## Exercise 1 A first CMSIS-RTOS2 project

This project will take you through the steps necessary to create and debug a CMSIS-RTOS2 based project.

**Open the Run Time Environment (RTE) by selecting the green diamond on the toolbar**

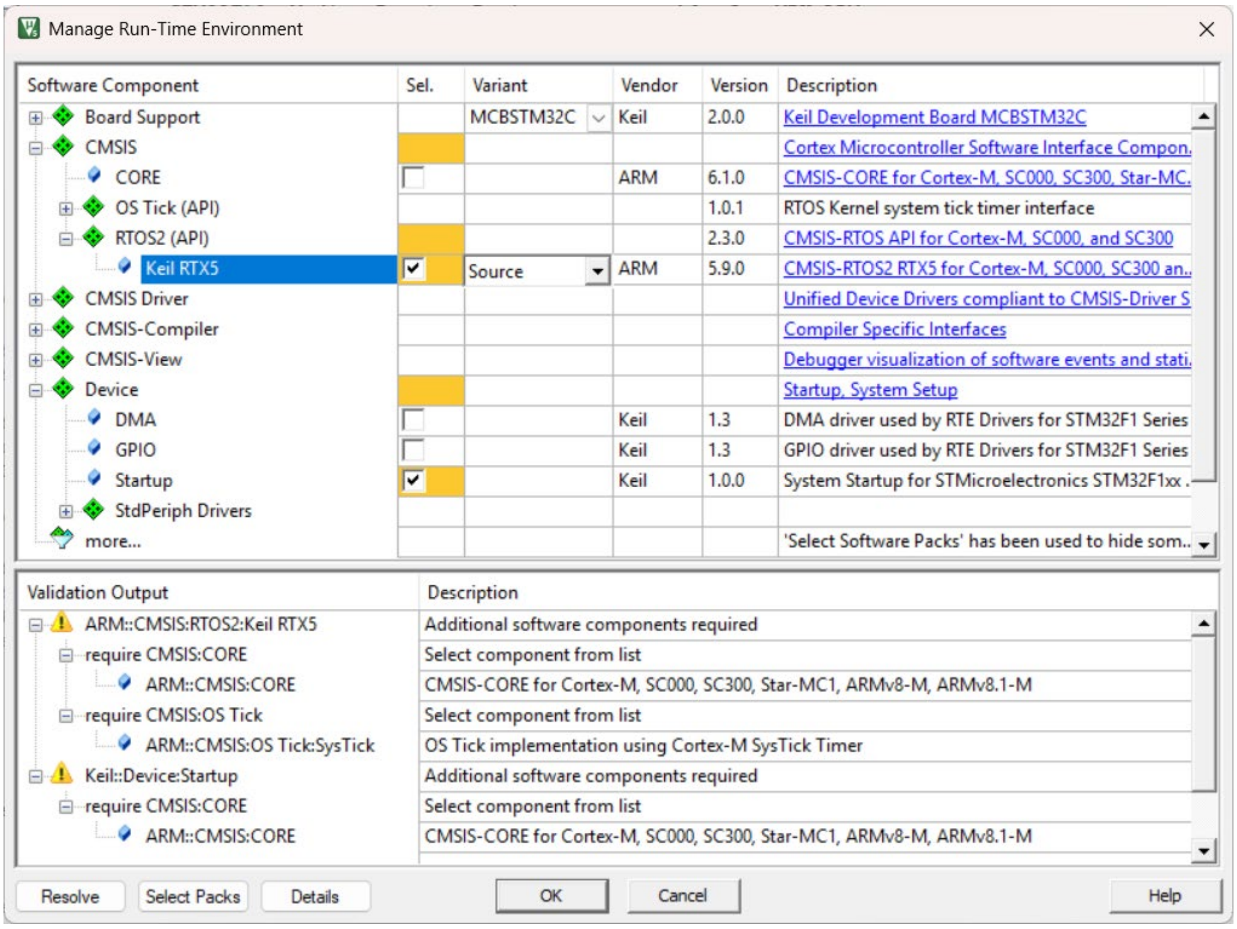
The RTE allows you to configure the platform of software components you are going to use in a given project. As well as displaying the available components the RTE understands their dependencies on other components.

**To configure the project for use with the CMSIS-RTOS2 Keil RTX, simply tick the CMSIS::RTOS2 (API):Keil RTX5 box.**

**Switch the Keil RTX5 dropdown variant box from 'Library' to 'Source'.**

**Also make sure to tick Device::Startup.**

This will cause the selection box to turn orange meaning that additional components are required. The required component will be displayed in the Validation Output window.



Manage Run-Time Environment

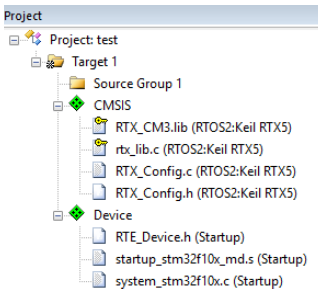
| Software Component | Sel. | Variant | Vendor | Version | Description |
|---|---|---|---|---|---|
| Board Support | | MCBSTM32C | Keil | 2.0.0 | Keil Development Board MCBSTM32C |
| CMSIS | 🟧 | | | | Cortex Microcontroller Software Interface Compon... |
| CORE | ☐ | | ARM | 6.1.0 | CMSIS-CORE for Cortex-M, SC000, SC300, Star-MC... |
| OS Tick (API) | | | | 1.0.1 | RTOS Kernel system tick timer interface |
| RTOS2 (API) | 🟧 | | | 2.3.0 | CMSIS-RTOS API for Cortex-M, SC000, and SC300 |
| Keil RTX5 | ✓ | Source | ARM | 5.9.0 | CMSIS-RTOS2 RTX5 for Cortex-M, SC000, SC300 an... |
| CMSIS Driver | | | | | Unified Device Drivers compliant to CMSIS-Driver S... |
| CMSIS-Compiler | | | | | Compiler Specific Interfaces |
| CMSIS-View | | | | | Debugger visualization of software events and stati... |
| Device | 🟧 | | | | Startup, System Setup |
| DMA | ☐ | | Keil | 1.3 | DMA driver used by RTE Drivers for STM32F1 Series |
| GPIO | ☐ | | Keil | 1.3 | GPIO driver used by RTE Drivers for STM32F1 Series |
| Startup | ✓ 🟧 | | Keil | 1.0.0 | System Startup for STMicroelectronics STM32F1xx ... |
| StdPeriph Drivers | | | | | |
| more... | | | | | 'Select Software Packs' has been used to hide som... |

| Validation Output | Description |
|---|---|
| ARM::CMSIS:RTOS2:Keil RTX5 | Additional software components required |
| require CMSIS:CORE | Select component from list |
| ARM::CMSIS:CORE | CMSIS-CORE for Cortex-M, SC000, SC300, Star-MC1, ARMv8-M, ARMv8.1-M |
| require CMSIS:OS Tick | Select component from list |
| ARM::CMSIS:OS Tick:SysTick | OS Tick implementation using Cortex-M SysTick Timer |
| Keil::Device:Startup | Additional software components required |
| require CMSIS:CORE | Select component from list |
| ARM::CMSIS:CORE | CMSIS-CORE for Cortex-M, SC000, SC300, Star-MC1, ARMv8-M, ARMv8.1-M |

**To add the missing components you can press the Resolve button in the bottom left hand corner of the RTE. When all the necessary components are present the selection column will turn green.**

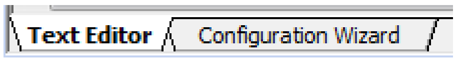
**Now press the OK button and all the selected components will be added to the new project**

**Open the Options for Target... by clicking the magic wand. Navigate to the tab 'C/C++ (AC6)' and choose Language C: c99. This is needed to build RTX5 from source.**

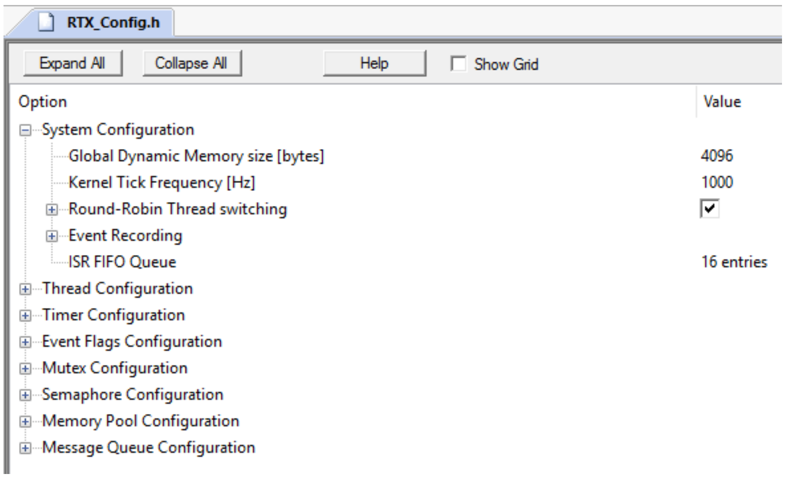

Project
Project: test
Target 1
Source Group 1
CMSIS
RTX_CM3.lib (RTOS2:Keil RTX5)
rtx_lib.c (RTOS2:Keil RTX5)
RTX_Config.c (RTOS2:Keil RTX5)
RTX_Config.h (RTOS2:Keil RTX5)
Device
RTE_Device.h (Startup)
startup_stm32f10x_md.s (Startup)
system_stm32f10x.c (Startup)

The CMSIS components are added to folders displayed as a green diamond. There are two types of file here. The first type is a library file which is held within the tool chain and is not editable. This file is shown with a yellow key to show that it is ‘locked’ (read-only). The second type of file is a configuration file. These files are copied to your project directory and can be edited as necessary. Each of these files can be displayed as a text files but it is also possible to view the configuration options as a set of pick lists and drop down menus.

**To see this open the RTX_Config.h file and at the bottom of the editor window select the ‘Configuration Wizard’ tab.**



**Click on Expand All to see all of the configuration options as a graphical pick list:**



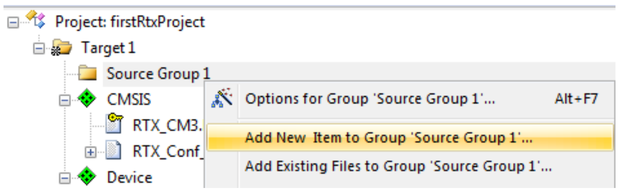
| Option | Value |
|---|---|
| System Configuration | |
| Global Dynamic Memory size [bytes] | 4096 |
| Kernel Tick Frequency [Hz] | 1000 |
| Round-Robin Thread switching | ✓ |
| Event Recording | |
| ISR FIFO Queue | 16 entries |
| Thread Configuration | |
| Timer Configuration | |
| Event Flags Configuration | |
| Mutex Configuration | |
| Semaphore Configuration | |
| Memory Pool Configuration | |
| Message Queue Configuration | |

Make the following changes:

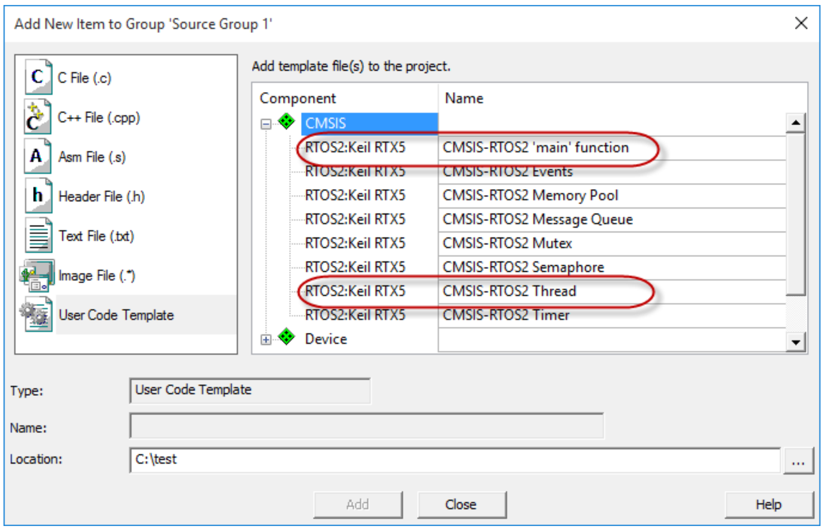
- **Global Dynamic Memory size [bytes]**: 4096
- **Default Thread Stack size [bytes]**: 512

Now that we have the basic platform for our project in place we can add some user source code which will start the RTOS and create a running thread.

**To do this right-click the 'Source Group 1' folder and select 'Add new item to Source Group 1'**


Project: firstRtxProject
Target 1
Source Group 1
CMSIS
RTX_CM3.
RTX_Conf_
Device
Options for Group 'Source Group 1'...
Alt+F7
Add New Item to Group 'Source Group 1'...
Add Existing Files to Group 'Source Group 1'...

**In the Add new Item dialog select the 'User code template' Icon and in the CMSIS section select the 'RTX5 'main' function' and click Add**

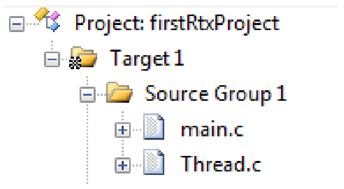


| Component | Name |
|---|---|
| CMSIS | |
| RTOS2:Keil RTX5 | CMSIS-RTOS2 'main' function |
| RTOS2:Keil RTX5 | CMSIS-RTOS2 Events |
| RTOS2:Keil RTX5 | CMSIS-RTOS2 Memory Pool |
| RTOS2:Keil RTX5 | CMSIS-RTOS2 Message Queue |
| RTOS2:Keil RTX5 | CMSIS-RTOS2 Mutex |
| RTOS2:Keil RTX5 | CMSIS-RTOS2 Semaphore |
| RTOS2:Keil RTX5 | CMSIS-RTOS2 Thread |
| RTOS2:Keil RTX5 | CMSIS-RTOS2 Timer |
| Device | |

Type: User Code Template
Name:
Location: C:\test

**Repeat this but this time select 'RTX5 Thread'.**

This will now add two source files to our project main.c and thread.c



**Open Thread.c in the editor**

We will look at the RTOS definitions in this project in the next section. For now this file contains two functions; Init_Thread() which is used to start the thread running, and the actual thread function.

**Then open main.c in the editor**

Main contains the functions to initialize and start the RTOS kernel. Then, unlike a bare metal project, main is allowed to terminate rather than enter an endless loop. However this is not really recommended and we will look at a more elegant way of terminating a thread later.

**In main.c add the Init_Thread prototype as an external declaration and then call it after the osKernelInitialize() function as shown below.**

```
/*----------------------------------------------------------------------
 * Application main thread
 *---------------------------------------------------------------------*/
__NO_RETURN static void app_main (void *argument) {
  (void)argument;
  extern int Init_Thread (void);
  Init_Thread ();
  for (;;) {}
}
```

**Build the project (F7)**

**Open the Options for Target... by clicking the magic wand. Navigate to the tab 'Debug' and choose Use Simulator.**

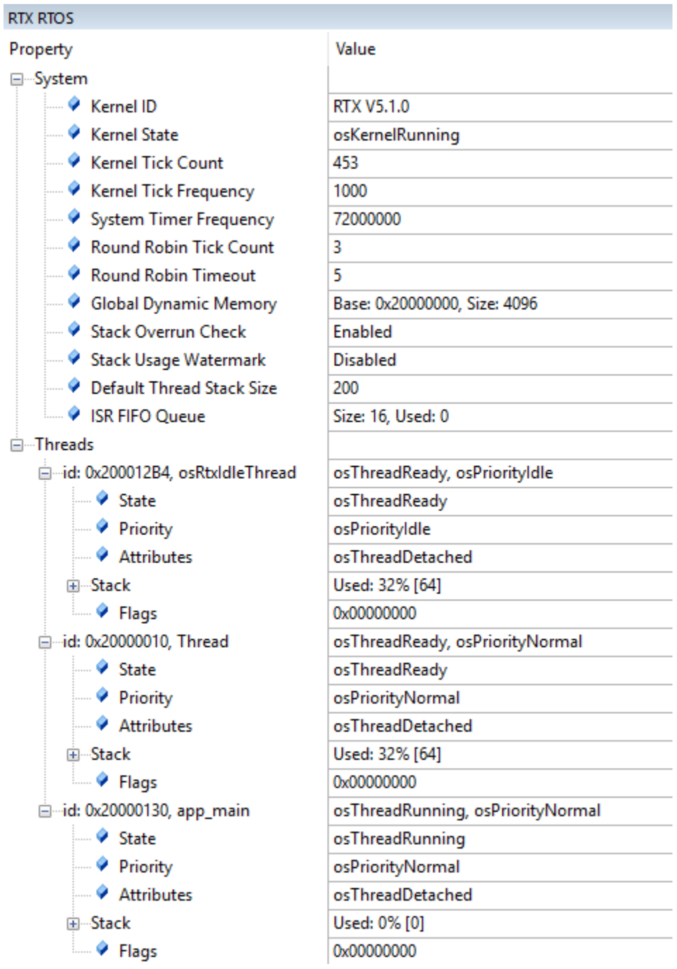
**Start the debugger (Ctrl+F5)**

This will run the code up to main

**Make sure View → Periodic Window Update is ticked**

**Open the → View → Watch Windows →RTX RTOS**

**Start the code running (F5)**



RTX RTOS

| Property | Value |
|---|---|
| System | |
| Kernel ID | RTX V5.1.0 |
| Kernel State | osKernelRunning |
| Kernel Tick Count | 453 |
| Kernel Tick Frequency | 1000 |
| System Timer Frequency | 72000000 |
| Round Robin Tick Count | 3 |
| Round Robin Timeout | 5 |
| Global Dynamic Memory | Base: 0x20000000, Size: 4096 |
| Stack Overrun Check | Enabled |
| Stack Usage Watermark | Disabled |
| Default Thread Stack Size | 200 |
| ISR FIFO Queue | Size: 16, Used: 0 |
| Threads | |
| id: 0x200012B4, osRtxIdleThread | osThreadReady, osPriorityIdle |
| State | osThreadReady |
| Priority | osPriorityIdle |
| Attributes | osThreadDetached |
| Stack | Used: 32% [64] |
| Flags | 0x00000000 |
| id: 0x20000010, Thread | osThreadReady, osPriorityNormal |
| State | osThreadReady |
| Priority | osPriorityNormal |
| Attributes | osThreadDetached |
| Stack | Used: 32% [64] |
| Flags | 0x00000000 |
| id: 0x20000130, app_main | osThreadRunning, osPriorityNormal |
| State | osThreadRunning |
| Priority | osPriorityNormal |
| Attributes | osThreadDetached |
| Stack | Used: 0% [0] |
| Flags | 0x00000000 |

This debug view shows all the running threads and their current state. At the moment we have three threads which are app_main, osRtxIdleThread and Thread.

This window is a component view which shows key variables in a software library (component). It is generated by an XML file. It is possible to create such a view for key variables in your application code. This is very useful if you have a long term project or code that you are going to give to a third party.

**Exit the debugger**

**While this project does not actually do anything it demonstrates the few steps necessary to start using CMSIS-RTOS2. As a last step, let's add some functionality.**

**Open Thread.c, and change the Thread definition as follows**

```
int x=0;

void Thread (void *argument) {
  while (1) {
    x++;
    osDelay(500);
  }
}
```

**Build the project again, then start a new Debugging session.**

There should now be one more thread; osRtxTimerThread. Add the variable `x` to a Watch window, and watch it increase over time. More on threads and timers in the next set of exercises.