"How to use MQX performance tool and TAD in MQX Lite"

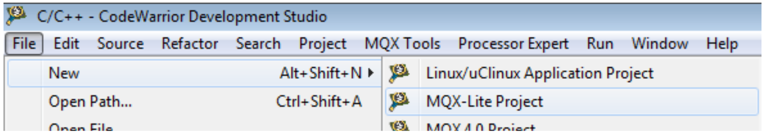
By Oralia Soledad Godínez Vega Aug, 2014

This document describes steps to configure and use the MQXLite Performance Tool to analyze and debug the applications of the Freescale MQXLite operating system.
The MQX Performance Tool is a set of small utilities and features (views) which can be used during a MQXLite debugger session in the CodeWarrior V10.6. The utilities are available in the MQX Tools menu in the CodeWarrior IDE.

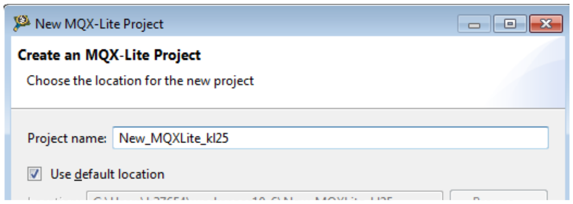
Creating a new MQXLite project

In order to create a new MQX Lite project select:

1. File -> New -> MQX-Lite Project



2. Give it a name and click Next.



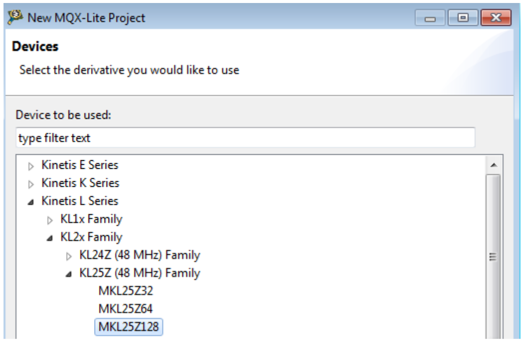
Create an MQX-Lite Project

Choose the location for the new project

Project name: New_MQXLite_kl25

☑ Use default location

3. Select the device, for this example the device is KL25Z.



Devices

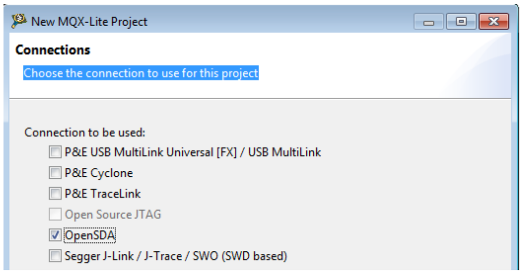
Select the derivative you would like to use

Device to be used:

type filter text

- Kinetis E Series
- Kinetis K Series
- Kinetis L Series
  - KL1x Family
  - KL2x Family
    - KL24Z (48 MHz) Family
    - KL25Z (48 MHz) Family
      - MKL25Z32
      - MKL25Z64
      - MKL25Z128

4. Choose the connection to use for this project.



New MQX-Lite Project

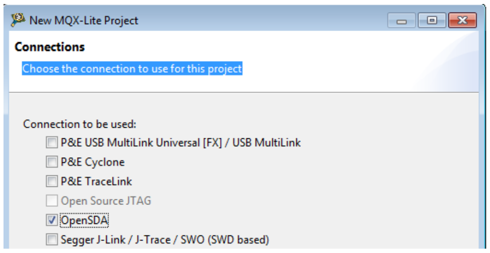
Connections

Choose the connection to use for this project

Connection to be used:

- ☐ P&E USB MultiLink Universal [FX] / USB MultiLink
- ☐ P&E Cyclone
- ☐ P&E TraceLink
- ☐ Open Source JTAG
- ☑ OpenSDA
- ☐ Segger J-Link / J-Trace / SWO (SWD based)

5. Choose the connection to use for this project and click Finish.



New MQX-Lite Project

Connections

Choose the connection to use for this project

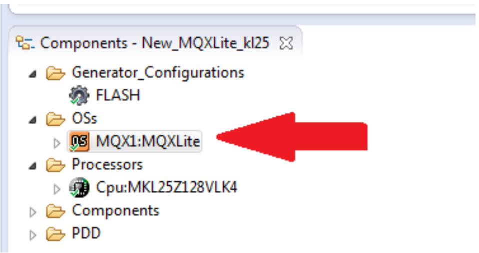
Connection to be used:

- ☐ P&E USB MultiLink Universal [FX] / USB MultiLink
- ☐ P&E Cyclone
- ☐ P&E TraceLink
- ☐ Open Source JTAG
- ☑ OpenSDA
- ☐ Segger J-Link / J-Trace / SWO (SWD based)

Creating new Tasks in MQX Lite

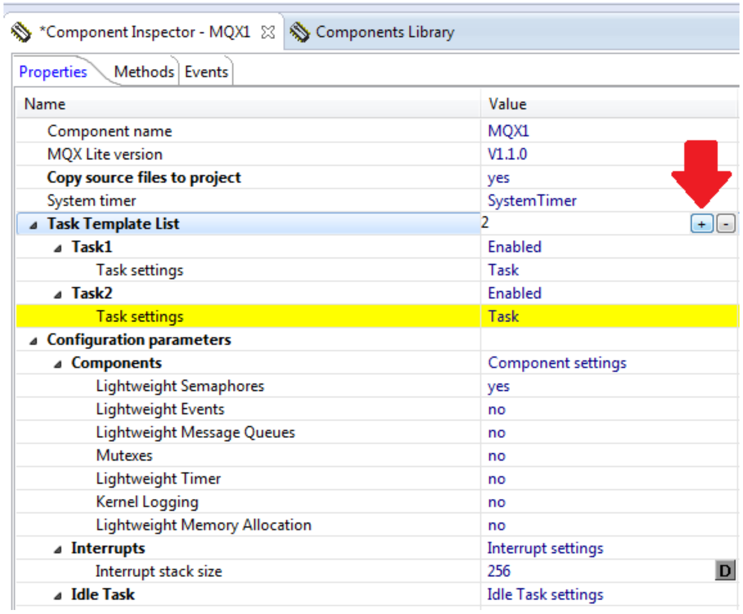
In order to create new Tasks in MQX Lite, it is necessary:

1. Select in Components view MQX1:MQXLite



```
Components - New_MQXLite_kl25
  Generator_Configurations
    FLASH
  OSs
    MQX1:MQXLite
  Processors
    Cpu:MKL25Z128VLK4
  Components
  PDD
```

2. After select MQX1:MQXLite in Component Inspector view, you can add the tasks desired.

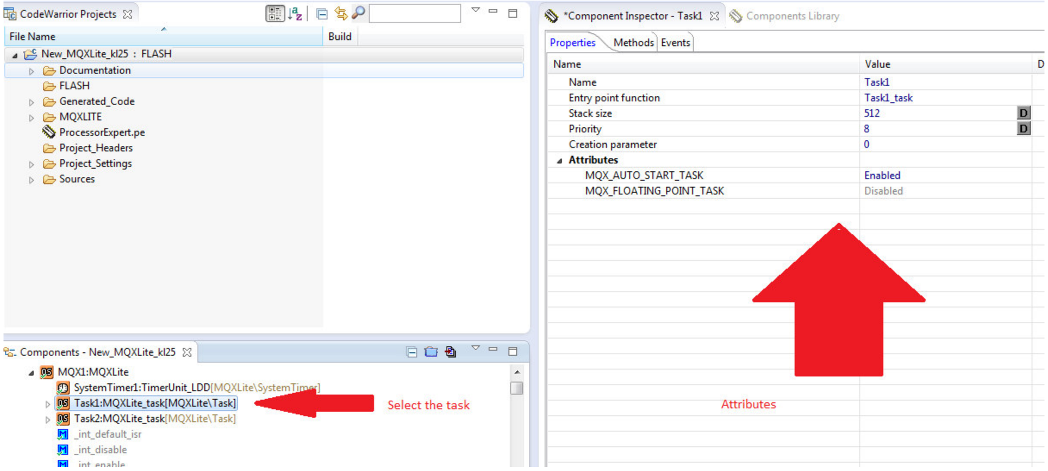


*Component Inspector - MQX1 | Components Library

Properties | Methods | Events

Name	Value
Component name	MQX1
MQX Lite version	V1.1.0
Copy source files to project	yes
System timer	SystemTimer
Task Template List	2
Task1	Enabled
Task settings	Task
Task2	Enabled
Task settings	Task
Configuration parameters	
Components	Component settings
Lightweight Semaphores	yes
Lightweight Events	no
Lightweight Message Queues	no
Mutexes	no
Lightweight Timer	no
Kernel Logging	no
Lightweight Memory Allocation	no
Interrupts	Interrupt settings
Interrupt stack size	256
Idle Task	Idle Task settings

3. Once the tasks are added, it is possible to modify the attributes of each task in the Component Inspector view. First select the task you want to modify.



CodeWarrior Projects

File Name	Build
New_MQXLite_kl25 : FLASH	
Documentation	
FLASH	
Generated_Code	
MQXLITE	
ProcessorExpert.pe	
Project_Headers	
Project_Settings	
Sources	

Components - New_MQXLite_kl25

- MQX1:MQXLite
  - SystemTimer1:TimerUnit_LDD[MQXLite\SystemTimer]
  - Task1:MQXLite_task[MQXLite\Task]
  - Task2:MQXLite_task[MQXLite\Task]
  - _int_default_isr
  - _int_disable
  - _int_enable

Select the task

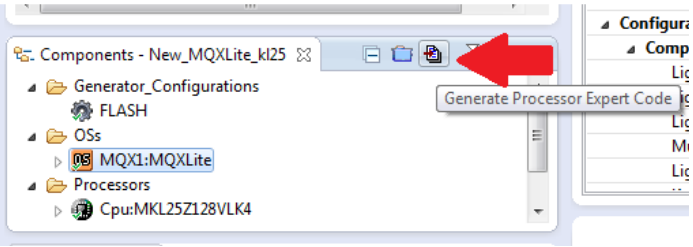
*Component Inspector - Task1 | Components Library

Properties | Methods | Events

Name	Value
Name	Task1
Entry point function	Task1_task
Stack size	512
Priority	8
Creation parameter	0
Attributes	
MQX_AUTO_START_TASK	Enabled
MQX_FLOATING_POINT_TASK	Disabled

Attributes

4. After set the desire configurations it is necessary to Generate Processor Expert Code.

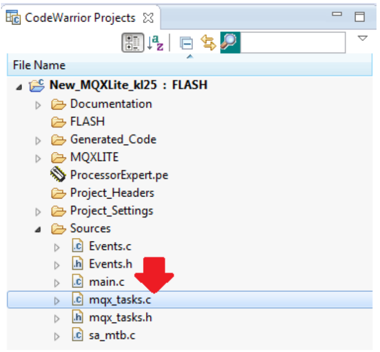


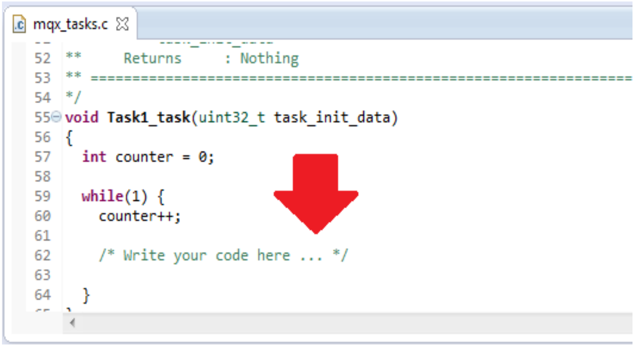
Components - New_MQXLite_kl25

- Generator_Configurations
  - FLASH
- OSs
  - MQX1:MQXLite
- Processors
  - Cpu:MKL25Z128VLK4

Generate Processor Expert Code

5. Subsequent to generate the code you can write the code for each task in the mqx_task.c file located at the source folder.





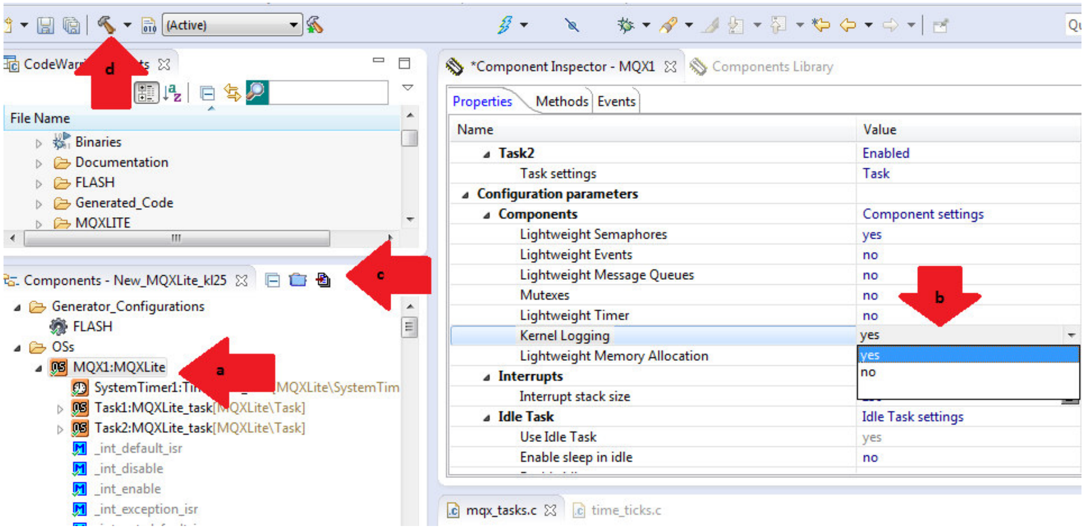
```
**     Returns     : Nothing
** ===================================================================
*/
void Task1_task(uint32_t task_init_data)
{
  int counter = 0;

  while(1) {
    counter++;

    /* Write your code here ... */

  }
```

6. Save, Build and Debug.

For this example, two tasks were created with the same attributes (auto-star), the same priority (8). The reason of this document is to utilize the TAD in MQX Lite, so it is used the default code for each task, only is added a blocking function in each task (_time_delay_ticks(1000)).

Steps to use the MQX performance tool in MQX Lite

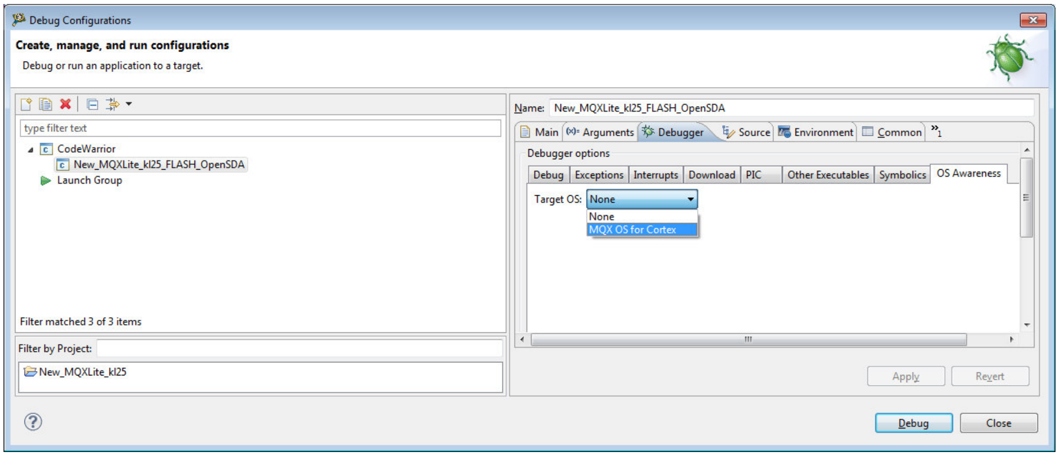
1. To use the tool, the MQX Lite application must have kernel logging enabled. In order to enable this, it is necessary: a) Select MQX1:MQXLite on Components view, b) Enable the Kernel Logging on Component Inspector view, c) Generated Processor Expert Code and d) Build the project.



Name	Value
Task2	Enabled
Task settings	Task
Configuration parameters	
Components	Component settings
Lightweight Semaphores	yes
Lightweight Events	no
Lightweight Message Queues	no
Mutexes	no
Lightweight Timer	no
Kernel Logging	yes
Lightweight Memory Allocation	
Interrupts	
Interrupt stack size	
Idle Task	Idle Task settings
Use Idle Task	yes
Enable sleep in idle	no

2. In CodeWarrior IDE, enable the option "MQX OS for Cortex". For do that go to Debug Configurations->Debugger->OS Awareness

and select on Target OS: MQX OS for Cortex; click Apply and debug the project.



Name: New_MQXLite_kl25_FLASH_OpenSDA

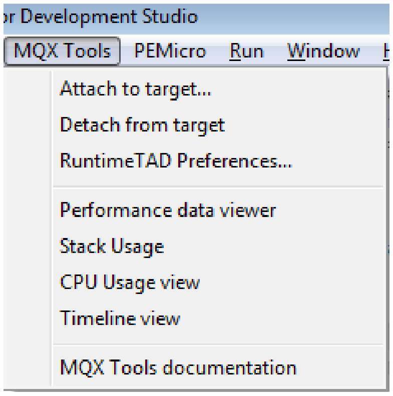
Target OS: None

3. At this point, you should be able to use the MQX performance tool.

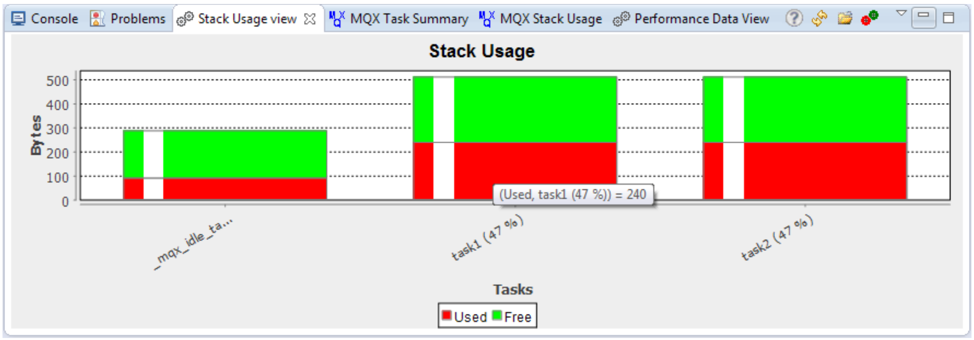
Using the Stack Usage View

It is necessary to follow the below steps:

1. Debug the application in CodeWarrior using the debug button.
2. To stop the application, use a breakpoint or use the suspend button.
3. In the MQX Tools menu select the Stack Usage View.



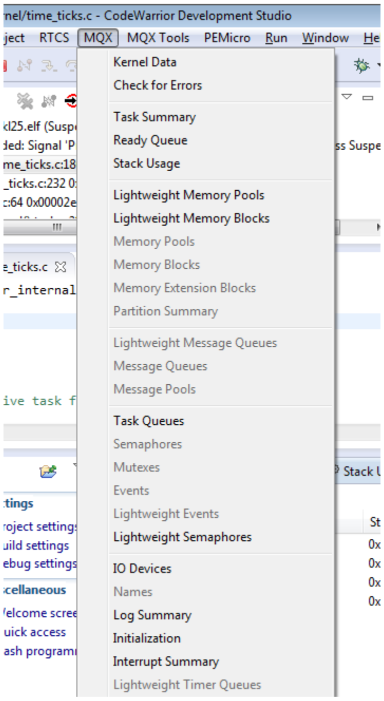
4. The Stack Usage opens in an offline mode by default with no data to be displayed.
5. Click the Toggle *offline* and *online* mode button to switch to online mode. The bar graph displays how much stack is used by each task.



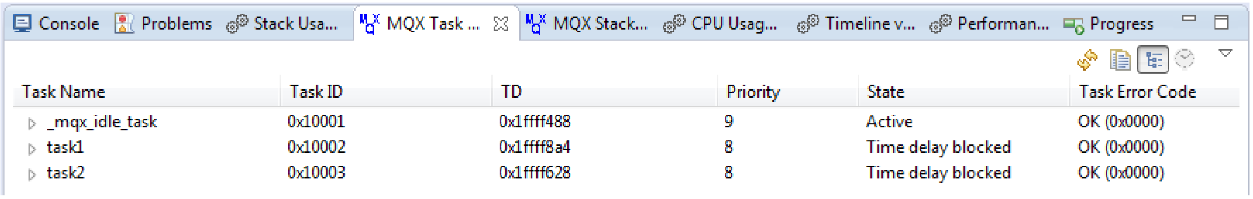
Using TAD Views

The Task Aware Debugging (TAD) Tool is a useful and powerful instrument to debug the MQX application.
In order to use TAD it is necessary to follow the below steps:

1. Debug the application in CodeWarrior using the debug button.
2. To stop the application, use a breakpoint or use the suspend button.
3. You can now use the MQX RTOS menu to open the various TAD views. Note that availability of individual menu items depends on the features used by the MQX application. The MQX menu shows information about the core and tasks running at that moment in MQX RTOS, information such as memory usage, stack usage, semaphores, mutexes, events, kernel data, errors, and so on.

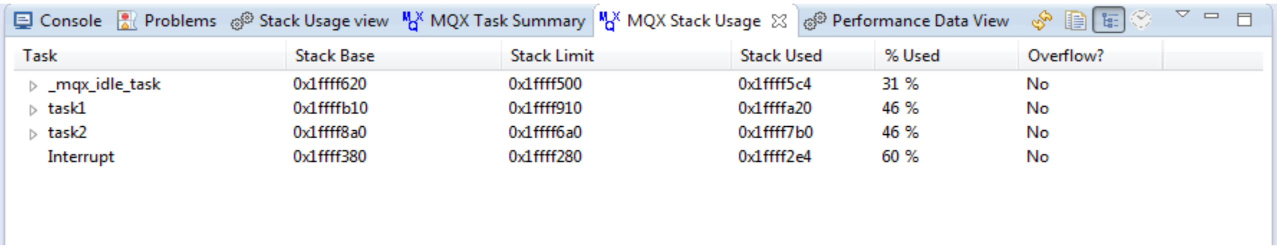


4. Go to the MQX menu and click on Task Summary option. If the connection is successfully established, the Task summary view will be opened. You should be able to see the MQX tasks running on the connected target.



Task Name	Task ID	TD	Priority	State	Task Error Code
_mqx_idle_task	0x10001	0x1ffff488	9	Active	OK (0x0000)
task1	0x10002	0x1ffff8a4	8	Time delay blocked	OK (0x0000)
task2	0x10003	0x1ffff628	8	Time delay blocked	OK (0x0000)

5. Similar as the Task Summary, the Stack Usage can be displayed and it shows the memory used by each task within its stack. It also shows if an overflow happens.



Task	Stack Base	Stack Limit	Stack Used	% Used	Overflow?
_mqx_idle_task	0x1ffff620	0x1ffff500	0x1ffff5c4	31 %	No
task1	0x1ffffb10	0x1ffff910	0x1ffffa20	46 %	No
task2	0x1ffff8a0	0x1ffff6a0	0x1ffff7b0	46 %	No
Interrupt	0x1ffff380	0x1ffff280	0x1ffff2e4	60 %	No

I hope this documents helps!!