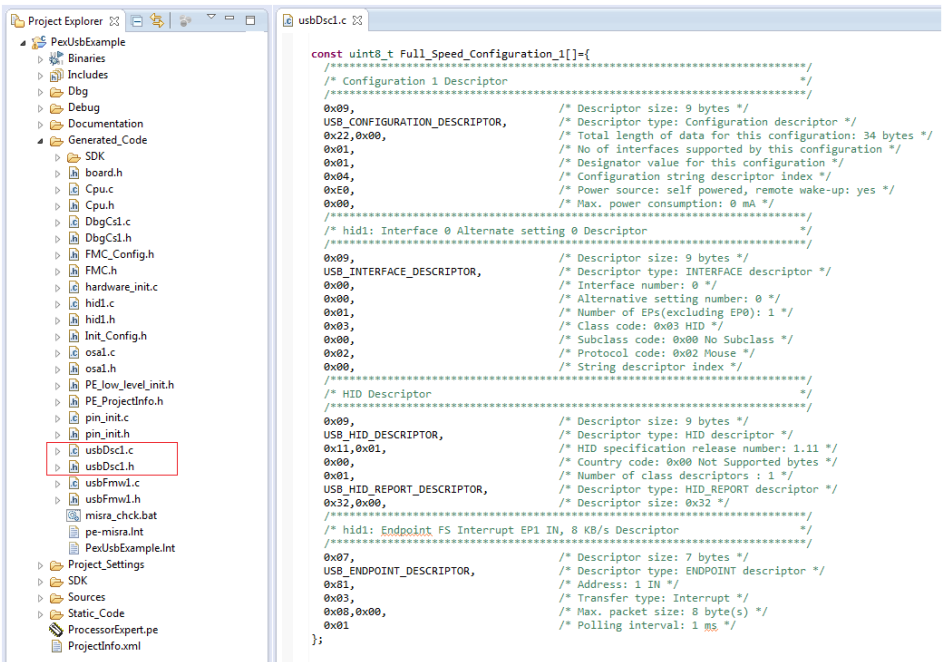


**Figure 12. Content of usbDsc1.c file, description of USB device**

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

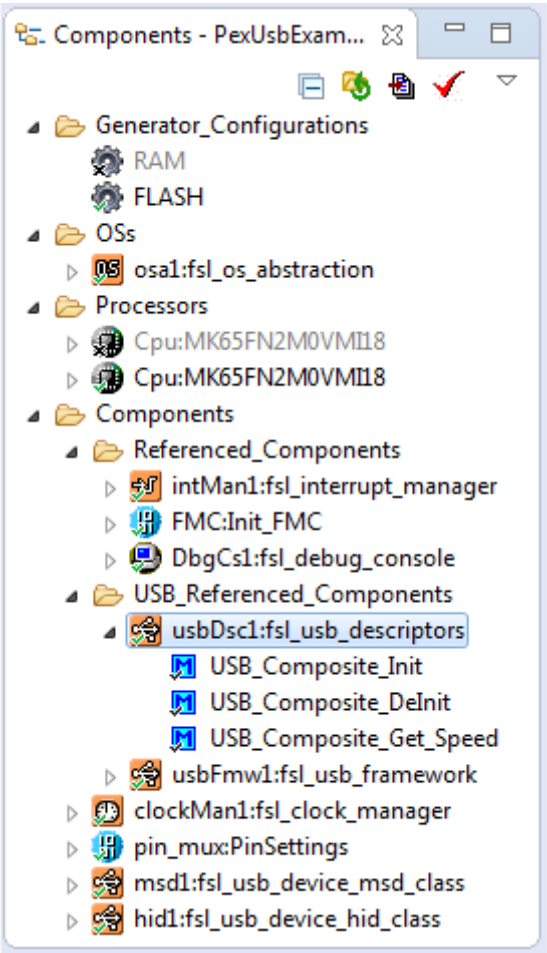


**Figure 13. Composite class API functions**

- Adding requirements to *fsl_usb_framework* component such as USB mode endpoint type, and count.
- Adding Kinetis SDK USB stack "Composite" driver files to the project (for USB composite mode), create kinetis SDK class configuration files.

## 3.4 fsl_usb_framework component

*fsl_usb_framework* component covers bottom layer of the USB interface that transmits and receives packets. *fsl_usb_framework* component USB low/full speed (KHCI) and USB low/full/high speed (EHCI) module.

*fsl_usb_framework* component supports three modes: DEVICE, HOST and OTG (USB OTG mode is not fully supported yet, OTG mode allows currently use DEVICE or HOST mode. HOST or DEVICE mode can be selected in runtime, according USB ID pin signal value, by usb_device/host_init() methods. According the selected USB mode, *fsl_usb_framework* component adds SDK USB stack source files to the project and generates USB stack configuration and BSP files. BSP files contains code for USB module timing, interrupt settings and PHY configuration.

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

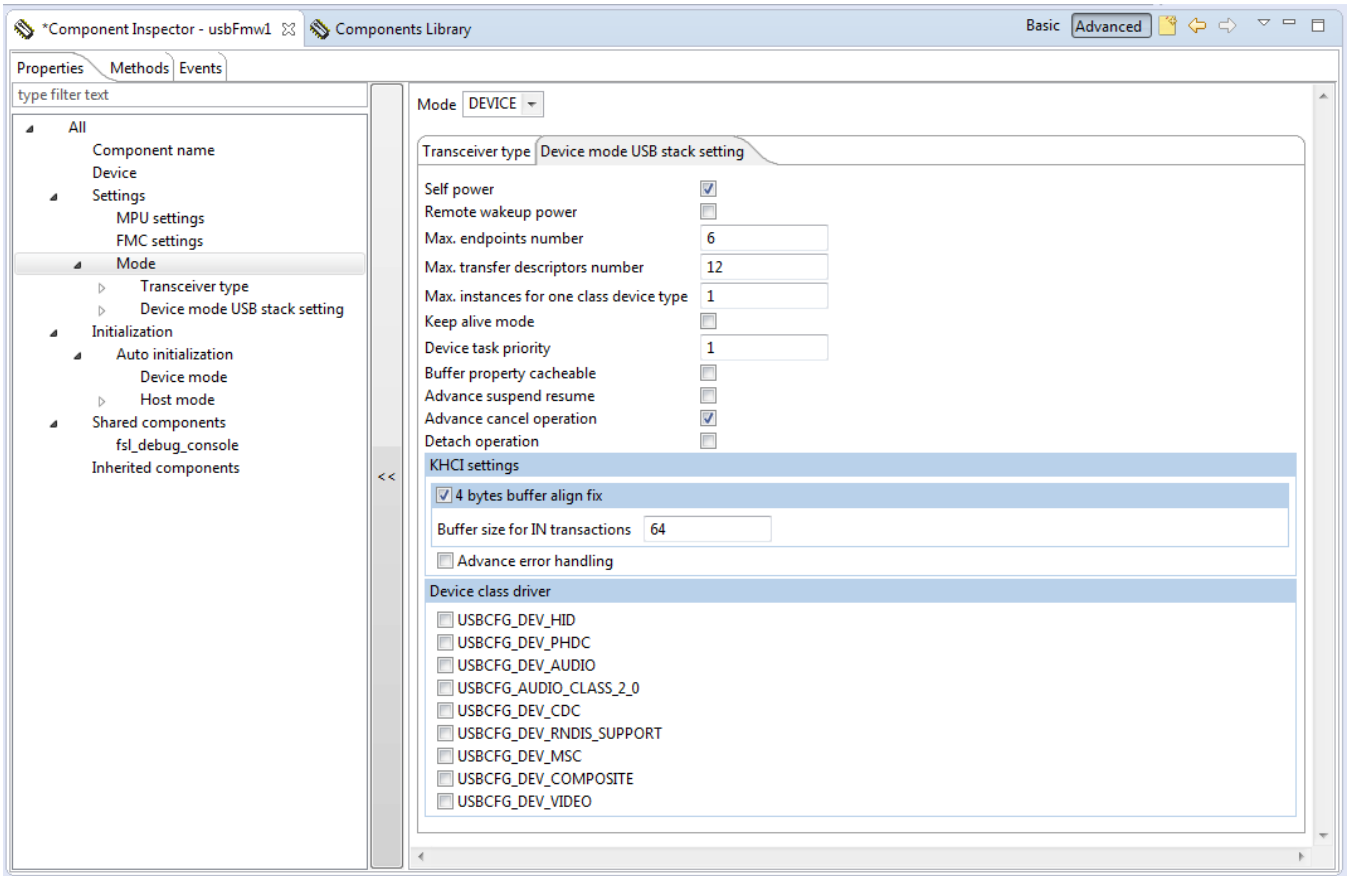


**Figure 14. USB DEVICE mode configuration**

- Creating configuration structures and variables for USB DEVICE or HOST mode for standalone component use.

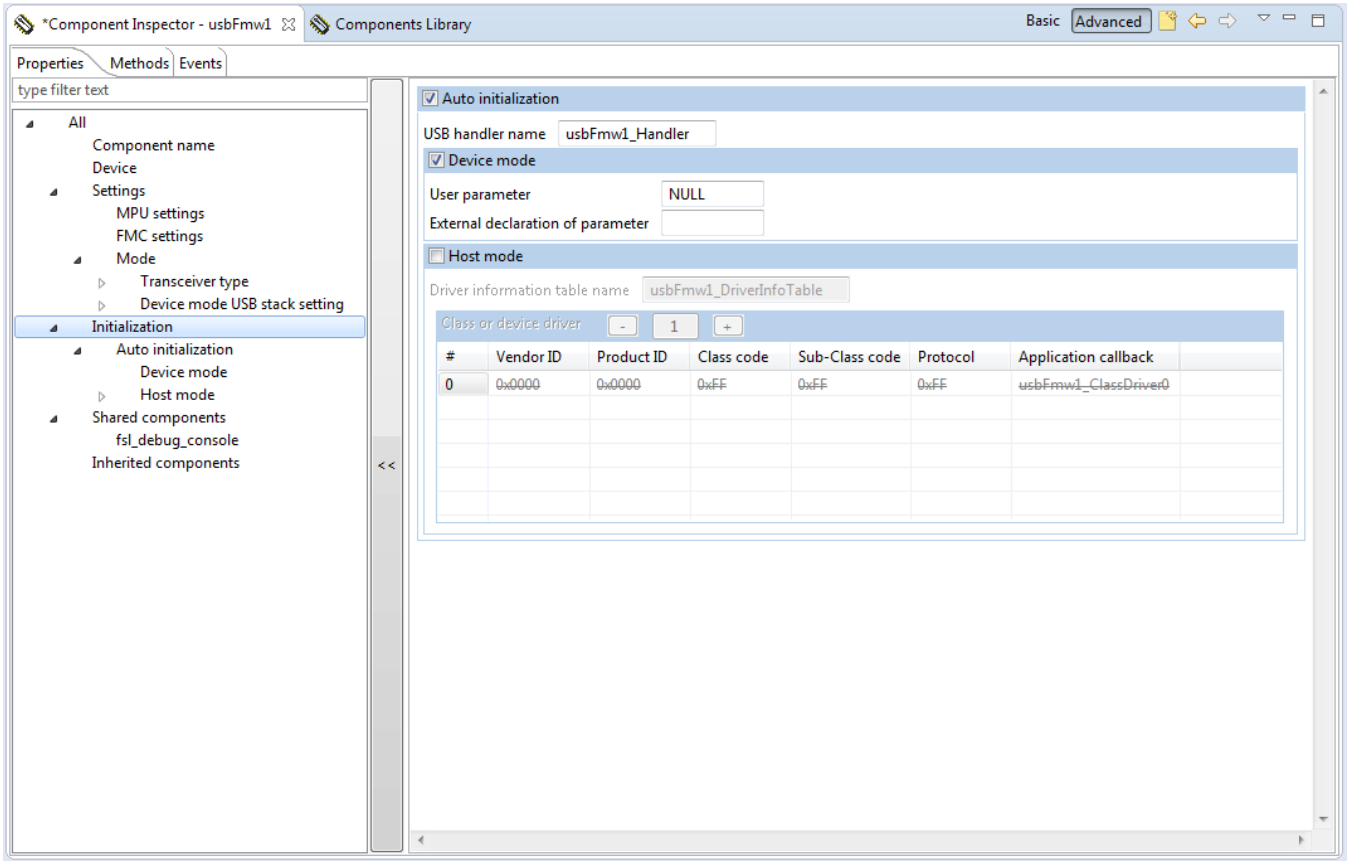


**Figure 15. Properties for device mode initialization**

- Accessing 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)

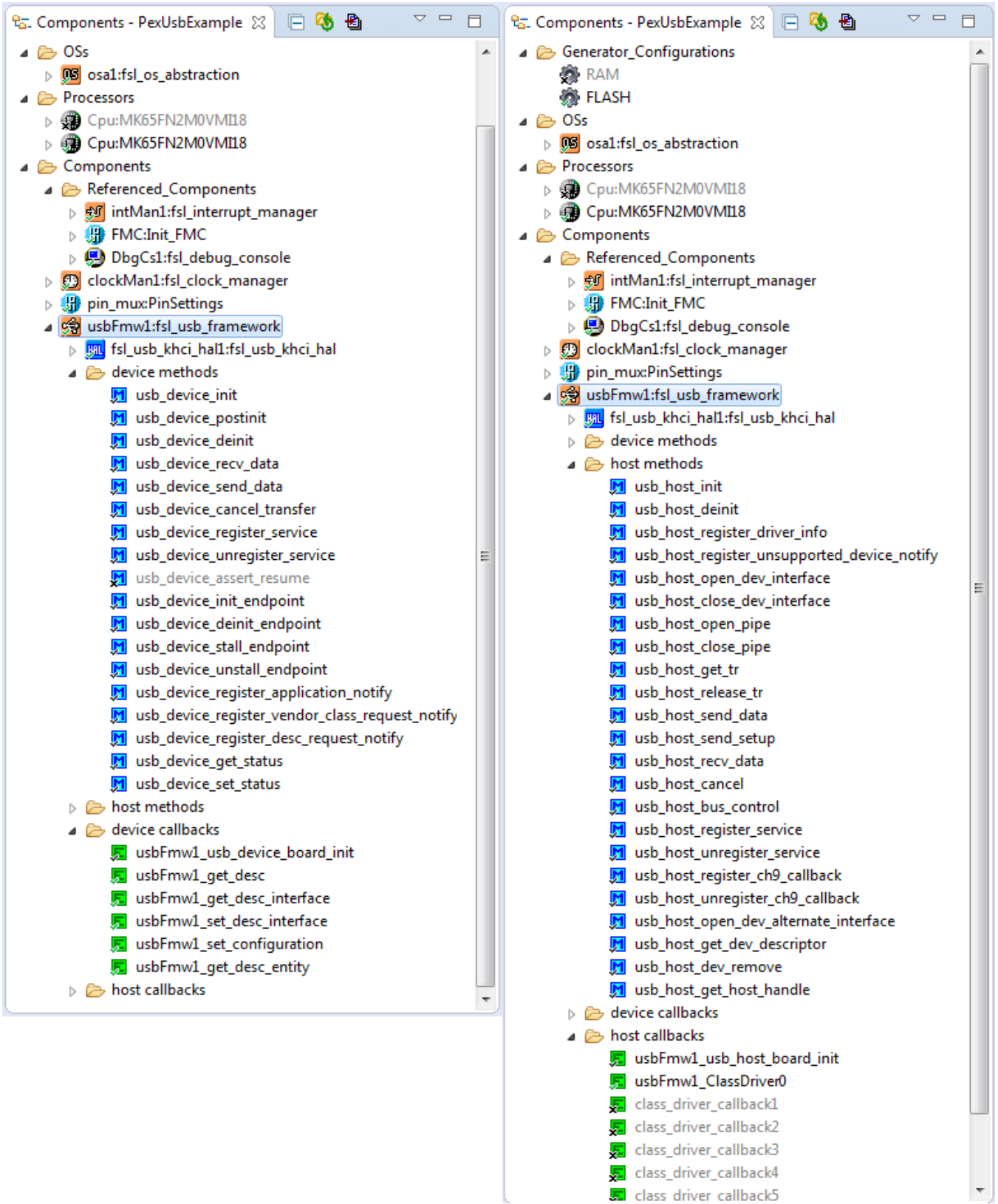


**Figure 16. USB DEVICE / HOST mode API and callback functions**

- MPU (Memory Protect Unit) and FMC (Flash Memory Controller) configuration (if MCU contains these devices). USB OTG controller (BUS master) can also access Flash memory (BUS slave) is through "crossbar switch“ (see RM for selected CPU). The crossbar switch connects bus masters and bus slaves using a crossbar switch structure.

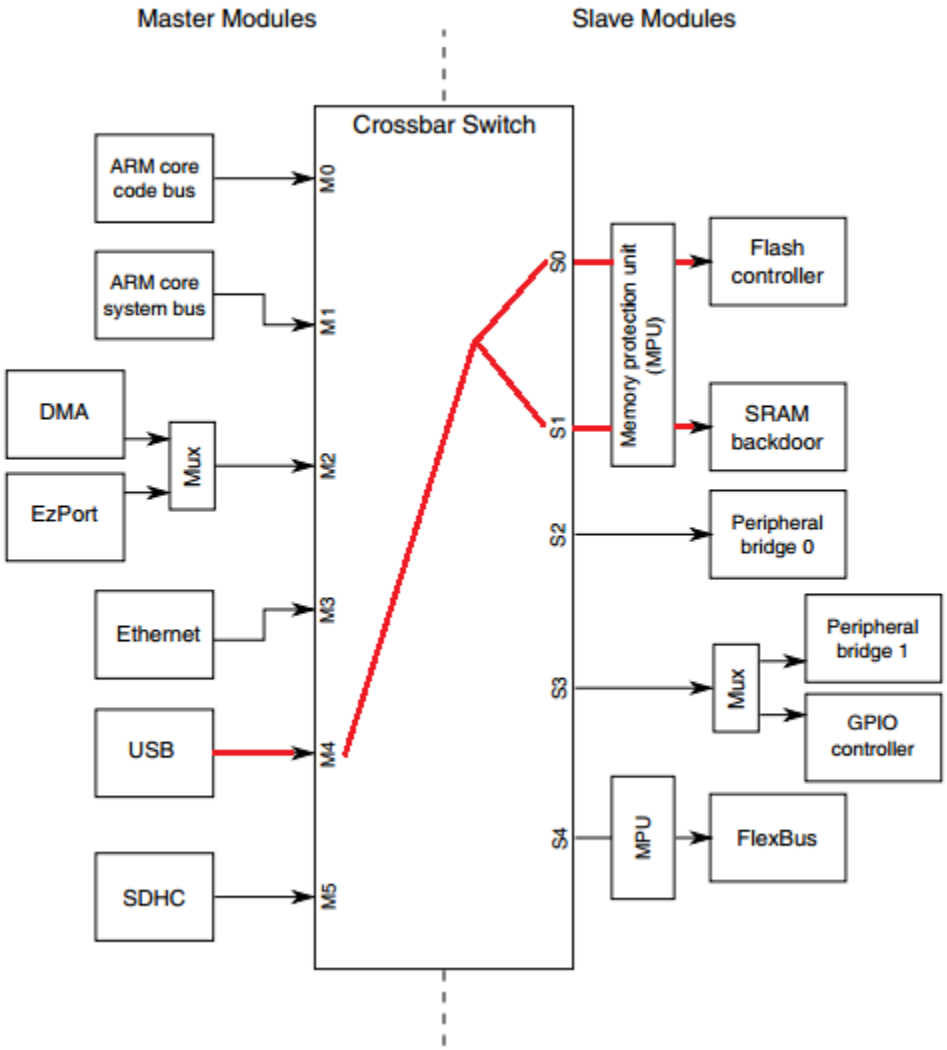


**Figure 17. Crossbar switch integration**

USB controller can access to variables which can are stored in the Flash memory. or example, the USB device/configuration descriptors. By default, the USB controller has no access to flash memory. Therefore, access must be permitted in Flash Memory Controller (FMC) and Memory Protect Unit (MPU):

- If the **FMC settings** option is enabled, *fsl_usb_framework* component links and configures the *Init_FMC* component for USB module Flash memory read access.
- If the **MPU settings** option is enabled, PEx generates MPU configuration code of the MPU module to the *usb_dev/host_bsp.c* module. Enable/disable state of MPU module is driven by the *MPU module* property.

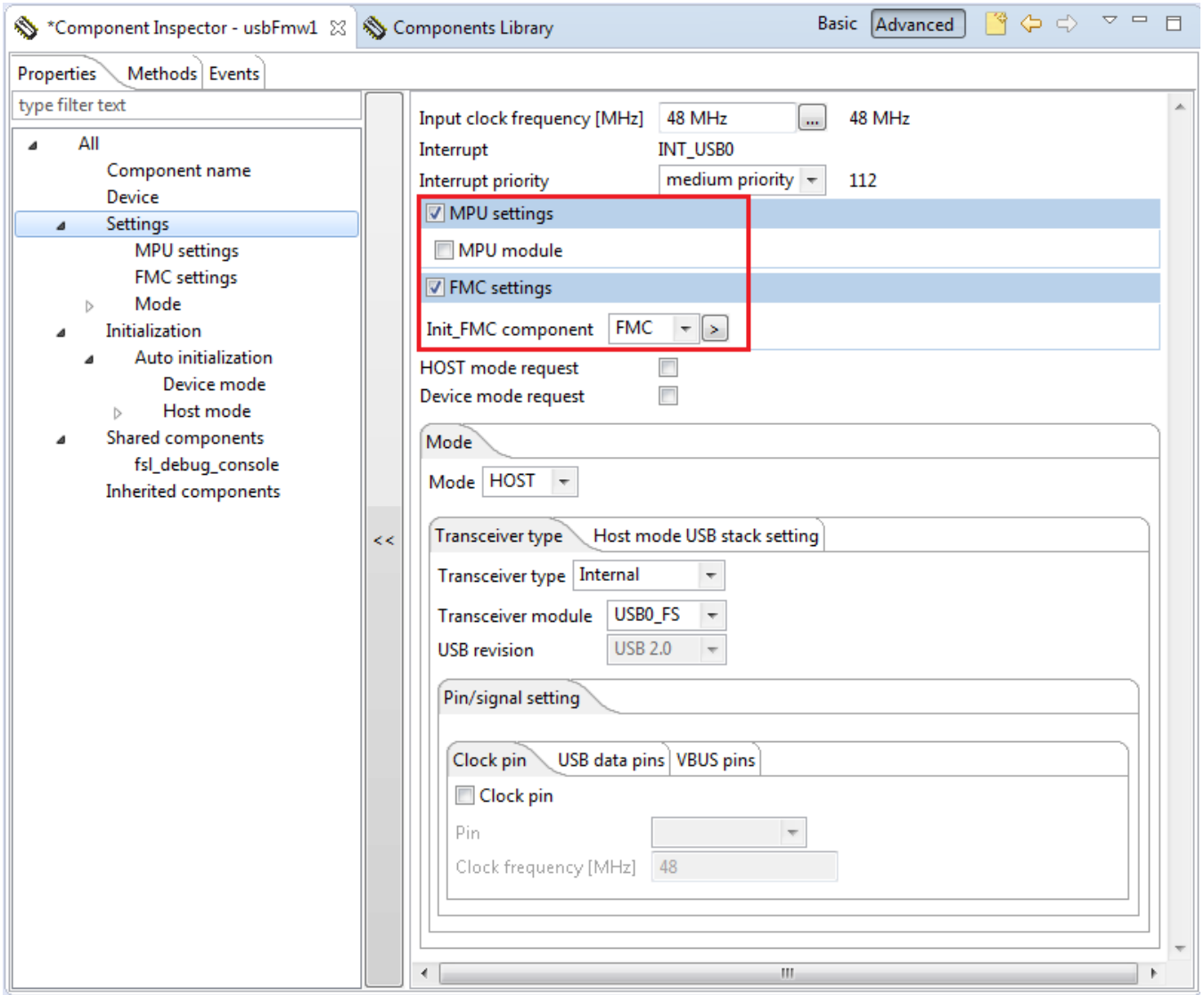


**Figure 18. Properties for device mode initialization**

## 3.5 fsl_usb_ehci_hal component

Used for high speed USB module, provides basic I/O macros for access to the USB high speed OTG controller (USB periphery read/write register access operation).

## 3.6 fsl_usb_khci_hal component

Used for full speed USB module, provides basic I/O macros for acces to the USB full speed OTG periphery (USB periphery read/write register access operation).

# 4 Creating common PEx USB project

Processor Expert new project wizard helps you to create project based on the selected processor or board type.

- **Boards** project type: CPU and fsl_clock_manager components are configured according selected board, such as Pin signals, XTAL, and MCU timing. In this project type, fsl_clock_manager component contains predefined clock configurations. The **USB clock setup** clock configuration is intended for a USB stack project.
- **Processor** project type: CPU and fsl_clock_manager components are configured to default (after reset) values. In this project type, you must configure the fsl_clock_manager component. The configuration includes properties such as System oscilator-XTAL, MCG mode, and MCU timing.

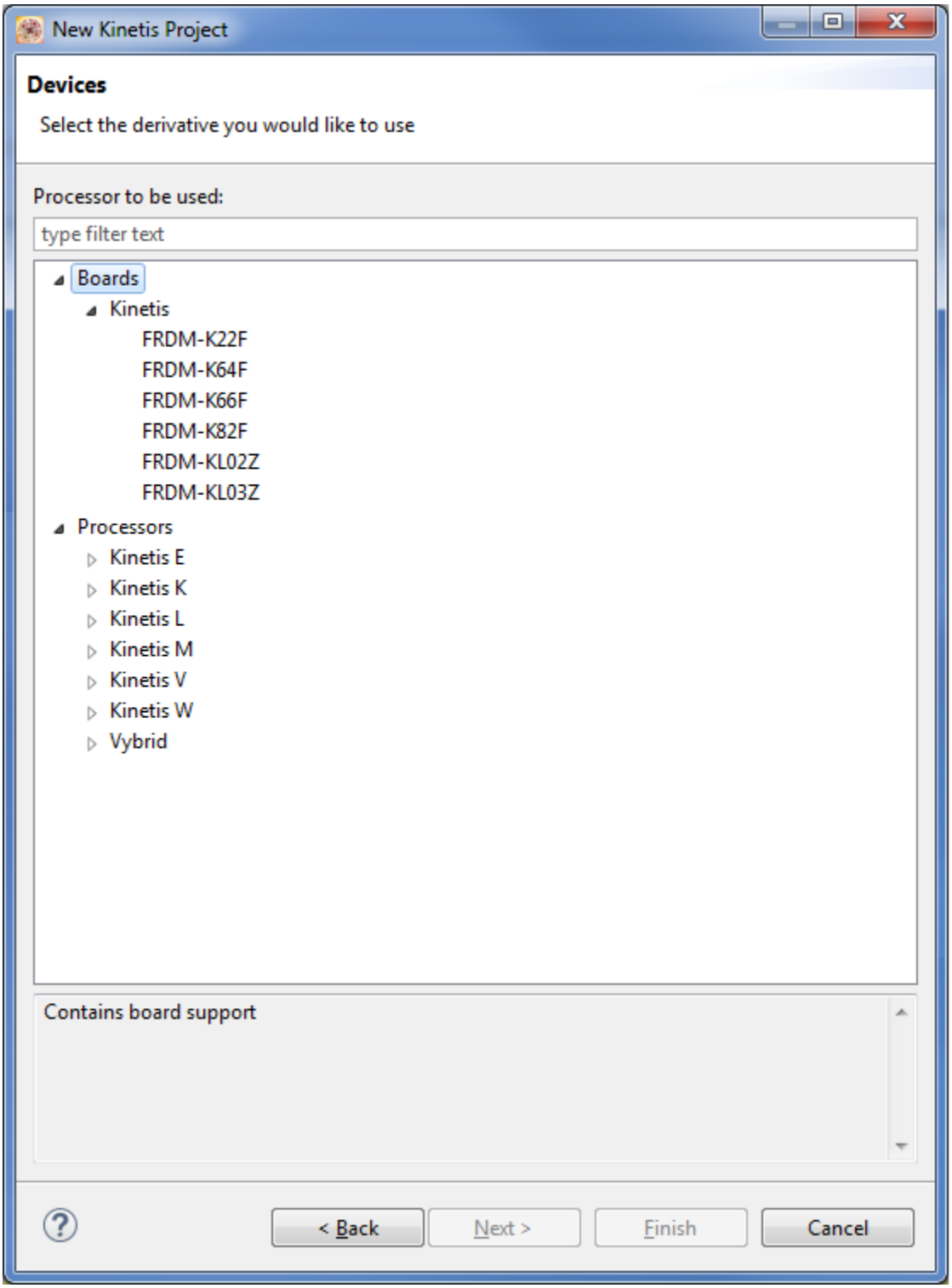


**Figure 19. Processor Expert new project wizard**

Project configuration for “Processor” project type:

1. Create Processor Expert SDK project with selected “Processor” - MK65F180M.

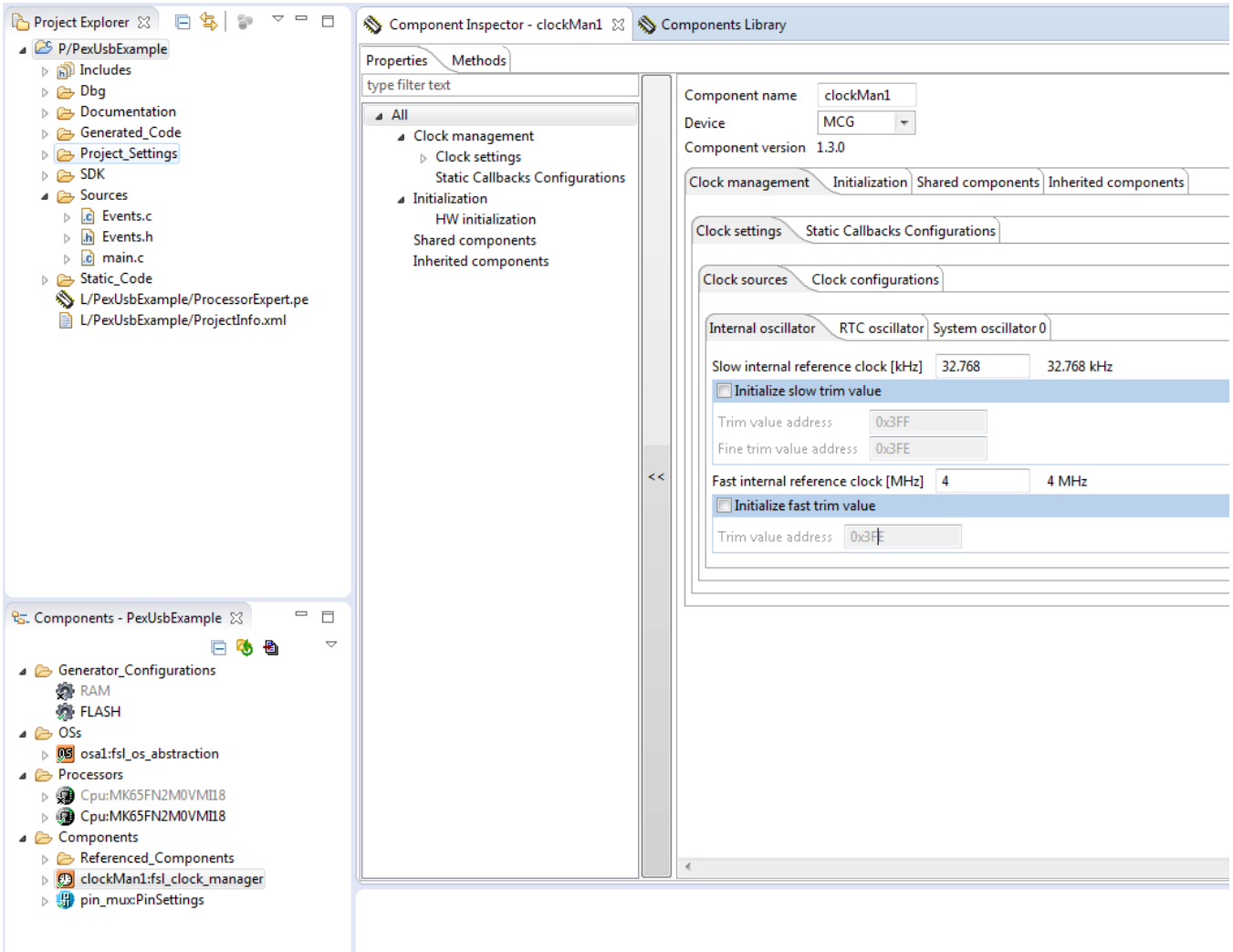


**Figure 20. Created PEx project - default settings**

2. USB full speed (FS) OTG controller requires 48MHz input clock source which depends on fsl_clock_manager component setting. For case if USB module is cloked from internal MCU clock source. The Default (after MCU 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.
3. In fsl_clock_manager component enables the **System oscilator 0** property. Select the **Clock source = External reference clock**Set „Clock frequency [MHz] to 16 (TWR-MK65F180M board contains 16MHz external crystal rezonator – see scheme for TWR- MK65F180M board.

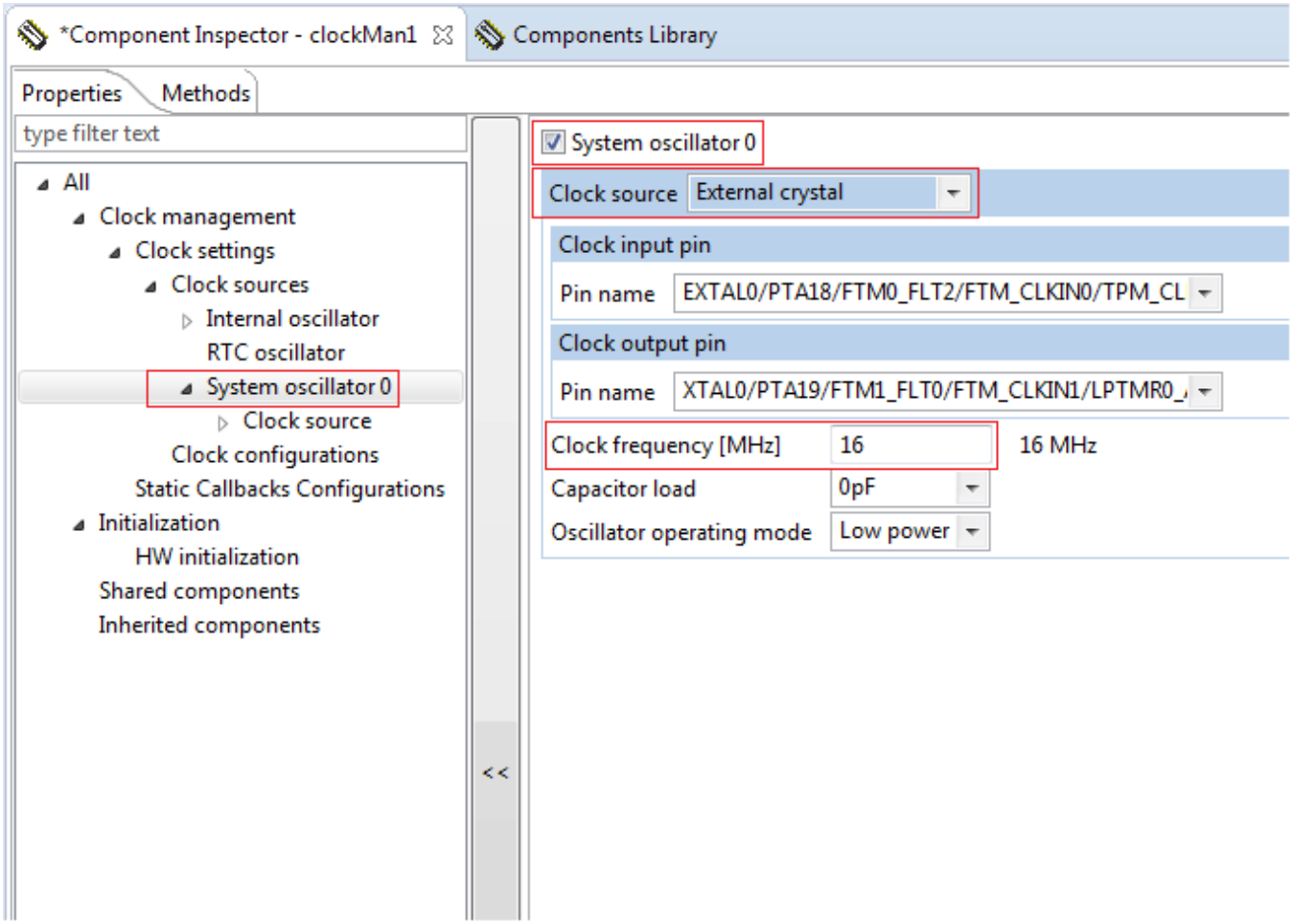


**Figure 21. "System oscillator 0" configuration**

4. In the **Clock configuration** menu click on the **MCG settings** tab and the **MCG mode** property. Select the **MCG mode** as *PEE* and set the **PLL output clock [MHz]** to 120.

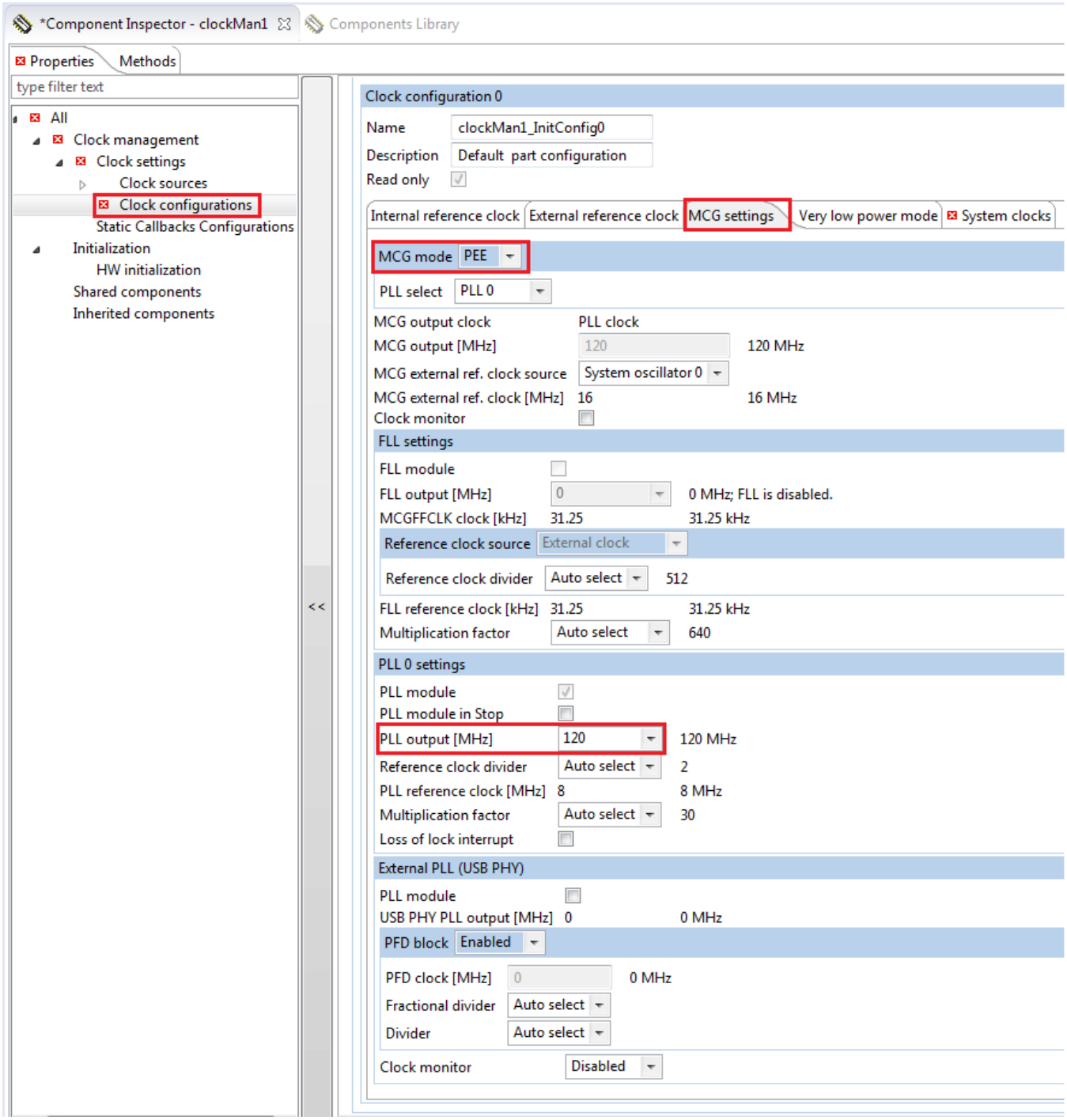


**Figure 22. MCG module configuration**

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

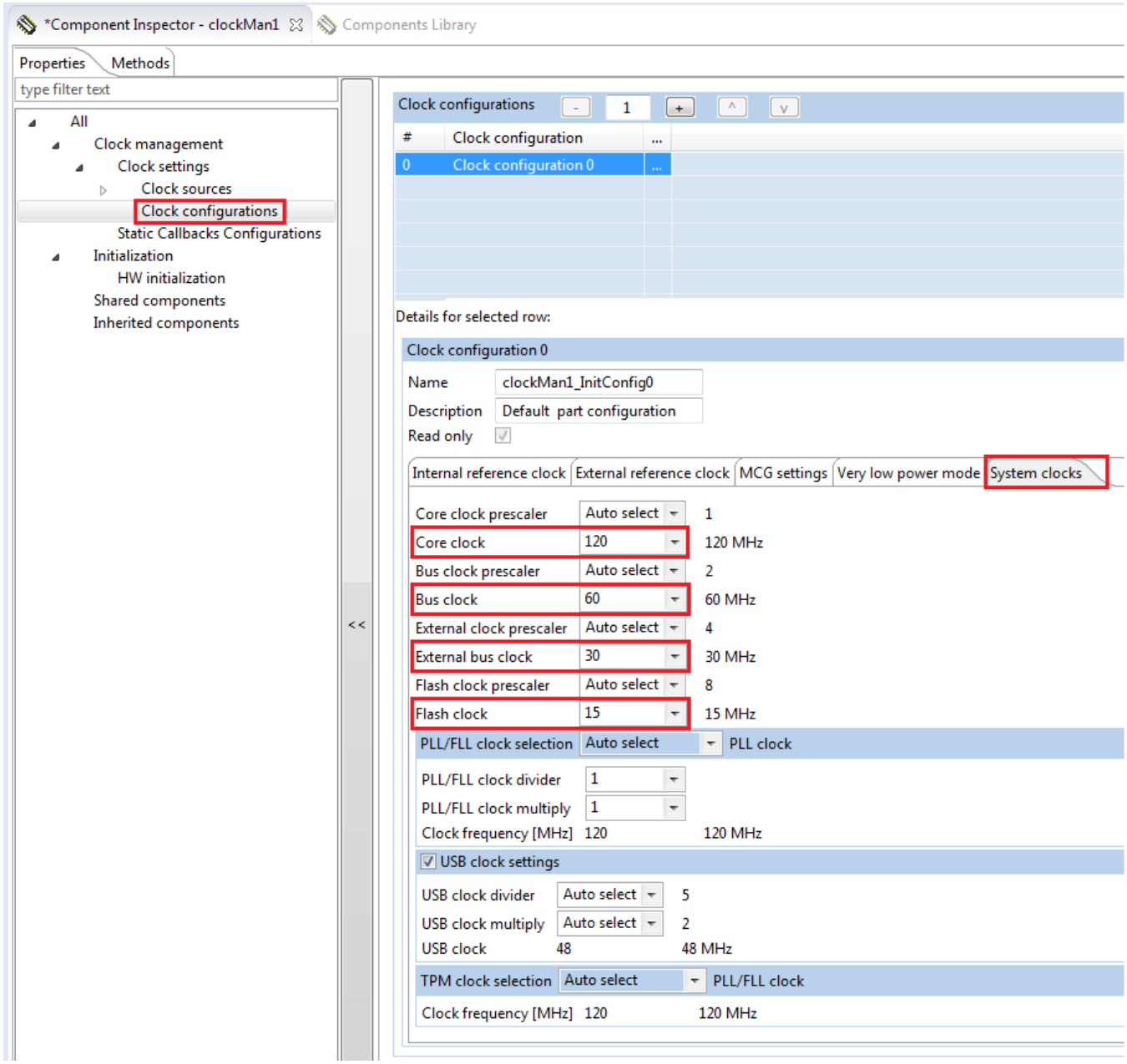


**Figure 23. CPU system clocks configuration**

6. USB high speed (HS) OTG controller requires 480MHz (clocked from External PLL USB PHY) input clock source which depends on fsl_clock_manager component setting. External PLL USB PHY is enabled by the **PLL module** property.

**NOTE**

For External PLL USB PHY = 480MHz must have the System oscillator 0 enabled, see the figure above.

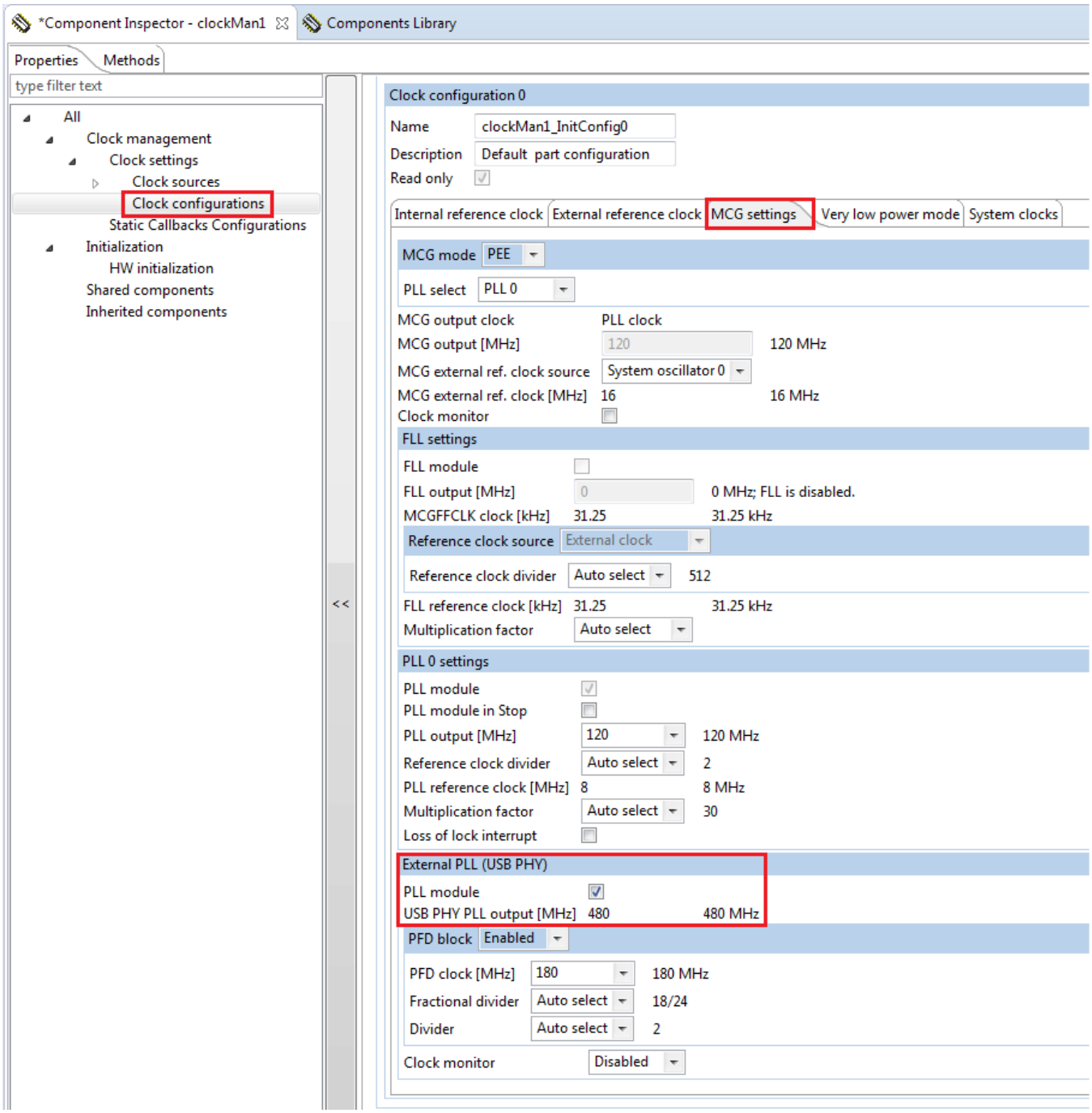


**Figure 24. External PLL USB PHY" configuration**

7. USB HS/FS/LS transceiver module requires voltage source (3.3V) provided by the USB regulator output. Input of the USB regulator is connected to the **Vregin** pin on the MCU package. This PIN can be connected to USB_VBUS line (USB transceiver is powered from USB host device, for example: PC, USB device BUS power mode) or to 5V board power supply (USB device self power mode).

**Figure 25. USB regulator, BUS power use case**

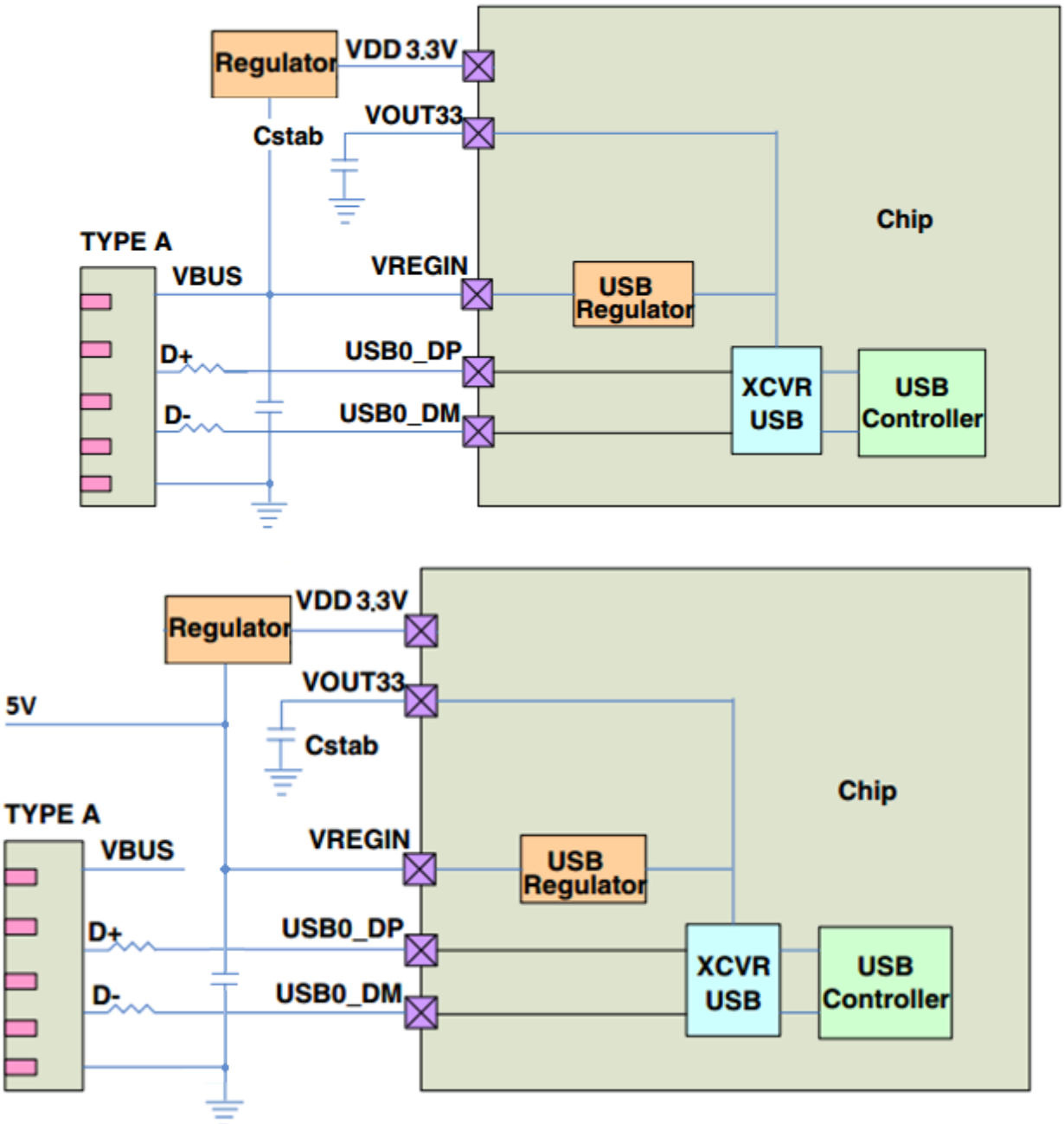


**Figure 26. USB self power use case**

**TWR-K65F180M_Rev B**: J31 selector on TWR- MK65F180M board specifies which USB module is connected J15 connector (USB micro):

- 1-2: J15 goes to USB HS and Elevator goes to USB FS
- 2-3: J15 goes to USB FS and Elevator goes to USB HS

**TWR-K65F180M_Rev C**:

- USBHS – connected to J15 connector (USB micro on TWR-K65F128 board)
- USBFS – connected to elevator (USB connector on TWR-SER board).

# 4.1 Creating PEx USB project

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

For better demonstration and function, the *fsl_usb_device_hid/msd_class* components contain properties for activating USB demo project. In this mode the class component is configured according to the selected demo mode type and the code in the "user" file is activated.

## 4.1.1 USB mass storage project

1. For USB mass storage project add the *fsl_usb_device_msd_class* component from **Components Library** window to the created project (see chapter Creating common PEx USB Project)

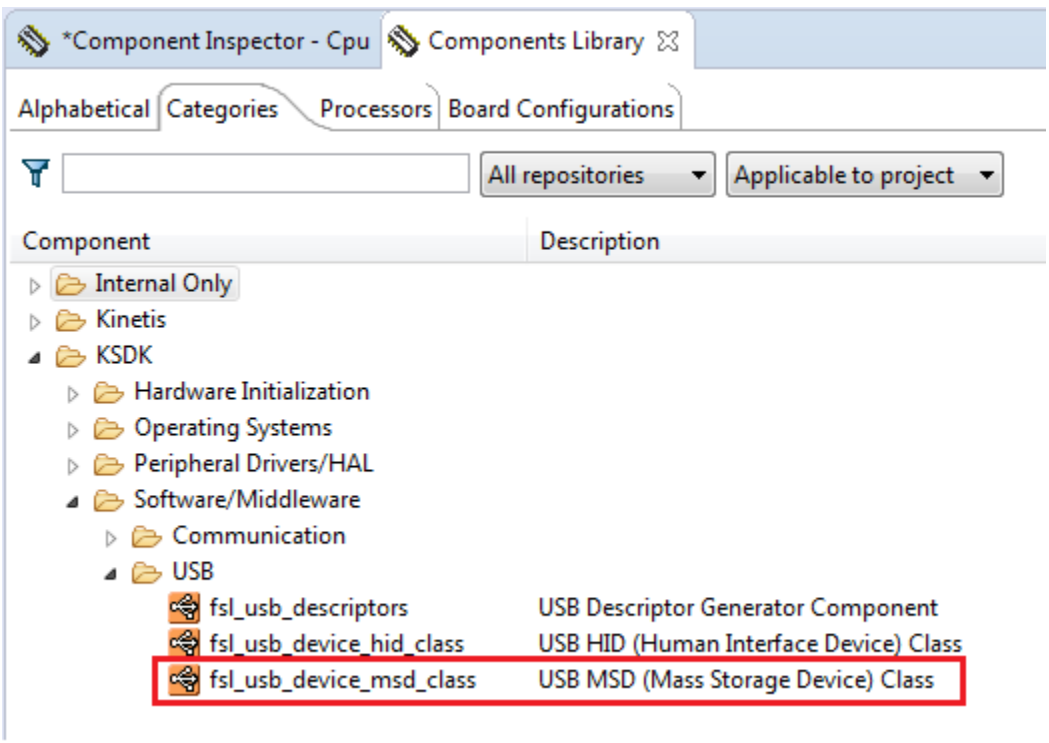


**Figure 27. USB components in "Component Library"**

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 others SDK drivers (fsl_clock and interrupt_manager, etc). All these components are automatically added to the project.

2. Configure "Subclass code" and "Protocol code" on "MSD Class" tab. Only SCSI transparent command and BBB (bulk only transport) are currently supported.

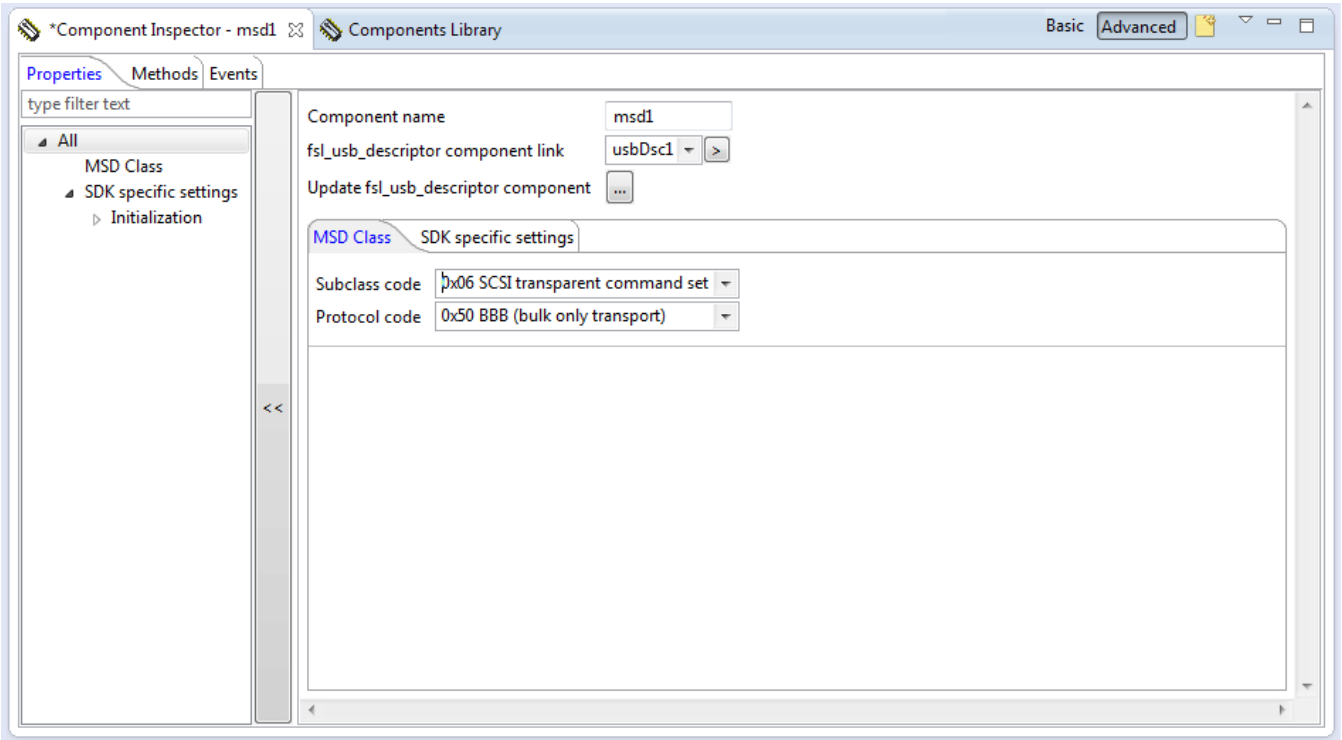


**Figure 28. MSD Subclass and Protocol code configuration**

3. On “SDK specific settings” tab:
   a. Select “Initialization” and “Auto initialization” checkboxes (They should be selected by default)
   b. 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).

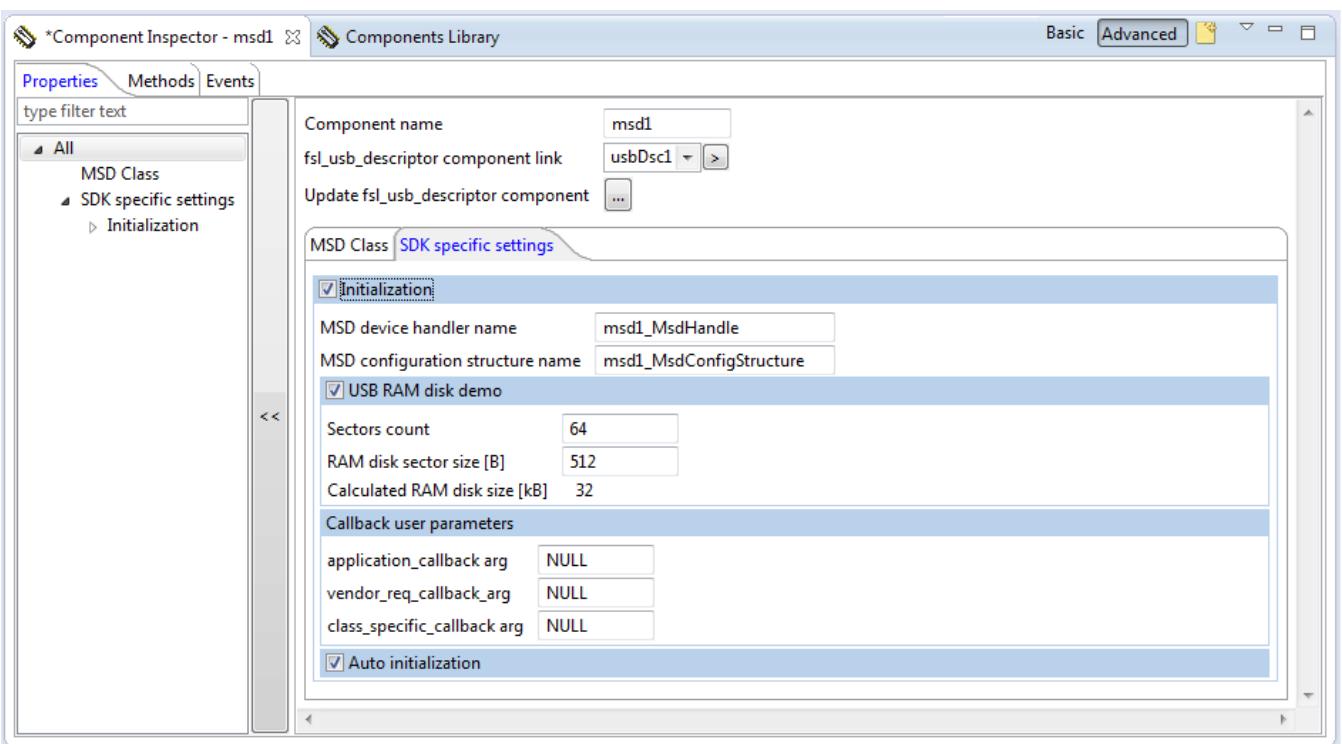


**Figure 29. Properties for MSD class initialization**

4. Press code generation button. As soon as PEx generation is done, all required files should be created /added in the project.

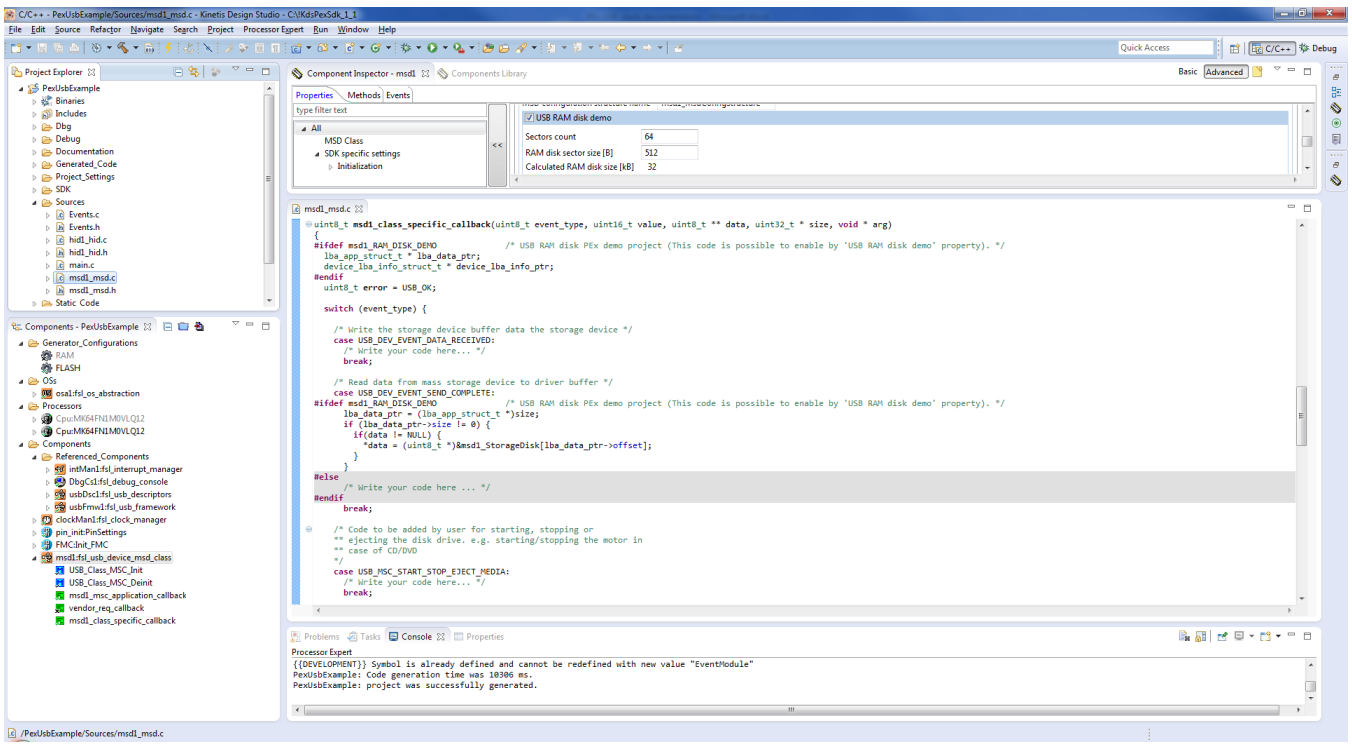


**Figure 30. MSD class callbacks API (with RAM disk demo code) generated by Processor Expert**

5. Build and load USB project into TWR-MK65F180M board.
6. Plug-in the USB connector of MK65F180M to PC.
7. The windows will prompt you to format the disk.

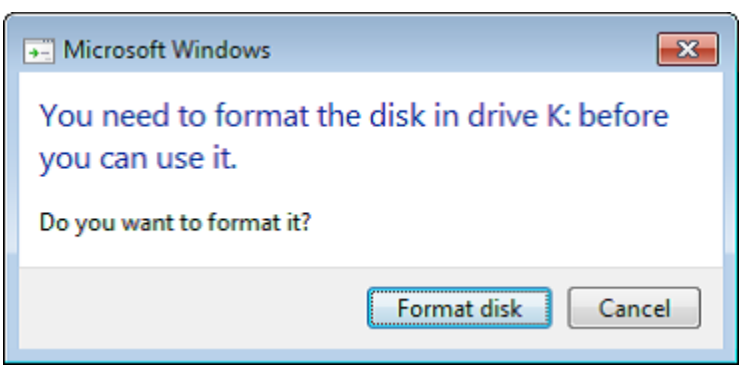


**Figure 31. OS Windows requires to format disk**

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

## 4.1.2 USB HID project

1. For USB HID project add the *fsl_usb_device_hid_class* component from „Components Library“ window to the created project (see chapter Creating common PEx USB Project).

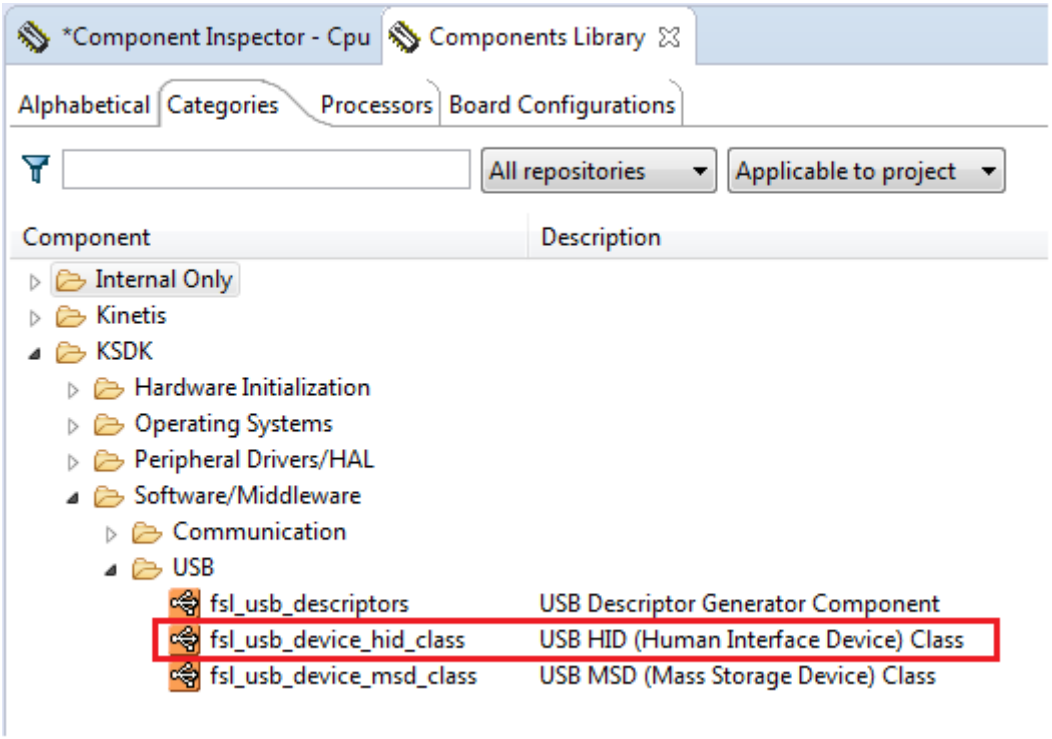


**Figure 32. USB components in "Component Library"**

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

After adding the *fsl_usb_device_hid_class* component to the project, *fsl_usb_device_hid_class* component must be configured to specify HID type. HID type is specified by Subclass, Protocol code and Report descriptor properties.

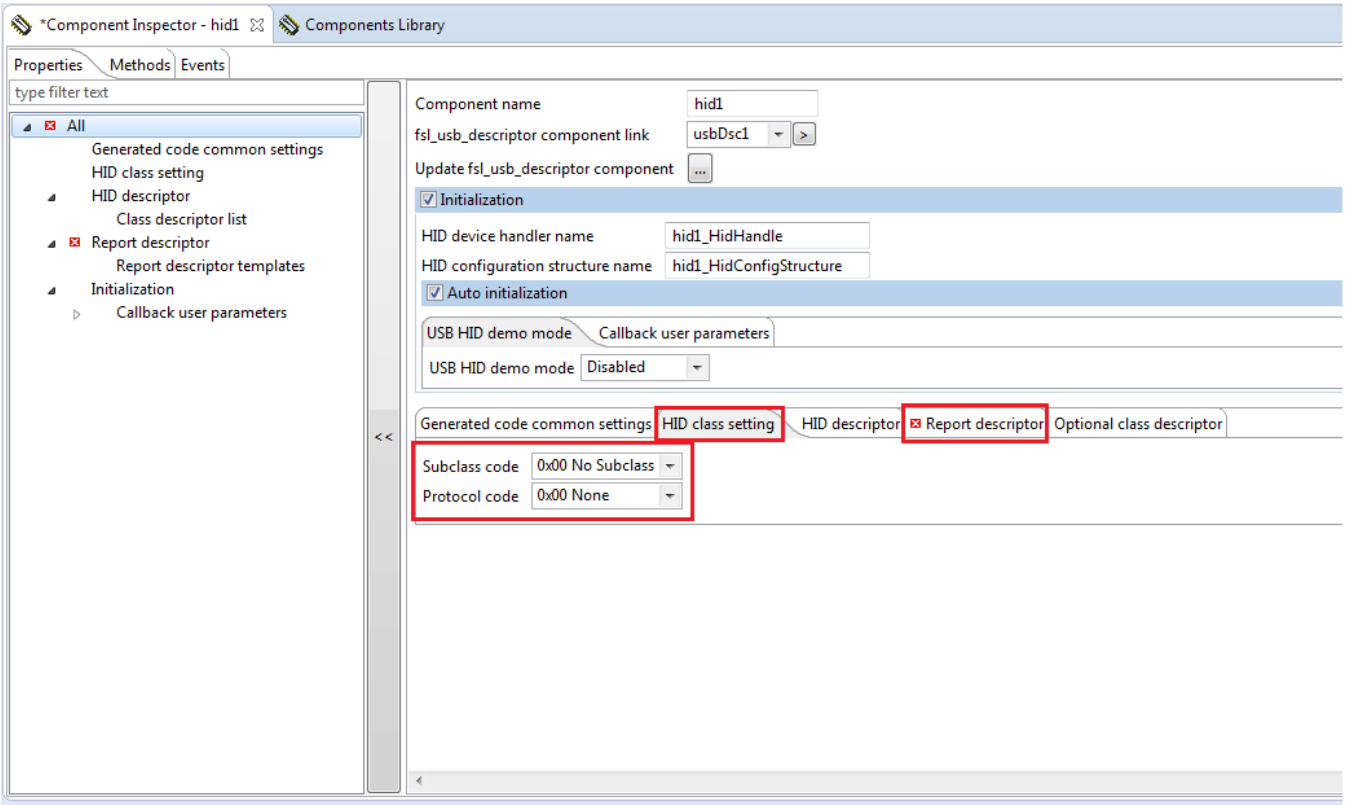


**Figure 33. HID type configuration**

Subclass, Protocol code and HID Report descriptor properties should be configurated according USB HID specification. For better fsl_usb_device_hid_class component function demonstration, fsl_usb_device_hid_class component contains HID type configuration properties:

- **Report descriptor template buttons** – configure report descriptor according to the one predefined templates: mouse, keyboard, thermometer.

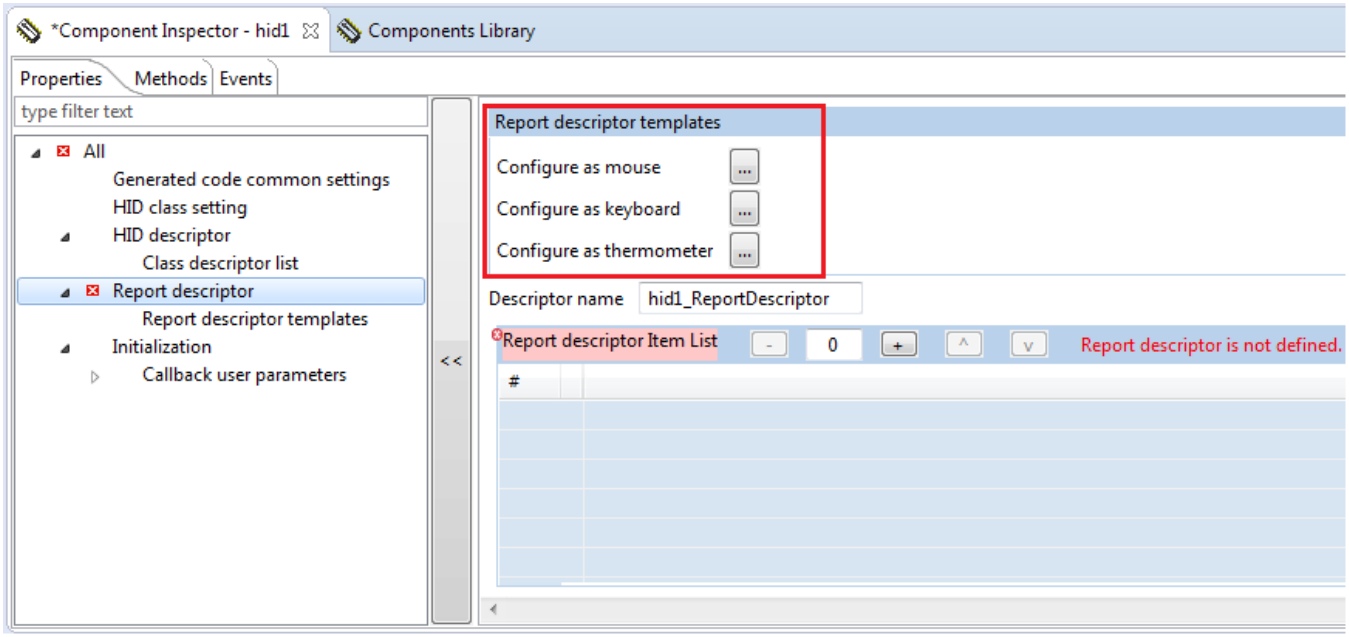


**Figure 34. HID template buttons**

- **USB HID demo mode** – configure report descriptor according according selected demo type and enable demo code.

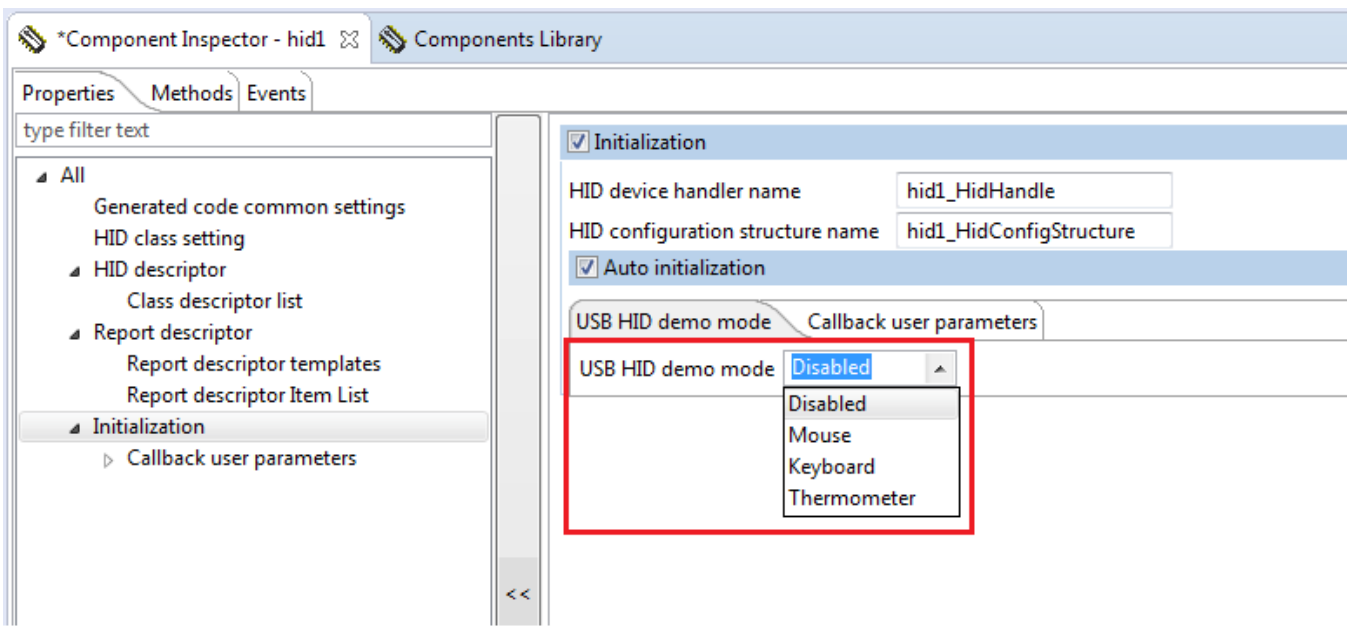


**Figure 35. USB HID demo mode selection**

HID template/demo types description:

- **Mouse** – Report descriptor is configured to standard three-button mouse detectable by BIOS. In mouse demo mode is mouse pointer left-right moved. Speed of mouse pointer move is defined by mouse PIPE „Polling interval“ (set to 128ms after demo mode activation) property in *fsl_usb_descriptors* component.

- **Keyboard** - Report descriptor is configured to standard keyboard detectable by BIOS. In keyboard demo mode are repeatedly PageUp/Down keys pressed. Speed of PageUp/Down keys pressed is defined by keyboard PIPE „Polling interval“ (set to 128ms after demo mode activation) property in *fsl_usb_descriptors* component.
- **Thermometer** - Report descriptor is configured to thermometer device, temperature unit = Kelvin, resolution = 1K, minimum temperature value = 218K, maximum temperature value = 393K. In thermometer demo mode are repeatedly 0K and 123K values sent to PC.
  For reading temperature value (on PC side) from HID t hermometer is need user application. Access to t hermometer in Windows OS is through standard windows functions:
  - ***CreateFile()*** – Creates or opens a file or I/O device,
  - ***ReadFile()*** – Reads data from the specified file or input/output (I/O) device,
  - ***WriteFile()*** - Writes data to the specified file or input/output (I/O) device,
  - ***CloseHandle()*** - Closes an open object handle.

For open a HID device (get handle) by ***CreateFile()*** function is need know ***lpFileName*** parameter value. ***lpFileName*** parameter specifies device name to be opened. ***lpFileName*** parameter value is possible to get by functions defined in “hid.dll” library according USB device Product ID & Vendor ID values (PID & VID values of USB device are defined in *fsl_usb_descriptors* component). “hid.dll” library is available in Windows \System32 folder.

List of functions used to get HID ***lpFileName***:

- ***HidD_GetHidGuid()*** – returns the device interface GUID for HIDClass devices.
- ***SetupDiGetClassDevs()*** – returns a handle to a device information set that contains requested device information elements for a local computer.
- ***SetupDiEnumDeviceInterfaces()*** – numerates the device interfaces that are contained in a device information set.
- ***SetupDiDestroyDeviceInfoList()*** – deletes a device information set and frees all associated memory.
- ***SetupDiGetDeviceInterfaceDetail()*** – returns details about a device interface.
- ***HidD_GetAttributes()*** – returns the attributes of a specified top-level collection.

More information about HID access functions is available here: https://msdn.microsoft.com.

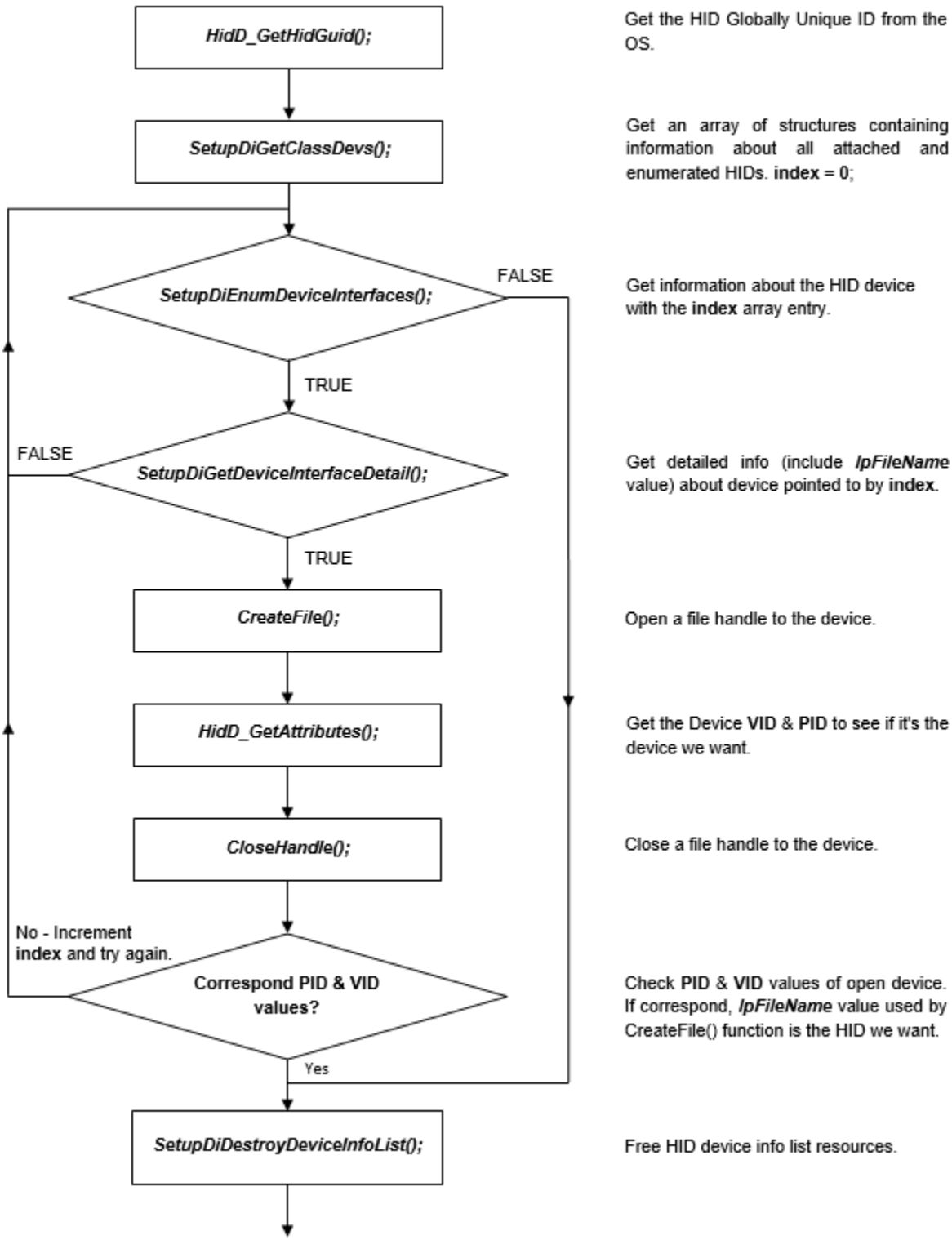


**Figure 36. How to get HID lpFileName value flowchart**

2. Press the code generation button. As soon as PEx generation is done, all required files should be created /added in the project.

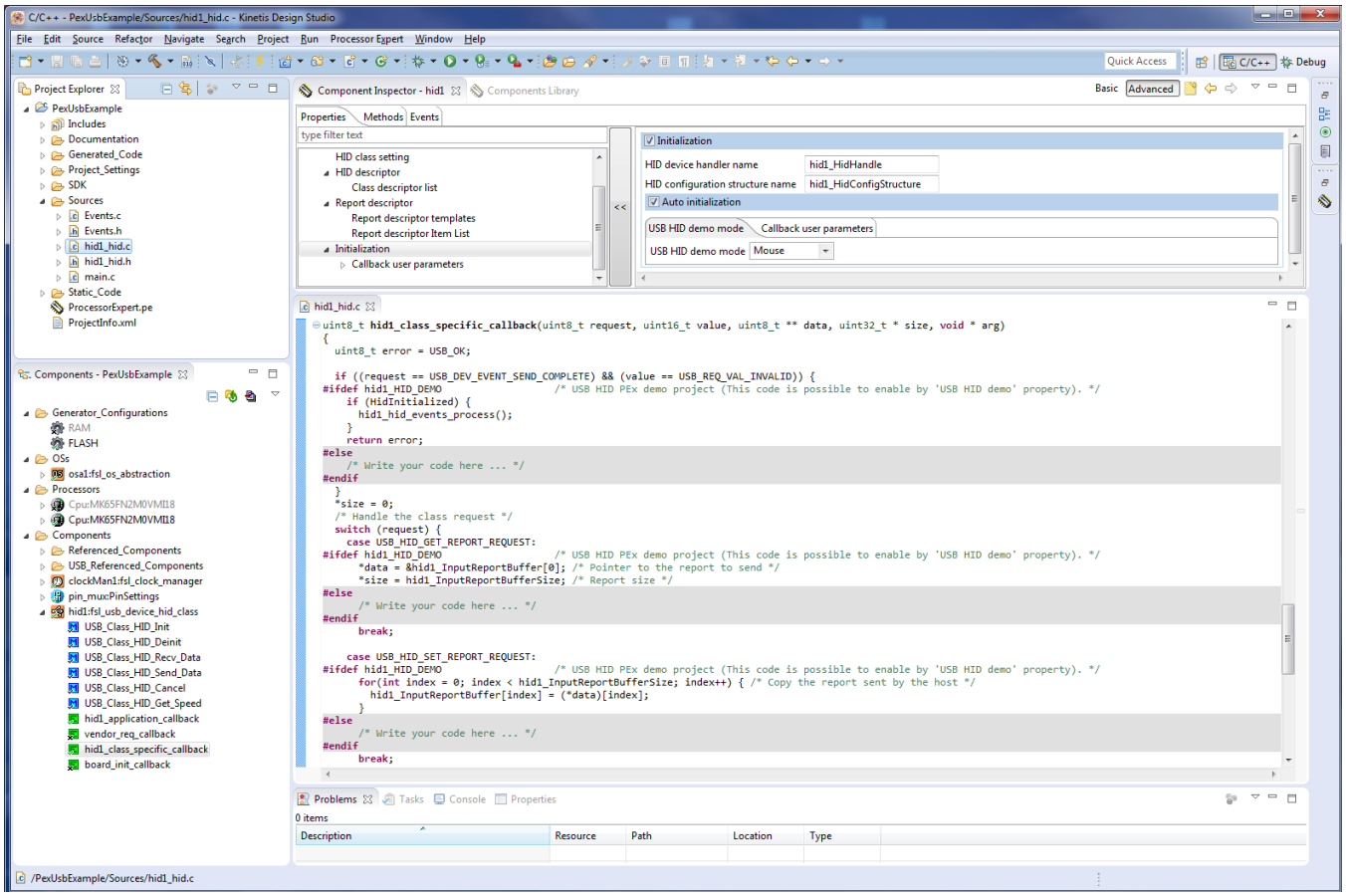


**Figure 37. HID class callbacks API with Mouse demo code generated by Processor Expert**

3. Build and load USB project into TWR-MK65F180M board.
4. Plug-in the USB connector of MK65F180M to PC.

***How to Reach Us:***

**Home Page:**
freescale.com

**Web Support:**
freescale.com/support



Document Number PEXKSDKUSBSI
Revision 0, 07/2015

freescale™