Document Number: PEXKSDKRTOSUG
Rev. 1.0.0, 04/2015

Processor Expert Kinetis SDK RTOSes Integration User's Guide

1 Introduction

Processor Expert supports Kinetis SDK 1.2.0 version FreeRTOS, MQX (KSDK variant in Lite and Standard configurations), uCOSII, and uCOSIII RTOSes. Processor Expert handles per each RTOS source files, compiler settings and helps you to create tasks using OS_Task component.

The main goal of the Processor Expert component is a smooth creation of an RTOS-based applications.

RTOSes source files are not a part of the basic Kinetis SDK package. However, you must either select them in Kinetis SDK installer or in the corresponding RTOS package and copy them into the Kinetis SDK file structure.

Contents

2 Terminology

Term	Description	Web page
Kinetis Design Studio (KDS)	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.	www.freescale.com/kds
Processor Expert software	Development system to create, configure, optimize, migrate, and deliver software components that generate source code for Freescale silicon. Processor Expert.	www.freescale.com/processorexpert
Kinetis Software Development Kit (KSDK)	Extensive suite of robust peripheral drivers, stacks, middleware and example applications designed to simplify and accelerate application development on any Kinetis MCU	www.freescale.com/ksdk

3 Basic Usage

All Kinetis SDK Processor Expert projects start with fsl_os_abstraction in project, by default. However, you must select the RTOS type from the OS drop-down menu of the *fsl_os_abstraction*, component The fsl_os_abstraction component/API helps in writing OS independent application.

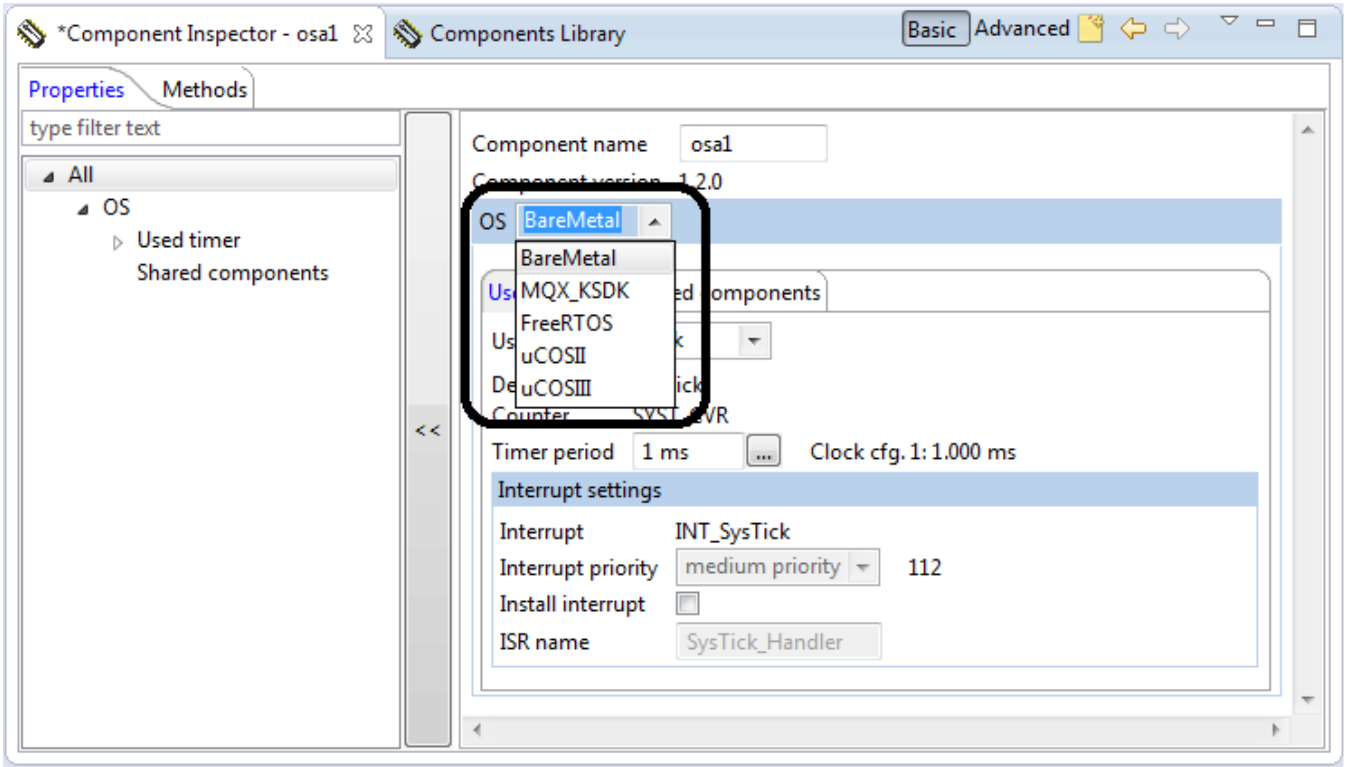

Figure 1. fsl_os_abstraction - RTOS type selection

The RTOS task can be initialized/started by selecting the OS_Task component from the **Components Library** view and adding it to the project.

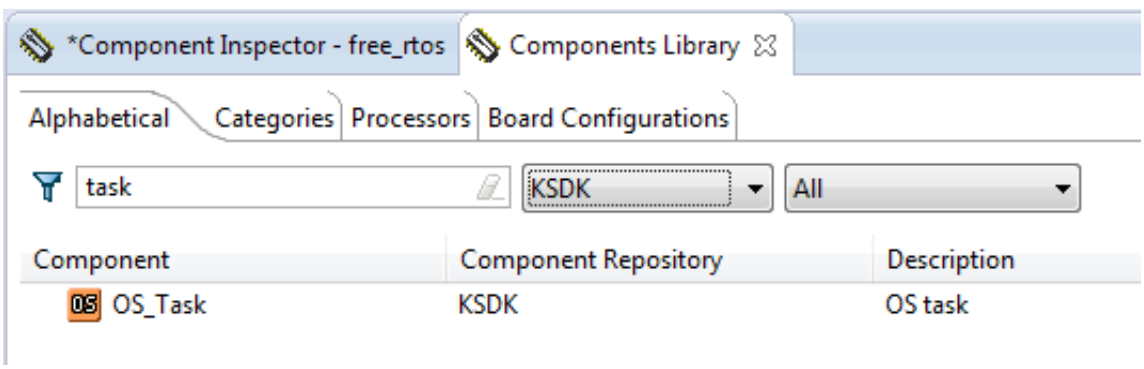

Figure 2. OS_Task component in Components Library

NOTE

The tasks with enabled **Auto initialization** property, start automatically after the RTOS initialization/startup.

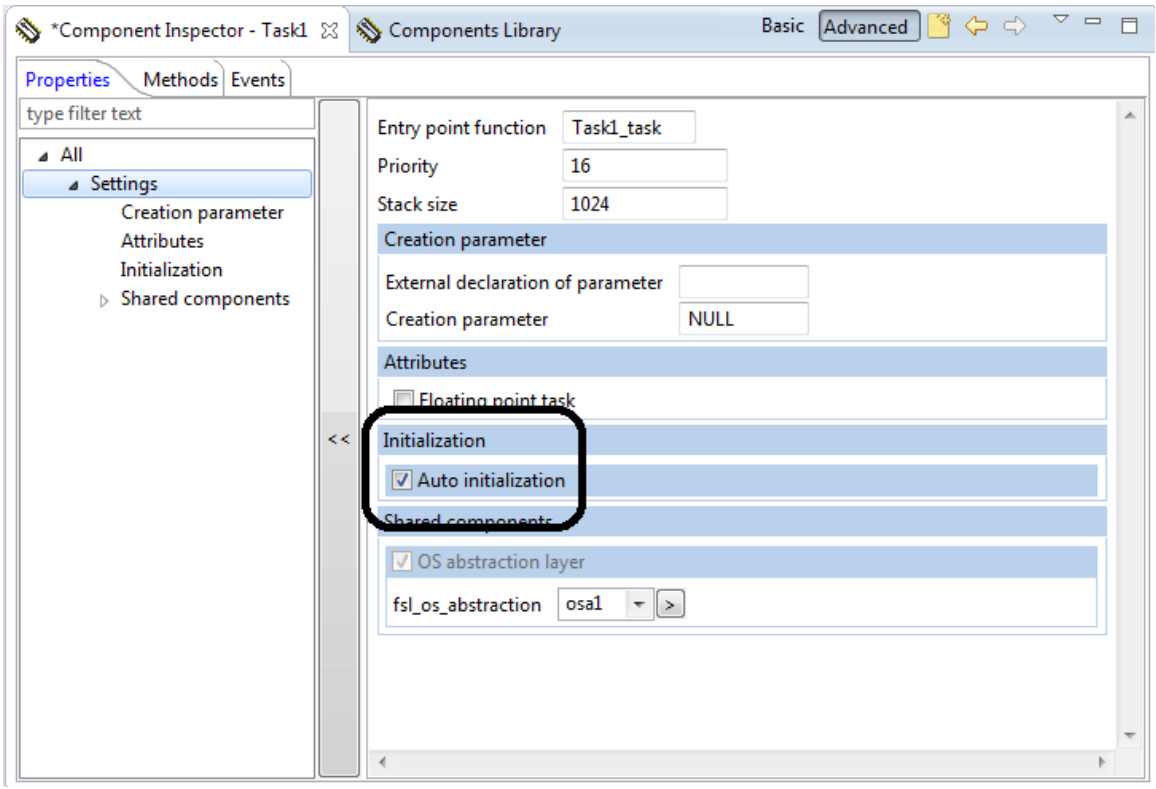

Figure 3. OS_Task component default configuration

After Processor Expert code generation, all required RTOS files and paths are added to the project and a task body is generated for the specific module. For example, os_tasks.c default. You can fill the task body with user code.

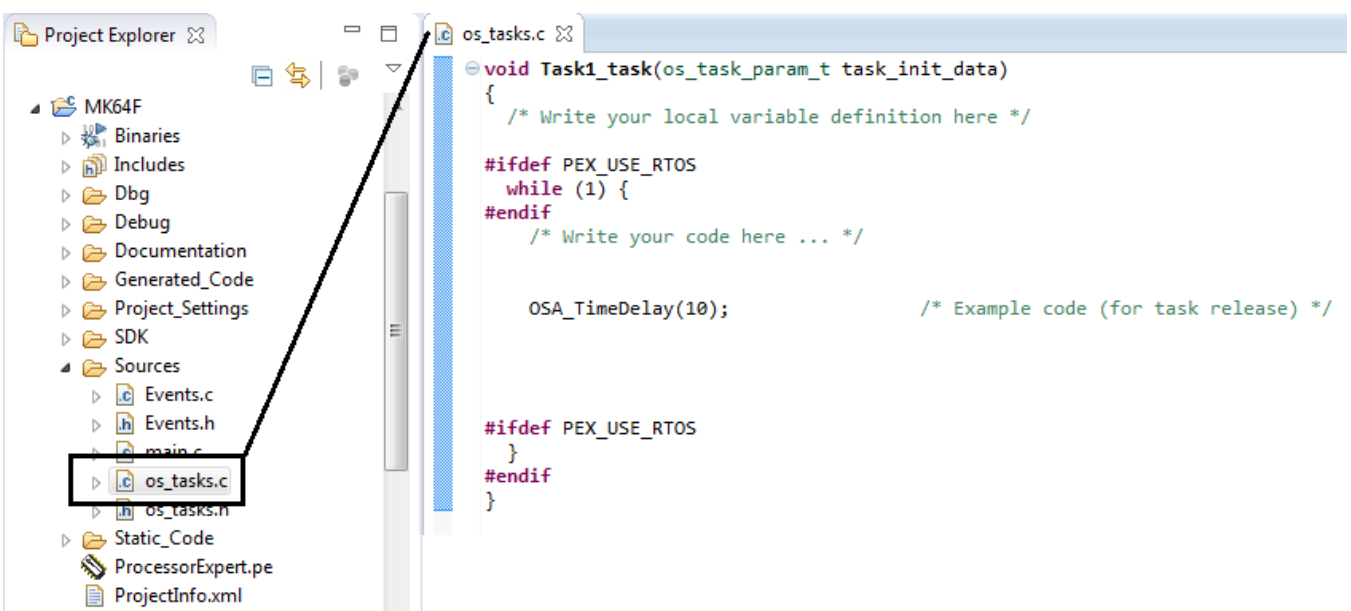

Figure 4. OS_Task C module and task body

4 Processor Expert RTOS Integration Structure

Processor Expert RTOS integration layers are based on RTOS layers and drivers supported in Kinetis SDK. Processor Expert tool provides an easy to use way for using RTOS in a project.

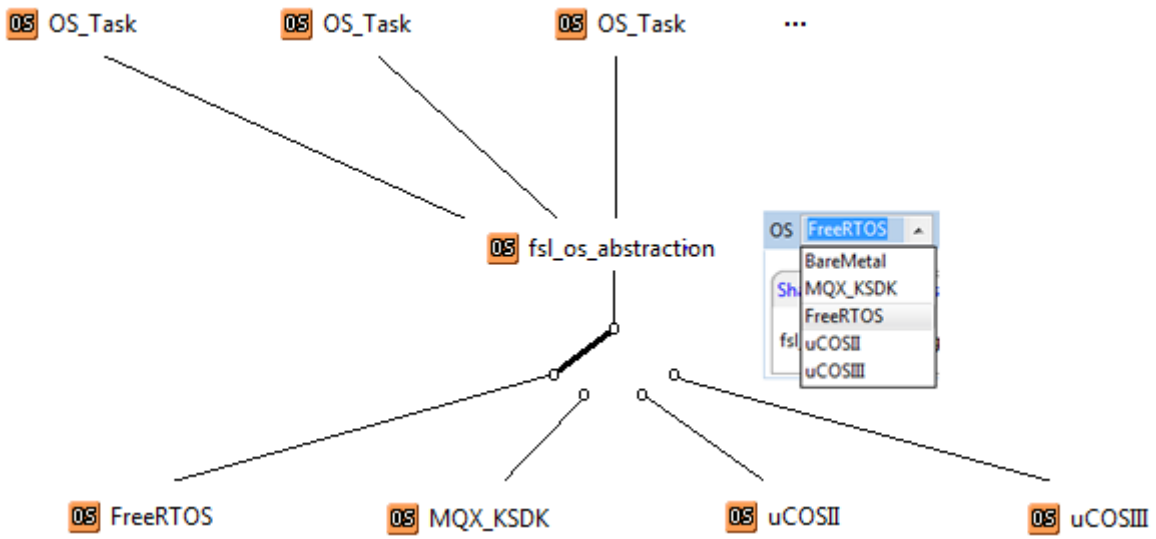

Figure 5. Processor Expert RTOS integration structure.

5 Processor Expert SDK RTOS Layers

5.1 fsl_os_abstraction

The *fsl_os_abstraction* (OSA) component allows BareMetal/RTOS project mode selection and provides the same API for *BareMetal* or all supported RTOSes. RTOS API functions of RTOS component (e.g. FreeRTOS, uCOSII/III, MQX_KSDK) can also be used, however application becames RTOS dependent and future transformation to another RTOS can be more problematic. BareMetal/RTOS project mode is selected from the OS drop-down list in the **Components Library** view.

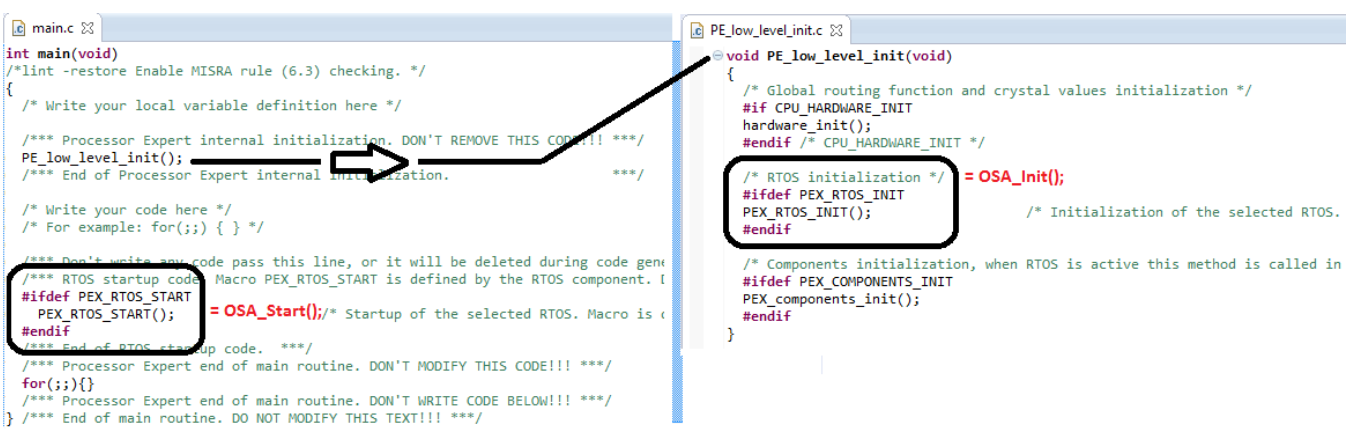

Figure 6. OS Property - BareMetal/RTOS project mode selection

BareMetal or RTOS OSA mode is initialized by OSA_Init() method and started by OSA_Start() method. OSA_Init() method is called in PE_low_level_init() function and OSA_Start() method on end of main() function.

5.2 BareMetal

The default value of the **OS** property is **BareMetal** mode. In this mode the *fsl_os_abstraction* method function includes task, semaphore, mutex, event, and software emulated by fsl_os_abstraction driver. For **BareMetal** mode timing, synchronization uses **SysTick** (ARM core System timer) as default. The following are the **Used Timer** options available in Processor Expert.

- **None** - **BareMetal** mode without timing function .
- **LPTMR** – Low power timer is used for OSA **BareMetal** timing function.
- **Systick** – System timer is used for OSA **BareMetal** timing function.
- **User defined** – In this mode user can write code for OSA **BareMetal** timing function.

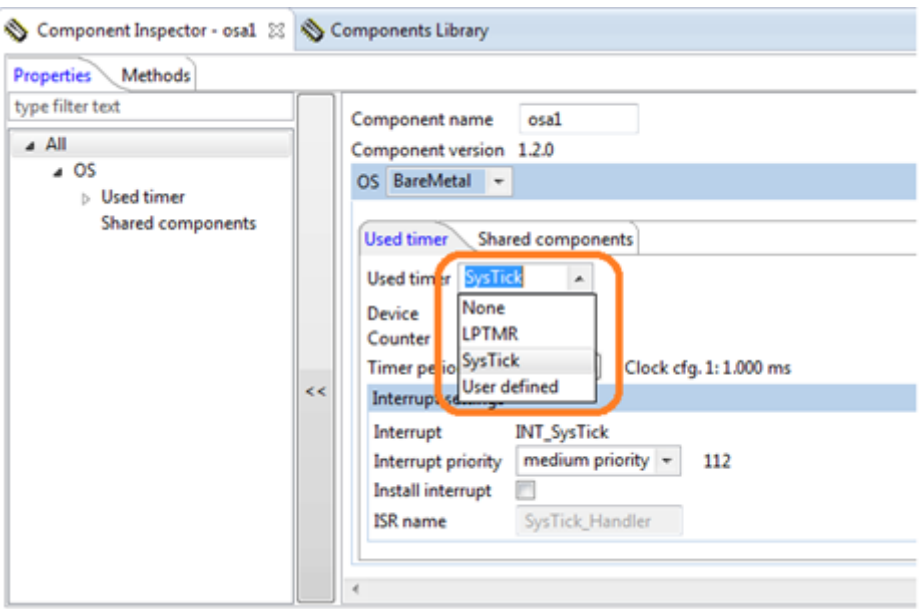

Figure 7. BareMetal – Used timer selection property

The **Tasks** function in the **BareMetal** mode is emulated and the next task (Task2 code teil) function is started after exiting from the previous task (Task1 code teil).

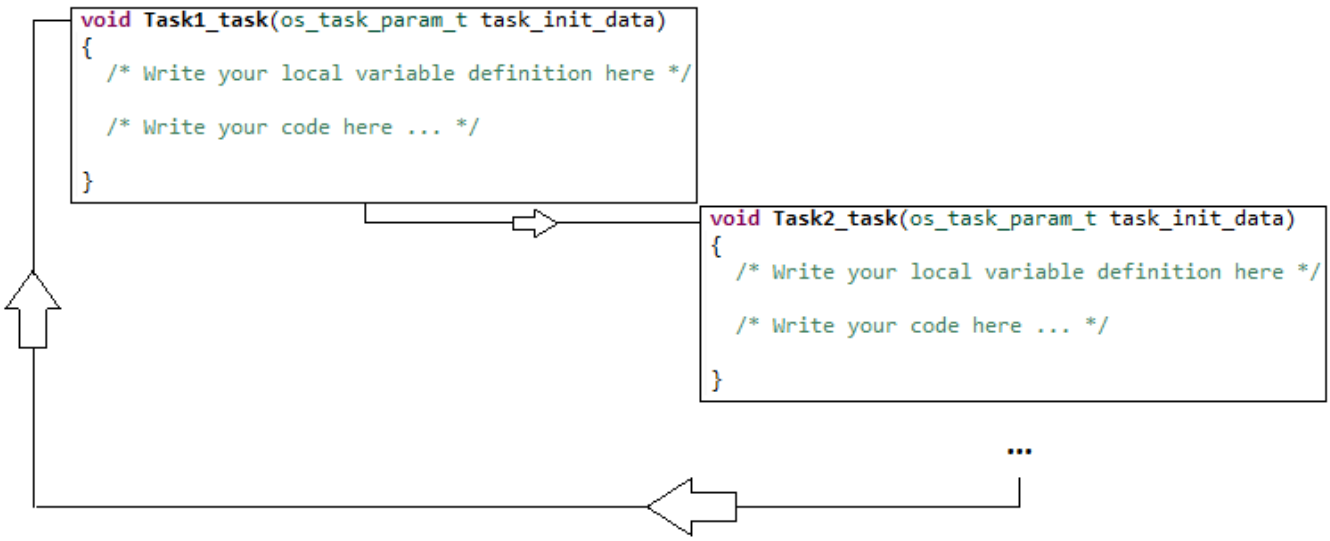

Figure 8. BareMetal – Emulating Task functions

5.3 FreeRTOS

The *FreeRTOS* component is inherited by the *fsl_os_abstraction* component. The *fsl_os_abstraction* component uses function of the FreeRTOS component. For better code migration between RTOS types, use the fsl_os_abstraction components methods or the FreeRTOS components methods.

The basic FreeRTOS component functions are to:

- Add all needed RTOS source files and paths to the project.
- Configure FreeRTOS > create *FreeRTOSConfig.h* configuration file. The content of the configuration file is specified by the *FreeRTOS* component properties settings.

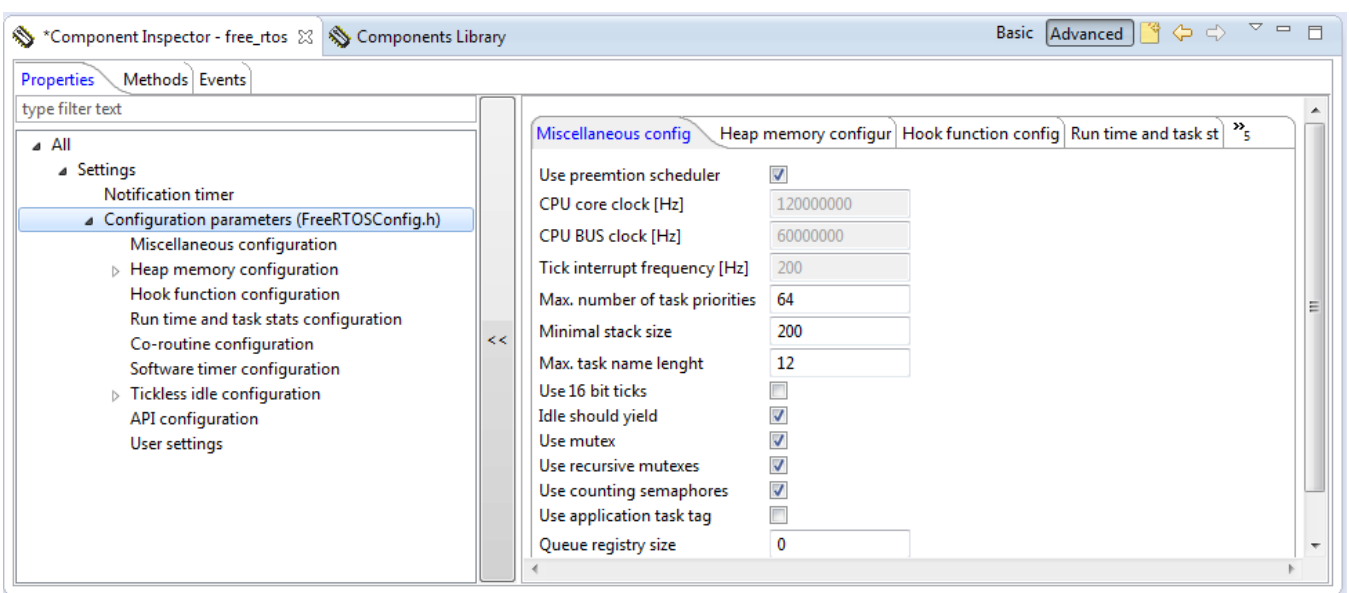

Figure 9. FreeRTOS component – Configuration properties

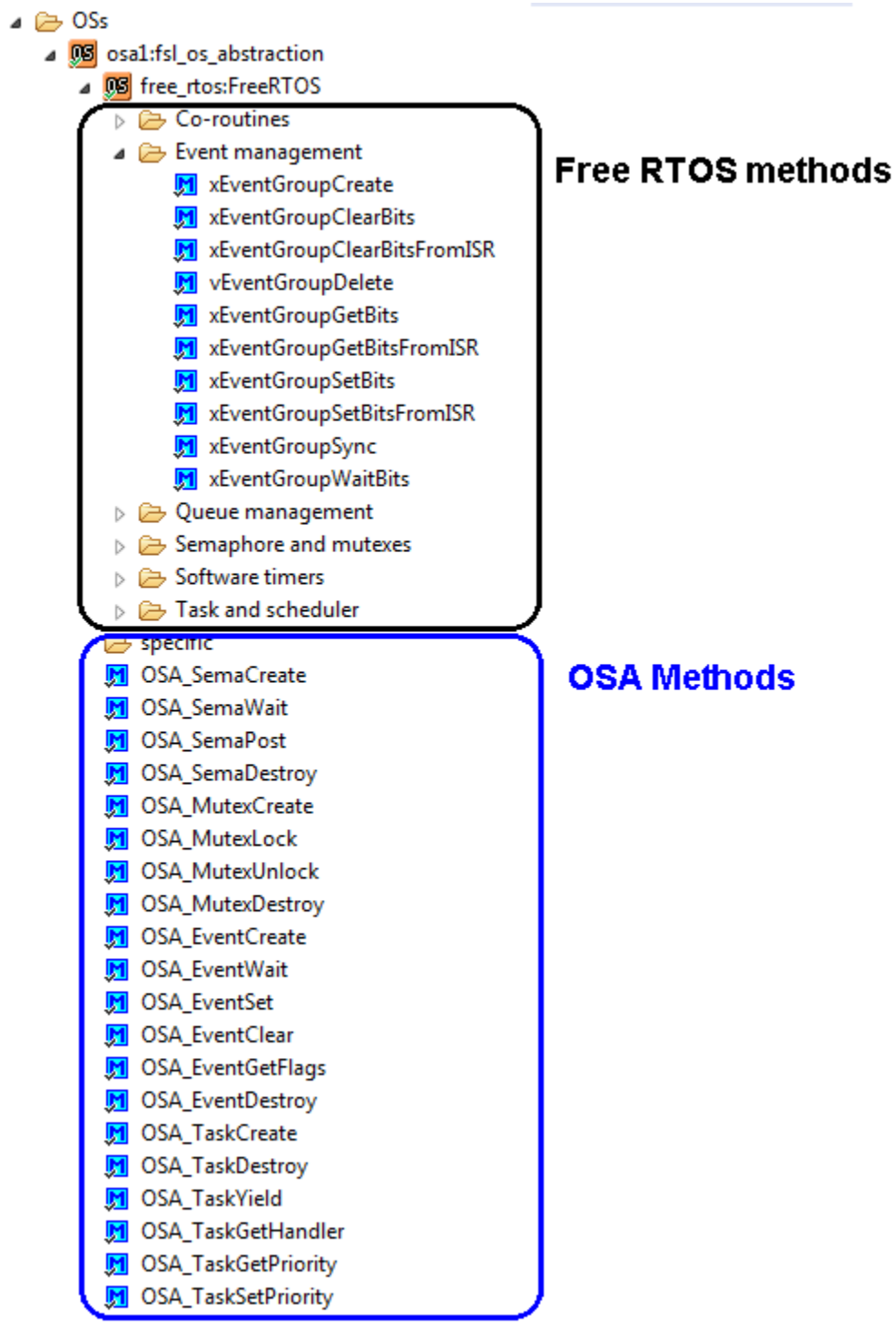

Figure 10. FreeRTOS and OSA methods (API)

5.4 MQX_KSDK

The MQX_KSDK component is inherited by the fsl_os_abstraction component. The *fsl_os_abstraction* component uses the function of the *MQX_KSDK* component. The MQX_KSDK component supports MQX-Standard and MQX-Lite configurations.

The basic MQX_KSDK component functions are to:

- Add all needed RTOS source files and paths to the project and accordingly select the configuration set.
- Create MQX configuration files.
- Configure compiler parameters.

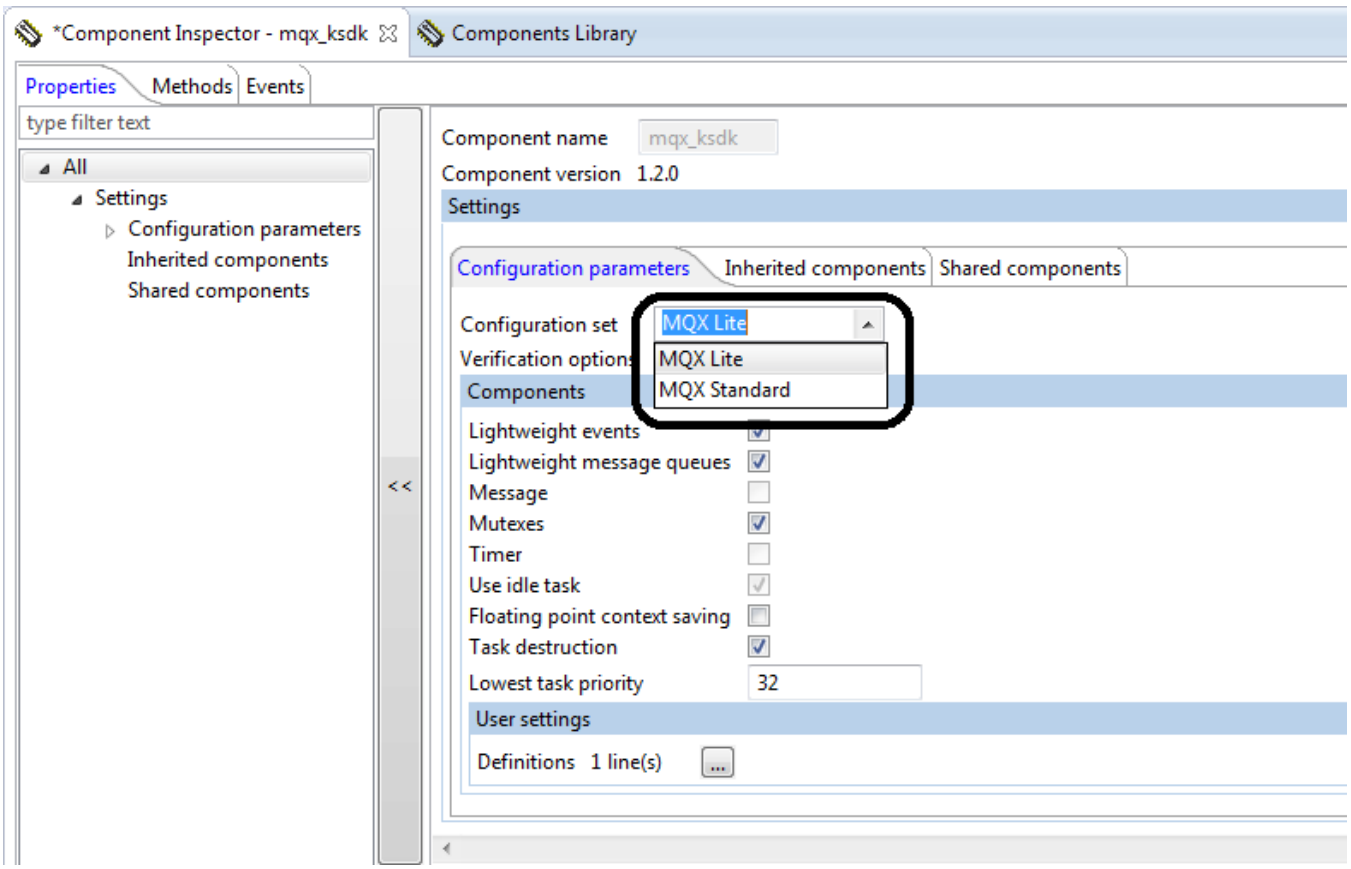

Figure 11. MQX_KSDK MQX-Standard/Lite configuration

For *MQX-Lite configuration set* the initialization scenario of the Processor Expert components is the same as for other RTOS/ BareMetal mode (Init() functions of components are called between OSA_Init() and OSA_Start() function).

For MQX-Standard the *init* methods of Processor Expert components are called within the *main_task* task; the task body is created within Sources/rtos_main_task.c file. The *main_task* is automatically added to the project, inherited by MQX-Standard configuration, and must be set to the highest task priority. The main_task initiates or starts as first task in the project.

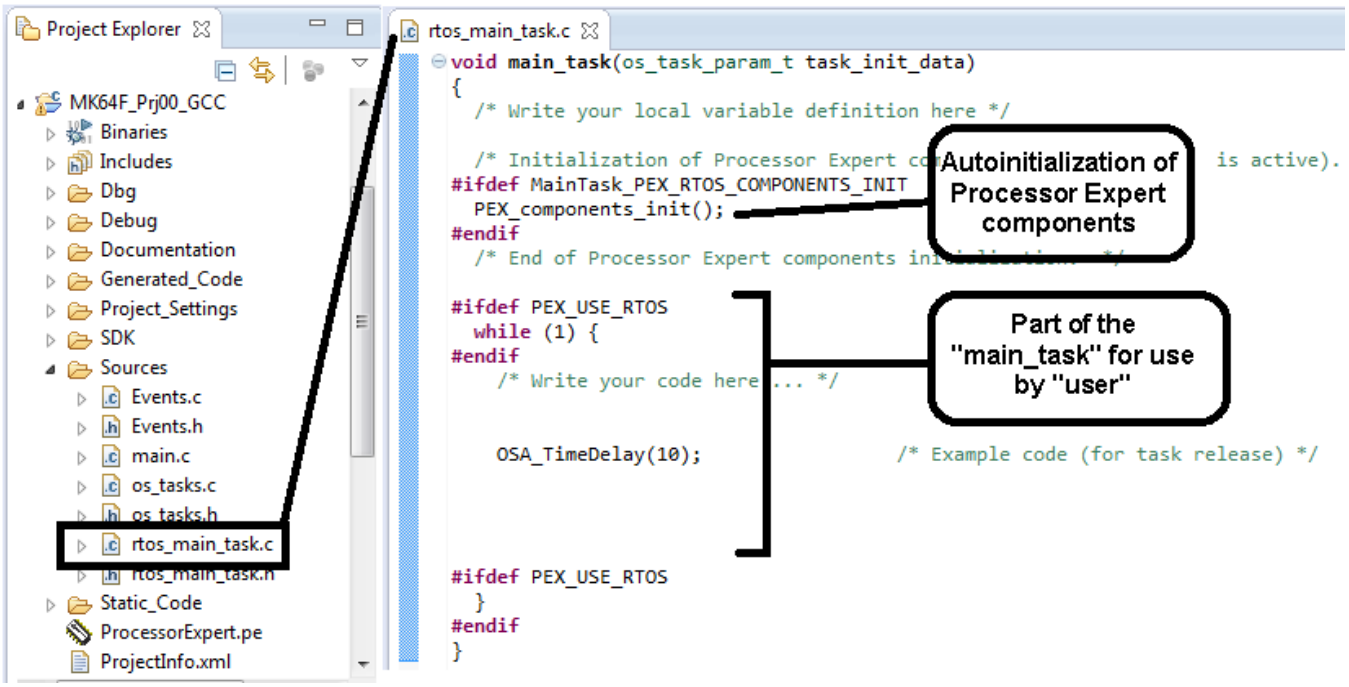

Figure 12. MQX-Standard – rtos_main_task

5.5 uCOSII

The *uCOSII* component is inherited by the *fsl_os_abstraction* component. The *fsl_os_abstraction* component uses function of the *uCOSII* component. s. For better code migration between RTOS types, use the *fsl_os_abstraction* components methods or the *uCOSIII* components methods.

The basic uCOSII component functions are to:

- Add all needed uCOSII source files and paths to the project.
- Configure uCOSII RTOS > creates *app_cfg.h*, *lib_cfg.h,* and *os_cfg.h* configuration files. The content of configuration files is specified by the uCOSII component properties settings.

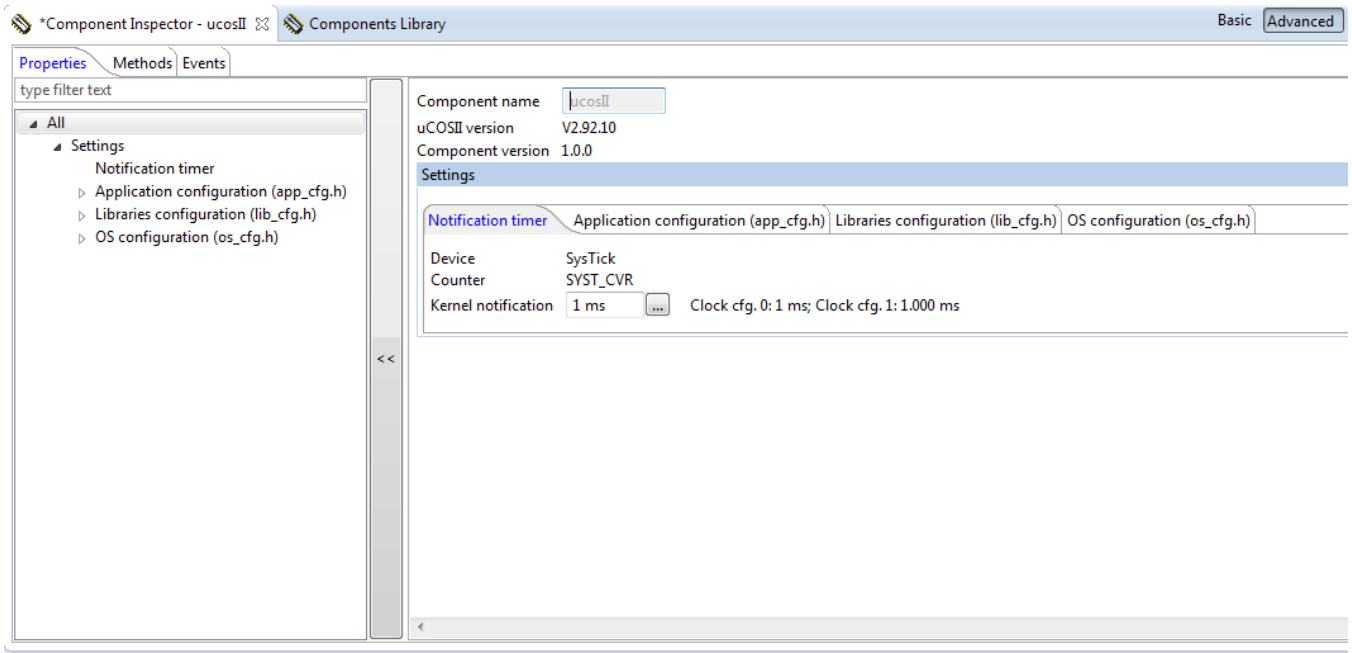

Figure 13. uCOSII component – Configuration properties

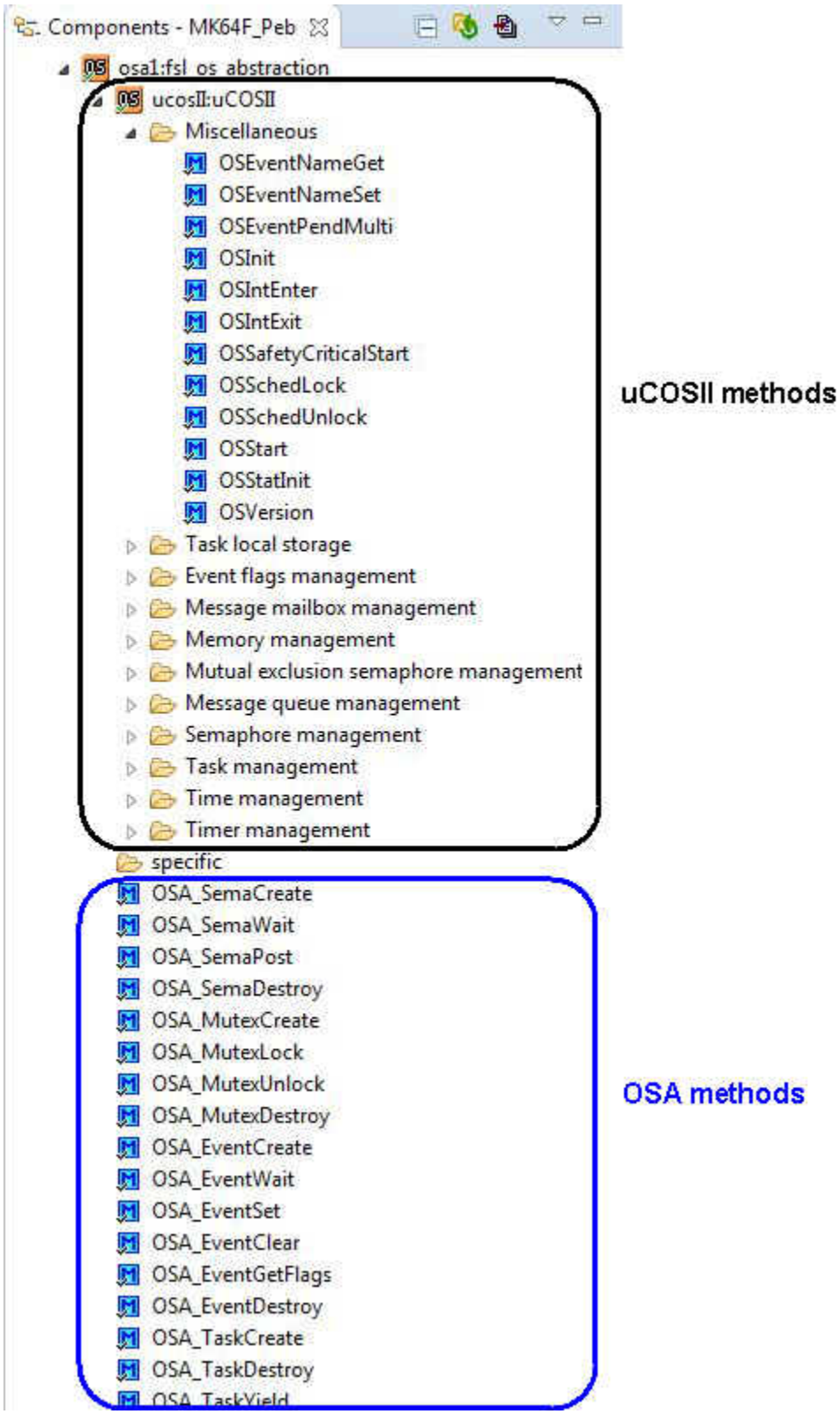

Figure 14. uCOSII and OSA methods (API)

5.6 uCOSIII

The uCOSIIIcomponent is inherited by the *fsl_os_abstraction* component. The *fsl_os_abstraction* component uses function of the *uCOSIII* component. For better code migration between RTOS types, use the *fsl_os_abstraction* components methods or the *uCOSIII* components methods.

The basic uCOSIII component functions are to:

- Add all needed uCOSIII source files and paths to the project.
- Configure uCOSIII RTOS > creates *app_cfg.h*, *lib_cfg.h*, and *os_cfg.h* configuration files. Content of configuration files is specified by the *uCOSIII* component properties settings.

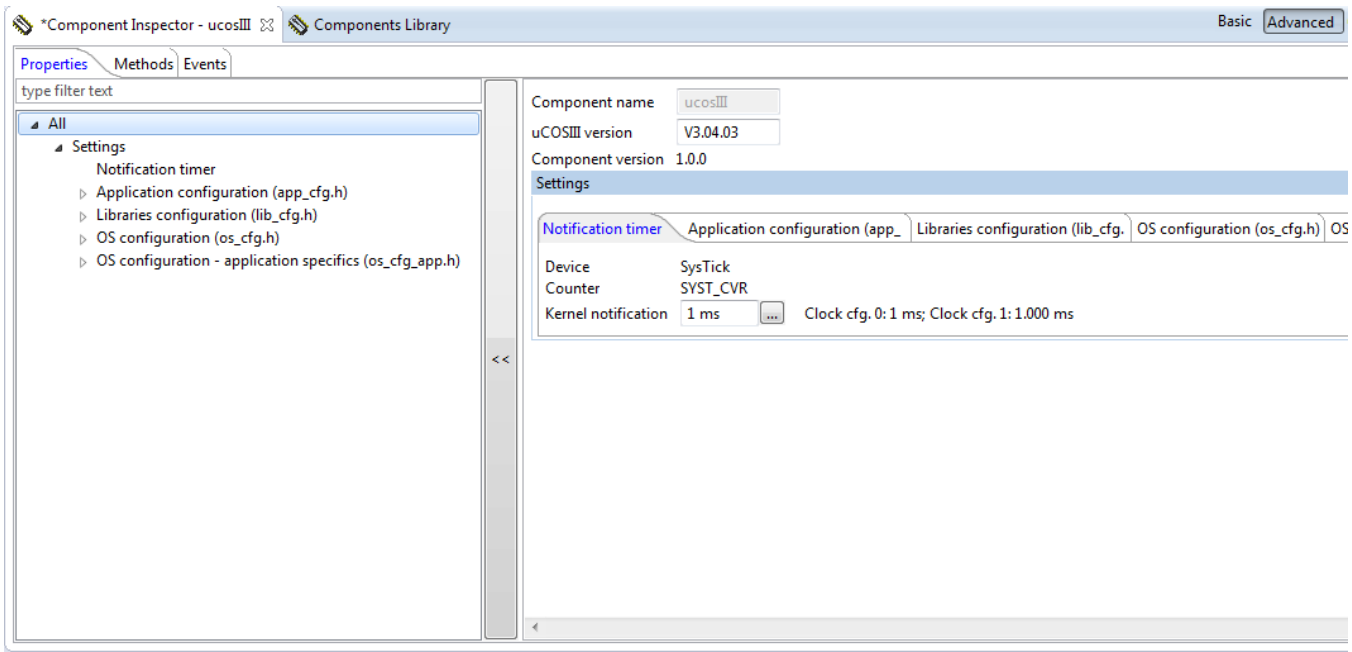

Figure 15. uCOSIII component – Configuration properties

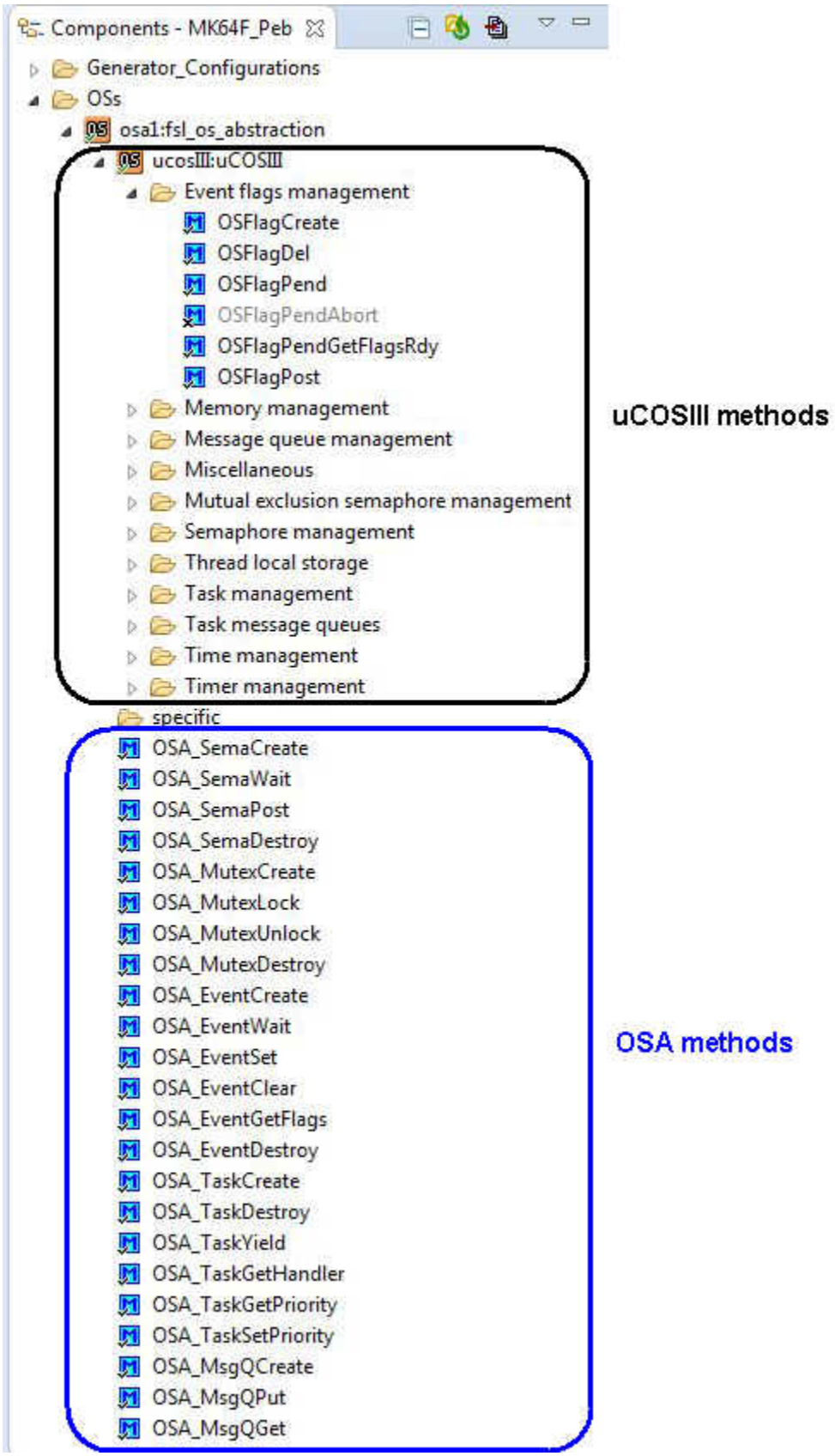

Figure 16. uCOSIII and OSA methods (API)

5.7 OS_Task

OS_Task is software component designed for creating RTOS tasks. OS_Task uses OSA functions, which helps setup task stack size, priority, and pre-generates task body in defined the *.c module.

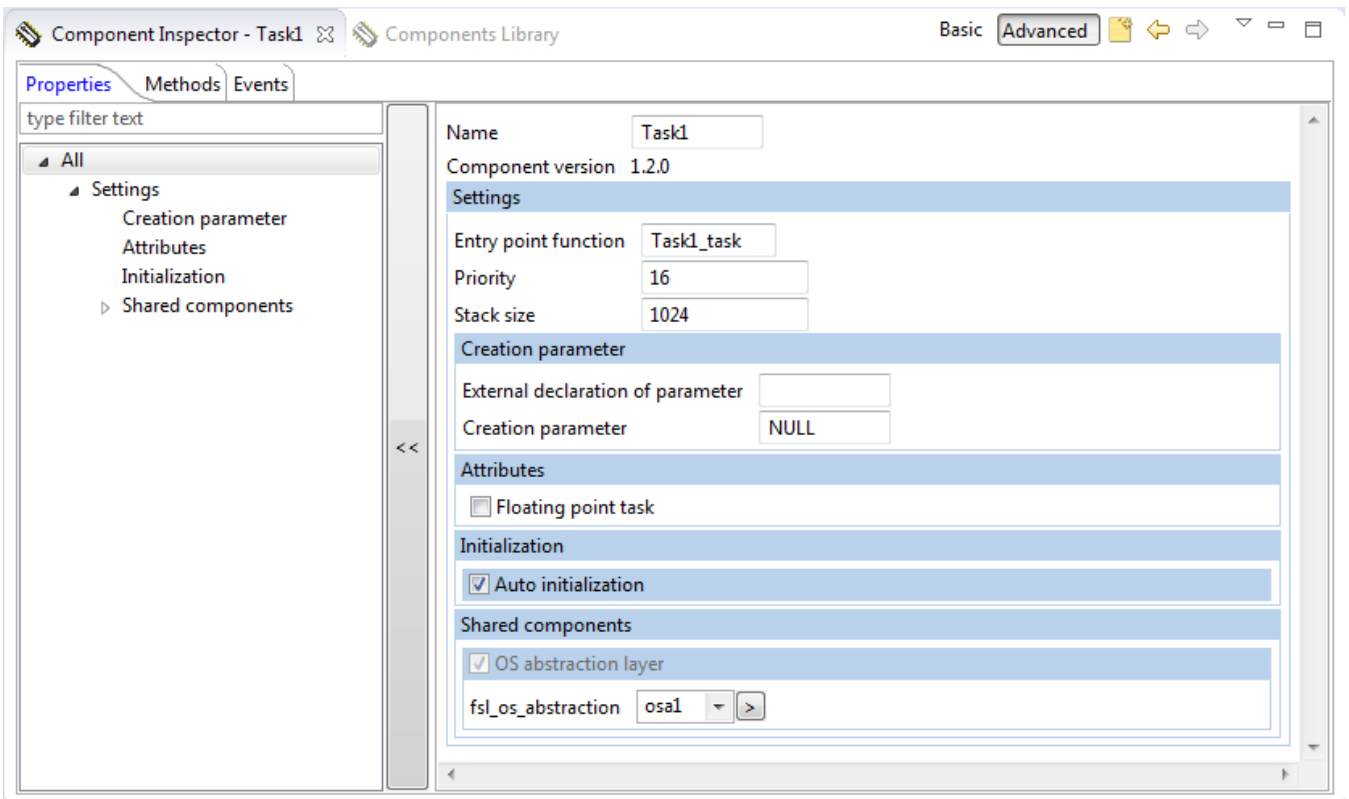

Figure 17. OS_Task component – Configuration properties

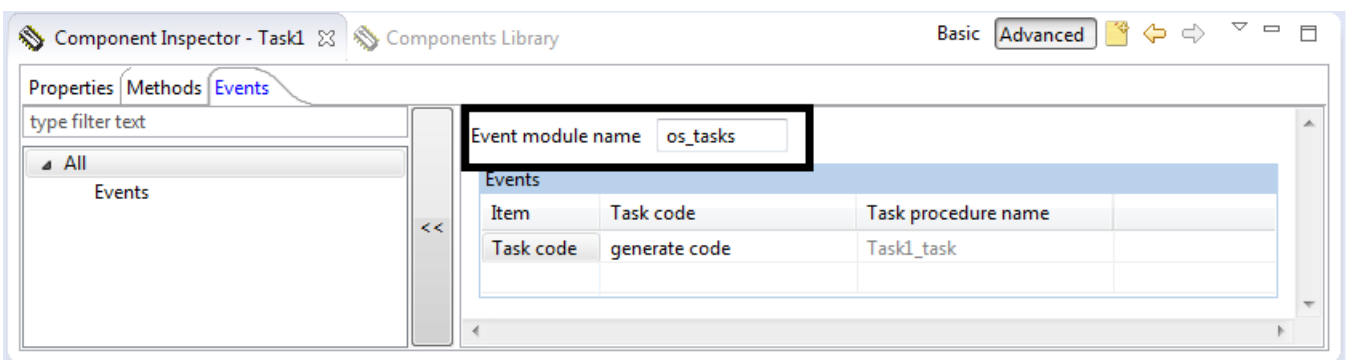

Figure 18. C module (os_tasks.c) where task body is generated

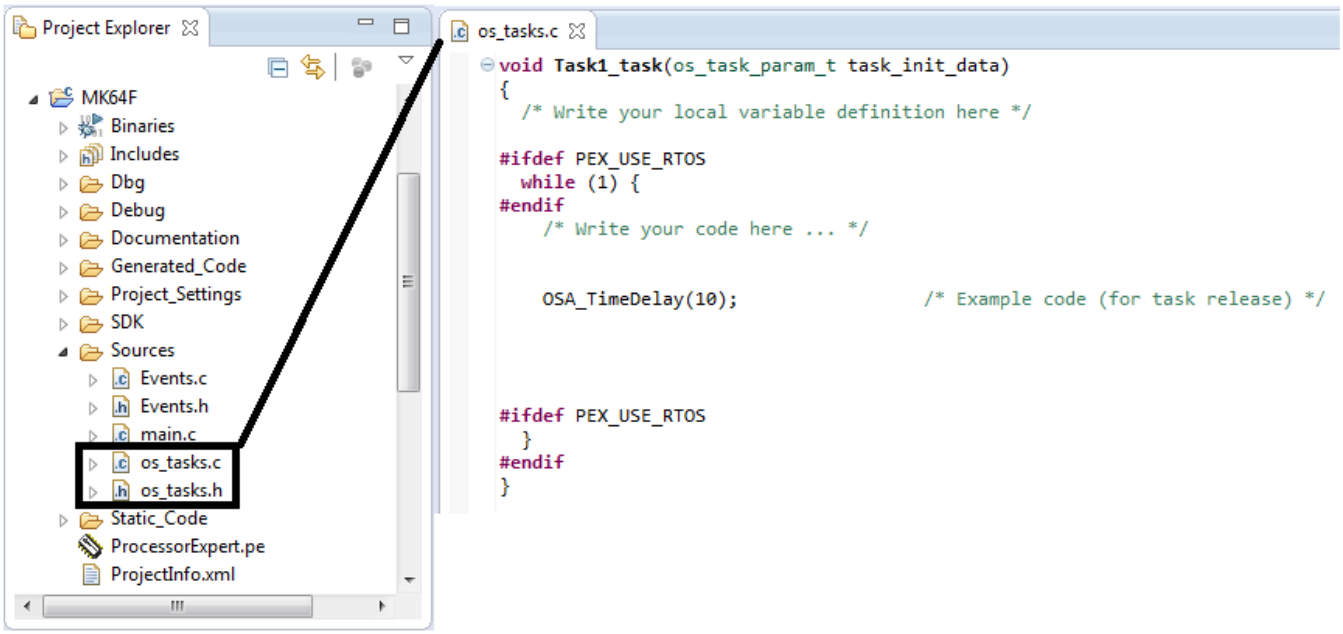

Figure 19. OS_Task C module and task body

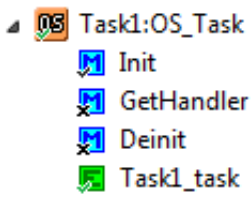

Figure 20. OS_Task methods/event (API)

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number PEXKSDKRTOSUG
Revision 1.0.0, 04/2015