Create a FreeRTOS project

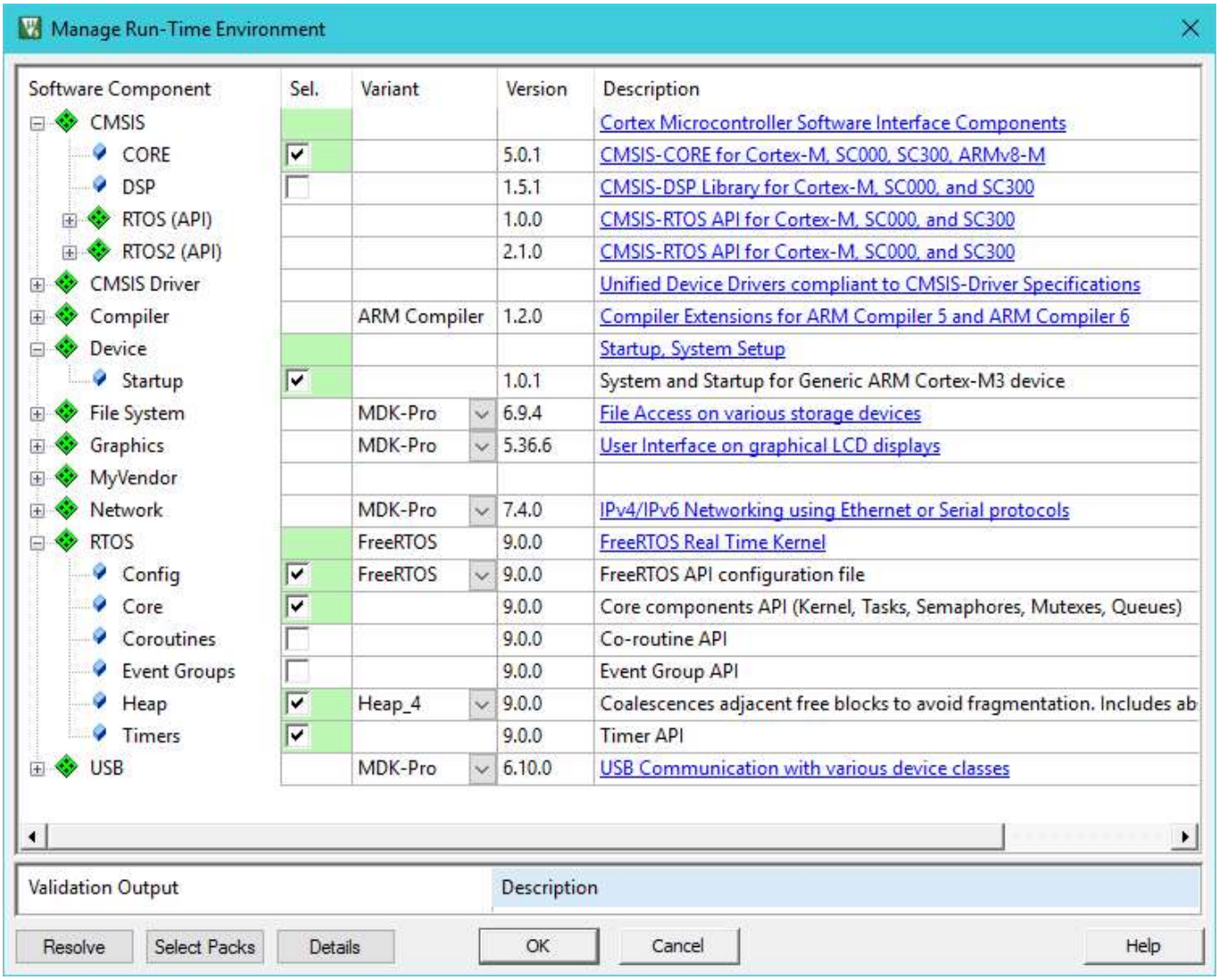
You can basically choose between two option when creating a FreeRTOS project:

1. **Create a native FreeRTOS project** using the FreeRTOS API and kernel.
2. **Create a CMSIS-FreeRTOS project** using the CMSIS-RTOS2 API with an underlying FreeRTOS kernel.

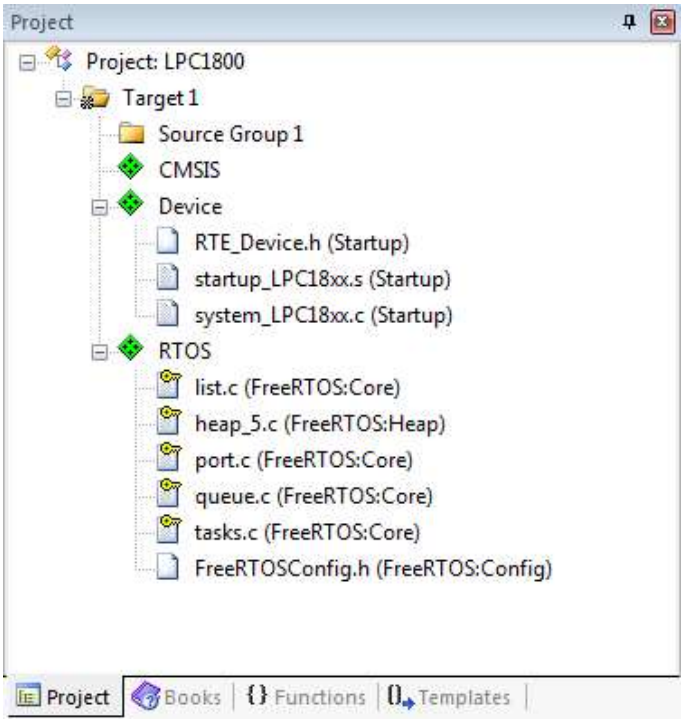
Create a native FreeRTOS project

The steps to create a microcontroller application using FreeRTOS are:

- Create a new project and select a microcontroller device.
- In the Manage Run-Time Environment window, select **::Device:Startup**, **::RTOS:CORE** and **::RTOS:Config** in the **FreeRTOS** variant and an applicable **::RTOS:Heap** scheme (for more information on the heap schemes, visit the FreeRTOS documentation):



- If the **Validation Output** requires other components to be present, try to use the **Resolve** button.
- Click **OK**. In the **Project** window, you will see the files that have been automatically added to you project, such as **FreeRTOSConfig.h**, the source code files, as well as the system and startup files:



Configure FreeRTOS

When you have created the native FreeRTOS project, you can configure the real-time operating system using the **FreeRTOSConfig.h** file. Please refer to the FreeRTOS documentation for more information on the specific settings.

FreeRTOSConfig.h

```
88
89  #define configCPU_CLOCK_HZ                    (SystemCoreClock)
90  #define configTICK_RATE_HZ                    ((TickType_t)1000)
91  #define configTOTAL_HEAP_SIZE                 ((size_t)(4096))
92  #define configMINIMAL_STACK_SIZE              ((unsigned short)130)
93  #define configCHECK_FOR_STACK_OVERFLOW        0
94  #define configMAX_PRIORITIES                  (5)
95  #define configUSE_PREEMPTION                  1
96  #define configIDLE_SHOULD_YIELD               1
97  #define configMAX_TASK_NAME_LEN               (10)
98
99  /* Software timer definitions. */
100 #define configUSE_TIMERS                      1
101 #define configTIMER_TASK_PRIORITY             (2)
102 #define configTIMER_QUEUE_LENGTH              5
103 #define configTIMER_TASK_STACK_DEPTH          (configMINIMAL_STACK_SIZE * 2)
104
105 #define configUSE_MUTEXES                     1
106 #define configUSE_RECURSIVE_MUTEXES           1
107 #define configUSE_COUNTING_SEMAPHORES         1
108 #define configUSE_QUEUE_SETS                  1
109 #define configUSE_IDLE_HOOK                   0
110 #define configUSE_TICK_HOOK                   0
111 #define configUSE_MALLOC_FAILED_HOOK          0
112 #define configUSE_16_BIT_TICKS                0
113
114 /* Set the following definitions to 1 to include the API function, or zero
115 to exclude the API function. */
116 #define INCLUDE_vTaskPrioritySet              1
117 #define INCLUDE_uxTaskPriorityGet             1
118 #define INCLUDE_vTaskDelete                   1
119 #define INCLUDE_vTaskSuspend                  1
120 #define INCLUDE_vTaskDelayUntil               1
121 #define INCLUDE_vTaskDelay                    1
122 #define INCLUDE_eTaskGetState                 1
123
124 /* Cortex-M specific definitions. */
125 #ifdef __NVIC_PRIO_BITS
126   /* __BVIC_PRIO_BITS will be specified when CMSIS is being used. */
127   #define configPRIO_BITS                     __NVIC_PRIO_BITS
128 #else
129   /* 15 priority levels */
130   #define configPRIO_BITS                     4
131 #endif
132
133 /* The lowest interrupt priority that can be used in a call to a "set priority"
134 function. */
135 #define configLIBRARY_LOWEST_INTERRUPT_PRIORITY       0xff
136
137 /* The highest interrupt priority that can be used by any interrupt service
138 routine that makes calls to interrupt safe FreeRTOS API functions.  DO NOT CALL
139 INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A HIGHER
140 PRIORITY THAN THIS! (higher priorities are lower numeric values. */
141 #define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY  10
142
143 /* Interrupt priorities used by the kernel port layer itself.  These are generic
144 to all Cortex-M ports, and do not rely on any particular library functions. */
145 #define configKERNEL_INTERRUPT_PRIORITY               (configLIBRARY_LOWEST_INTERRUPT_PRIORITY <<
146 /* !!!! configMAX_SYSCALL_INTERRUPT_PRIORITY must not be set to zero !!!!
147 See http://www.FreeRTOS.org/RTOS-Cortex-M3-M4.html. */
148 #define configMAX_SYSCALL_INTERRUPT_PRIORITY          (configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORIT
149
150 /* Normal assert() semantics without relying on the provision of an assert.h
151 header file. */
152 #define configASSERT( x ) if( ( x ) == 0 ) { taskDISABLE_INTERRUPTS(); for( ;; ); }
153
154 /* Map the FreeRTOS port interrupt handlers to their CMSIS standard names. */
155 #define xPortPendSVHandler                    PendSV_Handler
156 #define vPortSVCHandler                       SVC_Handler
157 #define xPortSysTickHandler                   SysTick_Handler
158
159 /* Include debug event definitions */
160 #include "freertos_evr.h"
161
```

```
162 #endif /* FREERTOS_CONFIG_H */
163
```

Interrupt priority configuration

FreeRTOS implements critical sections using the BASEPRI register (available in Armv7-M and Armv8-M architecture based devices) which masks only a subset of interrupts. This is configured via the `configMAX_SYSCALL_INTERRUPT_PRIORITY` setting. Therefore, it is needed to properly configure this setting. It is also needed to set appropriate interrupt priorities for interrupt service routines (ISR) that use RTOS functions. This can especially impact drivers which typically use peripheral interrupts. Normally, these use the RTOS directly or indirectly through registered callbacks.

Arm Cortex-M cores store interrupt priority values in the most significant bits of the interrupt priority registers which can have a maximum of eight bits. Many implementations offer only three priority bits. These three bits are shifted up to be bits five, six and seven respectively. `configMAX_SYSCALL_INTERRUPT_PRIORITY` must not be 0 and can be xxx00000. This results in the following table:

configMAX_SYSCALL_INTERRUPT_PRIORITY	Upper three bits	Priority
32	001	1 (Highest)
64	010	2
96	011	3
128	100	4
160	101	5
196	110	6
224	111	7 (Lowest)

Example

If you set `configMAX_SYSCALL_INTERRUPT_PRIORITY` to 32, then the priority of an interrupt service routine that uses RTOS functions must then be higher or equal to 1. This ensures that this interrupt will be masked during critical a section.

A WiFi driver using the SPI interface registers a callback to SPI which is executed in an interrupt context. The callback function in the WiFi driver uses RTOS functions. Therefore, the SPI interrupt priority must be set to a value equal or higher to the FreeRTOS preempt priority, for example 1.

Note

For a detailed description of how FreeRTOS is using Cortex-M code registers, refer to Running the RTOS on a ARM Cortex-M Core.

Add Event Recorder Visibility

- To use the Event Recorder together with FreeRTOS, add the software component **::Compiler:Event Recorder** to your project.
- Open **FreeRTOSConfig.h** and
  - verify the header file **freertos_evr.h** is included
  - add Event Recorder configuration definitions (see **Configure Event Recorder**)
- Call **EvrFreeRTOSSetup()** in your application code (ideally in `main()`).
- If you are using simulation mode, add an initialization file with the following content:

```
MAP 0xE0001000, 0xE0001007 READ WRITE
signal void DWT_CYCCNT (void) {
while (1) {
  rwatch(0xE0001004);
  _WWORD(0xE0001004, states);
  }
}
DWT_CYCCNT()
```

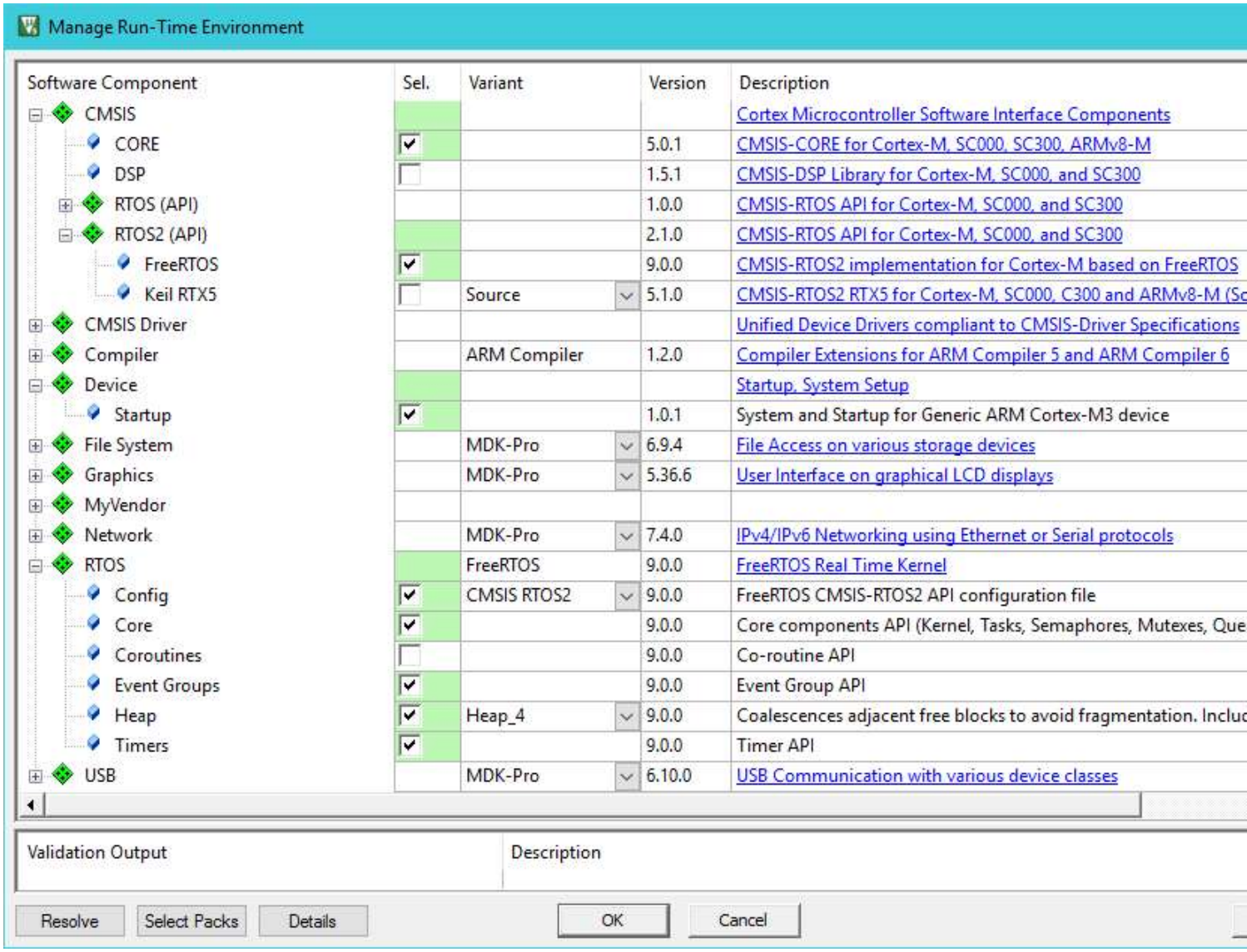
- Build the application code and download it to the debug hardware or run it in simulation.

Once the target application generates event information, it can be viewed in the ÂµVision debugger using the **Event Recorder**.

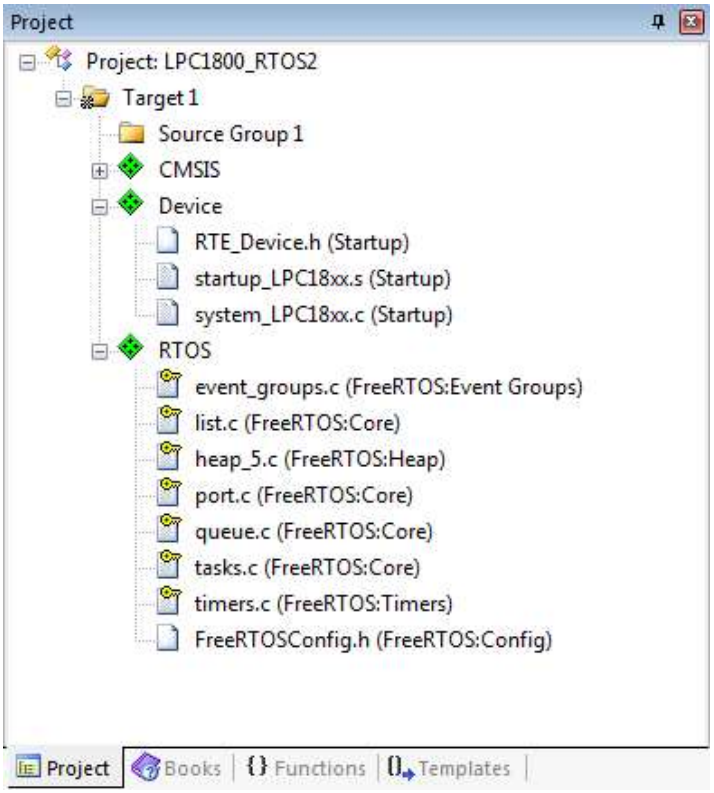
Create a CMSIS-FreeRTOS project

The steps to create a microcontroller application using CMSIS-FreeRTOS are:

- Create a new project and select a microcontroller device.
- In the Manage Run-Time Environment window, select **::Device:Startup**, **::CMSIS::RTOS2 (API)::FreeRTOS**, **::RTOS:CORE** in the **FreeRTOS** variant, **::RTOS:Config** in the **CMSIS RTOS2** variant, **::RTOS:Timers**, **::RTOS:Event Groups**, and an applicable **::RTOS:Heap** scheme (for more information on the heap schemes, visit the FreeRTOS documentation):

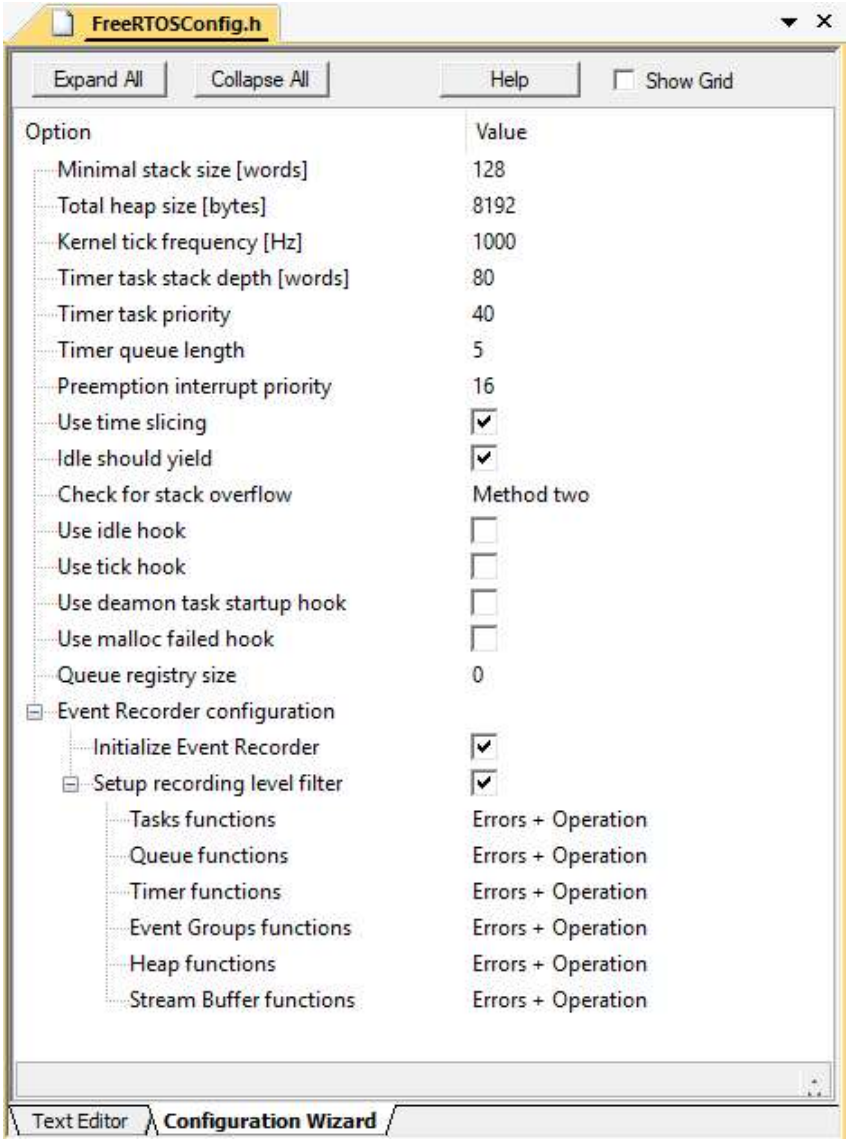


- If the **Validation Output** requires other components to be present, try to use the **Resolve** button.
- Click **OK**. In the **Project** window, you will see the files that have been automatically added to you project, such as **FreeRTOSConfig.h**, the source code files, as well as the system and startup files:



Configure CMSIS-FreeRTOS

When you have created the CMSIS-FreeRTOS project, you can configure the real-time operating system using the **FreeRTOSConfig.h** file. It can be opened using the Configuration Wizard view:



The following settings are available:

Name	#define	Description
Minimal stack size [words]	configMINIMAL_STACK_SIZE	Stack for idle task and default task stack in words.
Total heap size [bytes]	configTOTAL_HEAP_SIZE	Heap memory size in bytes.
Kernel tick frequency [Hz]	configTICK_RATE_HZ	Kernel tick rate in Hz.
Timer task stack depth [words]	configTIMER_TASK_STACK_DEPTH	Stack for timer task in words.
Timer task priority	configTIMER_TASK_PRIORITY	Timer task priority.
Timer queue length	configTIMER_QUEUE_LENGTH	Timer command queue length.
Preemption interrupt priority	configMAX_SYSCALL_INTERRUPT_PRIORITY	Maximum priority of interrupts that are safe to call FreeRTOS API.
Use time slicing	configUSE_TIME_SLICING	Enable setting to use time slicing.
Idle should yield	configIDLE_SHOULD_YIELD	Control Yield behavior of the idle task.
Check for stack overflow	configCHECK_FOR_STACK_OVERFLOW	Enable or disable stack overflow checking.
Use idle hook	configUSE_IDLE_HOOK	Enable callback function call on each idle task iteration.
Use tick hook	configUSE_TICK_HOOK	Enable callback function call during each tick interrupt.

Use daemon task startup hook	configUSE_DAEMON_TASK_STARTUP_HOOK	Enable callback function call when timer service starts.
Use malloc failed hook	configUSE_MALLOC_FAILED_HOOK	Enable callback function call when out of dynamic memory.
Queue registry size	configQUEUE_REGISTRY_SIZE	Define maximum number of queue objects registered for debug purposes.

Note
Refer to **Interrupt priority configuration** for more information on the usage of `configMAX_SYSCALL_INTERRUPT_PRIORITY`.

Event Recorder Configuration

The following settings are available (see **Configure Event Recorder** for details):

Name	#define	Description
Initialize Event Recorder	configEVR_INITIALIZE	Initialize Event Recorder before FreeRTOS kernel start.
Setup recording level filter	configEVR_SETUP_LEVEL	Enable configuration of FreeRTOS events recording level.
Task functions	configEVR_LEVEL_TASKS	Define event recording level bitmask for events generated from Tasks functions.
Queue functions	configEVR_LEVEL_QUEUE	Define event recording level bitmask for events generated from Queue functions.
Timer functions	configEVR_LEVEL_TIMERS	Define event recording level bitmask for events generated from Timer functions.
Event Groups functions	configEVR_LEVEL_EVENTGROUPS	Define event recording level bitmask for events generated from Event Groups functions.
Heap functions	configEVR_LEVEL_HEAP	Define event recording level bitmask for events generated from Heap functions.
Stream Buffer functions	configEVR_LEVEL_STREAMBUFFER	Define event recording level bitmask for events generated from Stream Buffer functions.

Add Event Recorder Visibility

- To use the Event Recorder together with FreeRTOS, add the software component **::Compiler:Event Recorder** to your project.
- Open **FreeRTOSConfig.h** and
  - verify the header file **freertos_evr.h** is included
  - modify Event Recorder configuration definitions (see **Configure Event Recorder**) to change default configuration
- Call **osKernelInitialize()** in your application code (ideally in `main()`) to setup Event Recorder according to configuration settings.
- If you are using simulation mode, add an initialization file with the following content:

```
MAP 0xE0001000, 0xE0001007 READ WRITE
signal void DWT_CYCCNT (void) {
while (1) {
  rwatch(0xE0001004);
  _WWORD(0xE0001004, states);
  }
}
DWT_CYCCNT()
```

- Build the application code and download it to the debug hardware or run it in simulation.

Once the target application generates event information, it can be viewed in the ÂµVision debugger using the **Event Recorder.**

Create a mixed-interface project

Using CMSIS-RTOS2 API and native FreeRTOS API simultaneously is possible and some projects do require using the native FreeRTOS API and the CMSIS-RTOS2 API at the same time. Such project should be **created as CMSIS-FreeRTOS project**.

Depending on the application requirements, FreeRTOS kernel can be started either by using FreeRTOS native API or by using CMSIS-RTOS2 API.

Start the kernel using CMSIS-RTOS2 API

```
/*
  Application thread: Initialize and start the Application
*/
void app_main (void *argument) {
  while(1) {
    // Application code
```

```
    // ...
  }
}
/*
  Main function: Initialize and start the kernel
*/
int main (void) {
  SystemCoreClockUpdate();

  // Initialize CMSIS-RTOS2
  osKernelInitialize();

  // Create application main thread
  osThreadNew(app_main, NULL, NULL);

  // Start the kernel and execute the first thread
  osKernelStart();

  while(1);
}
```

Restrictions

After the kernel is started using CMSIS-RTOS2 API, FreeRTOS native API can be used with the following restrictions:

- vTaskStartScheduler must not be called

Start the kernel using native API

```
/*
  Application main thread: Initialize and start the application
*/
void app_main (void *argument) {
  while(1) {
    // Application code
    // ...
  }
}
/*
  Main function: Initialize and start the kernel
*/
int main (void) {
  SystemCoreClockUpdate();

  // Setup the Event Recorder (optionally)
  EvrFreeRTOSSetup(0);

  // Create application main thread
  xTaskCreate (app_main, "app_main", 64, NULL, tskIDLE_PRIORITY+1, NULL);

  // Start the kernel and execute the first thread
  vTaskStartScheduler();

  while(1);
}
```

Restrictions

After the kernel is started using FreeRTOS native API, CMSIS-RTOS2 API can be used without restrictions.

Configure Event Recorder

This section describes the configuration settings for the Event Recorder annotations. For more information refer to section **Add Event Recorder Visibility to native FreeRTOS project** or **Add Event Recorder Visibility to CMSIS-FreeRTOS project**.

Use below definitions to configure Event Recorder initialization and recording level filter setup.

```
#define configEVR_INITIALIZE
```

Value	Description
0	Disable Event Recorder initialization
1	Enable Event Recorder initialization

Definition configEVR_INITIALIZE enables Event Recorder initialization during execution of function **EvrFreeRTOSSetup**. Default value is *1*.

```
#define configEVR_SETUP_LEVEL
```

Value	Description
0	Disable recording level filter setup
1	Enable recording level filter setup

Definition configEVR_SETUP_LEVEL enables setup of recording level filter for events generated by FreeRTOS. Recording level is configured during execution of function **EvrFreeRTOSSetup**. Default value is *1*.

```
#define configEVR_LEVEL_TASKS
#define configEVR_LEVEL_QUEUE
#define configEVR_LEVEL_TIMERS
#define configEVR_LEVEL_EVENTGROUPS
#define configEVR_LEVEL_HEAP
#define configEVR_LEVEL_STREAMBUFFER
```

Value	Description
0x00	Disable event generation
0x01	Enable generation of error events
0x05	Enable generation of error and operational events
0x0F	Enable generation of all events

Definitions configEVR_LEVEL_x set the recording level bitmask for events generated by each function group. They are taken into account only when recording level filter setup is enabled. Default value is *0x05*.

Debug a CMSIS-FreeRTOS project

Note

The following only applies when used with Arm Keil MDK. If you are using a different toolchain, please consult its user's manual.

Apart from the debug capabilities that **Event Recorder** offers, Keil MDK also supports thread aware breakpoints, just like for the standard CMSIS-RTOS.

Code Example

```
BS FuncN1, 1, "break = (CURR_TID == tid_phaseA) ? 1 : 0"
BS FuncN1, 1, "break = (CURR_TID == tid_phaseA || CURR_TID == tid_phaseD) ? 1 : 0"
BS \\Blinky\Blinky.c\FuncN1\179, 1, "break = (CURR_TID == tid_phaseA || CURR_TID == tid_phaseD) ? 1 : 0"
```

Note

- For more information on conditional breakpoints in Keil MDK, consult the user's manual.
- Enabling Periodic Window Update is required to capture register values for active running threads while executing. When turned off, only the current FreeRTOS thread can be unwound after execution has been stopped.

Caveats

- You cannot specify individual breakpoints on the same address. The following is not possible:

  ```
  BS ThCallee, 1, "break = (CURR_TID==tid_phaseA) ? 1 : 0"
  BS ThCallee, 1, "break = (CURR_TID==tid_phaseD) ? 1 : 0"
  ```

  Instead, use this:

  ```
  BS ThCallee, 1, "break= (CURR_TID==tid_phaseA || CURR_TID==tid_phaseD) ? 1 : 0"
  ```

- If you don't want to use Periodic Window Update, obtain the thread and unwind information by adding a function that gets called from each thread of interest:

  ```
  __attribute__((noinline)) int FuncN1 (int n1) {
    ...
  }
  ```

  Then, specify a thread aware breakpoint using an "invalid" thread ID:

  ```
  BS FuncN1, 1, "break = (CURR_TID == tid_phaseA + 1) ? 1 : 0"
  ```

  `'tid_phaseA'` would be valid, `'tid_phaseA + 1'` is not but will still capture the most recent registers and store them to the actual thread context each time a thread aware breakpoint is checked.
- Function inlining typically causes thread aware breakpoints to fail. To avoid this, prepend the `'noinline'` attribute to the function that is used to stop when the current FreeRTOS thread id matches:

  ```
  __attribute__((noinline)) int FuncN1 (int n1) {
    ...
  }
  ```

  This helps to make thread aware breakpoints far less dependent on the compiler optimization level.
- Thread aware breakpoints should be setup using a debug script. Reason being that thread aware breakpoints are of a 'hybrid' type, that is a combined address and condition expression that works best when run from a debug script.