MCUXpresso IDE FreeRTOS Debug Guide

Rev. 10.0.2 — 4 July, 2017 User guide

4 July, 2017

1. Introduction

Many of the examples provided as part of MCUXpresso SDK and LPCOpen packages are built around the *FreeRTOS* real time operating system. FreeRTOS is also a popular choice when developing MCU software applications for real products.

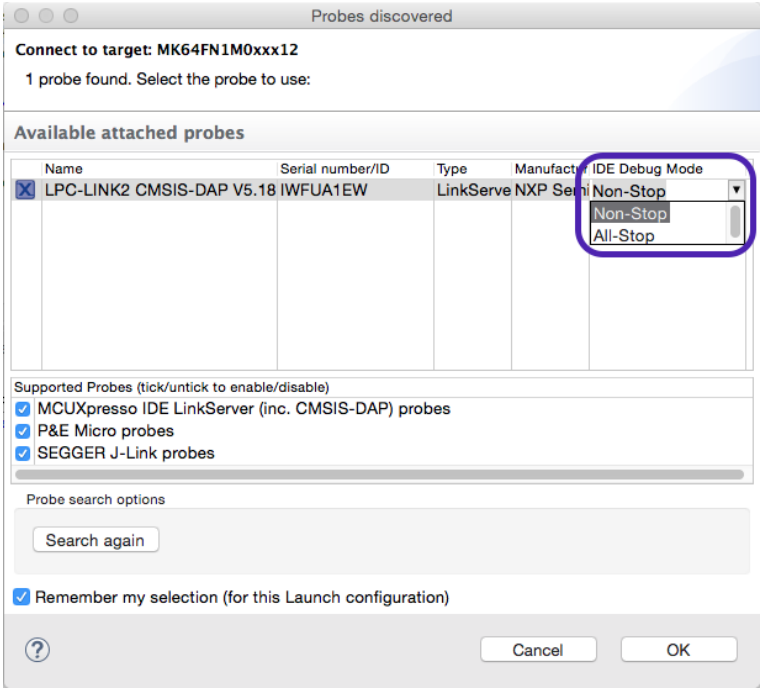
For more information on FreeRTOS please visit http://www.freertos.org

This guide examines some of the functionality included in MCUXpresso IDE to assist you in debugging applications built around FreeRTOS. It does not provide any information on FreeRTOS itself or on developing applications that use FreeRTOS.

2. LinkServer FreeRTOS Thread Aware Debugging

When debugging via LinkServer debug probes, the MCUXpresso IDE debugger can provide FreeRTOS thread aware debug if :

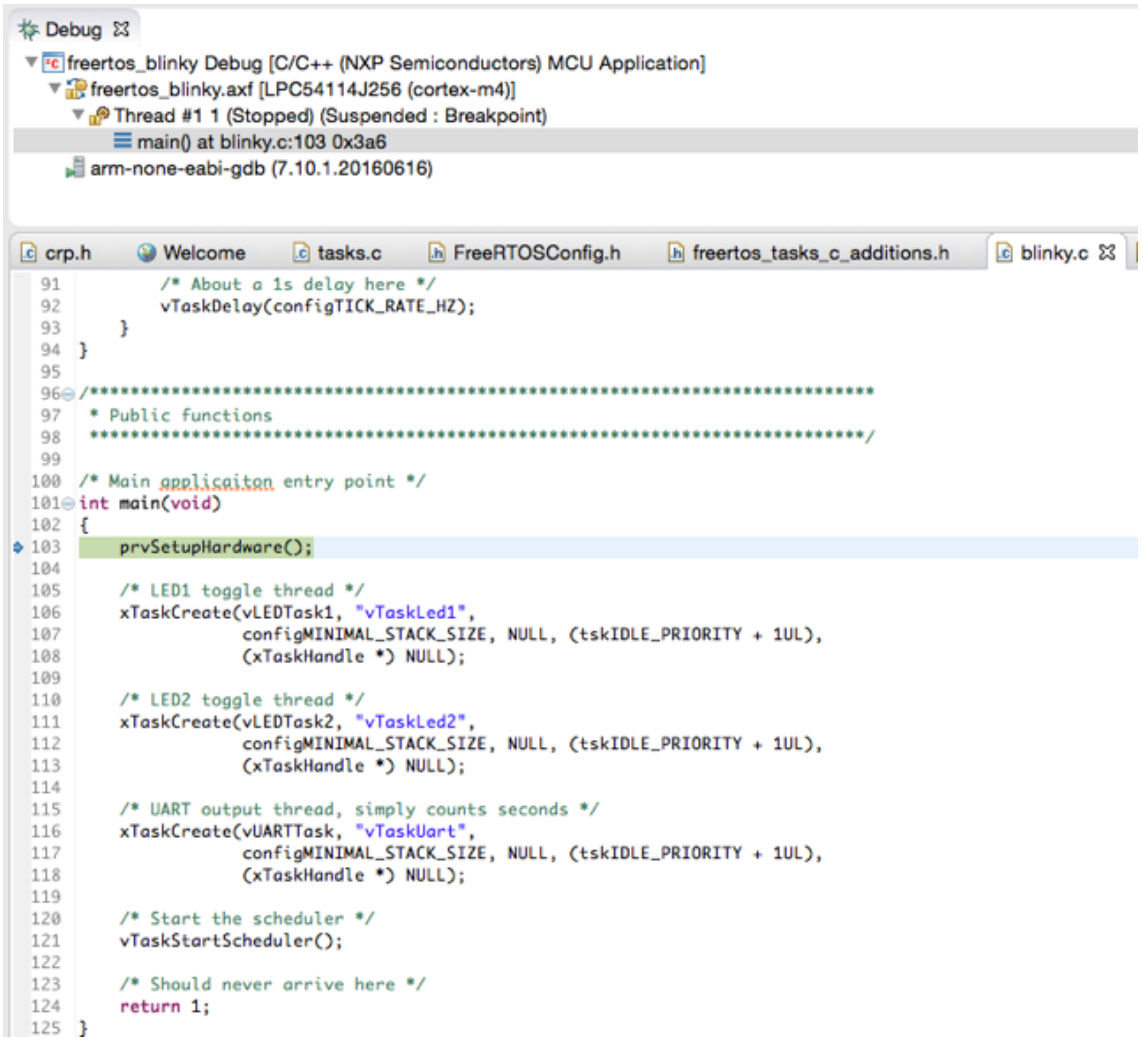
1. Minor modifications are made to the application, so that configuration information required by the debugger is present in the image file.
2. Debugging is carried out in All-stop mode (rather than the default Non-stop mode). This selection is made when first making a debug connection for a particular project (or after deleting an existing launch configuration). For more details, please see the MCUXpresso IDE User Guide.



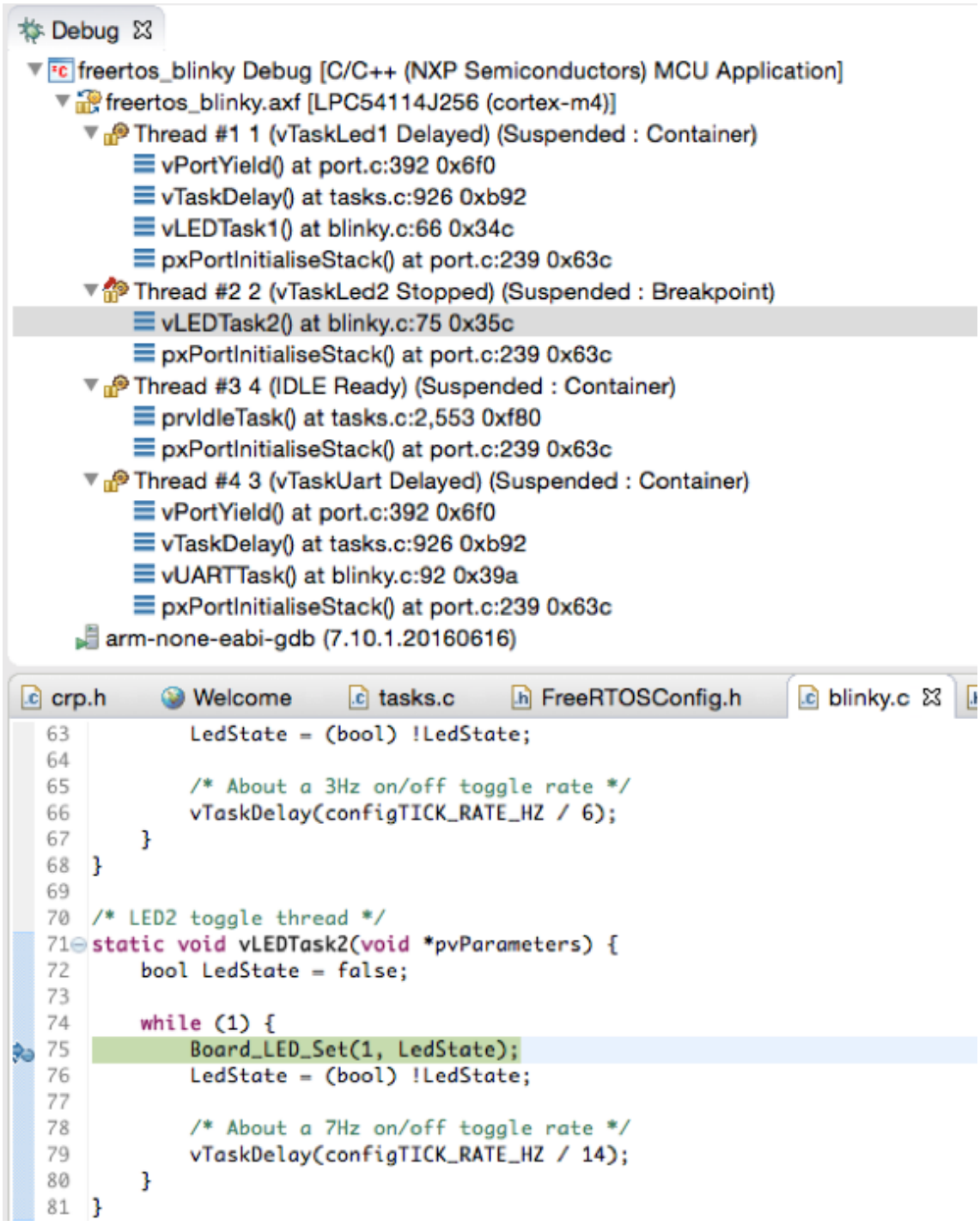
The source code modifications required are described in Required Source Code Changes [5].

Note: Example projects supplied as part of MCUXpresso IDE compatible SDK packages should already have had these changes made to them.

Without these changes, or if Non-stop debug mode is used, only the current thread will be seen in the Debug View, as shown in the below screenshot:



However, once the necessary changes are made to the application source, and All-stop debug mode is used, the Debug View will display each thread separately, as shown in the next screenshot:



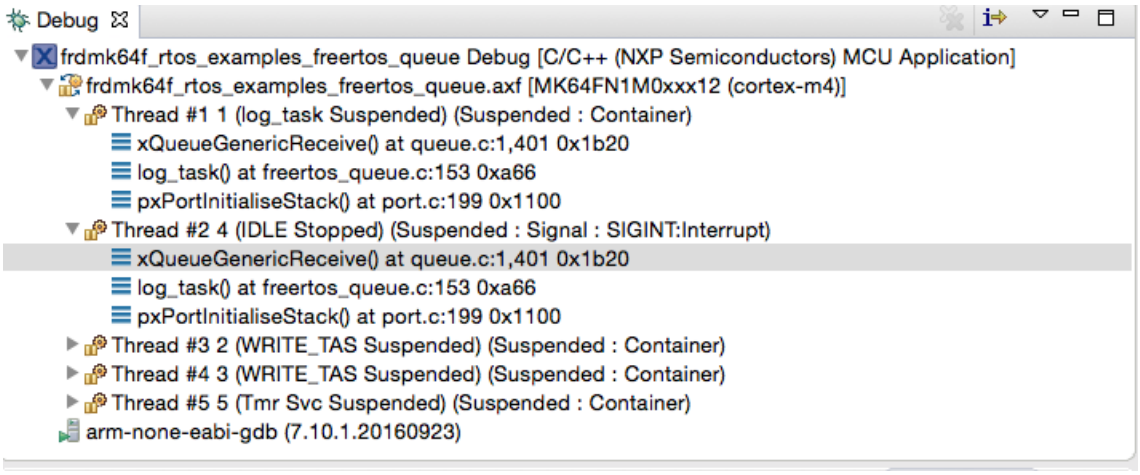
Note: MCUXpresso IDE v10.0.0 provided a limited form of thread aware debugging in Non-stop debug mode. However due to restrictions in the way GDB's non-stop debug functionality operates, this has been removed in MCUXpresso IDE v10.0.2 and later. This means that it is no longer possible to make use of Live Variables functionality within the Global Variables View at the same time as LinkServer FreeRTOS thread aware debug.

2.1 Behavior when thread aware debugging

MCUXpresso IDE LinkServer FreeRTOS thread aware debugging is available once the FreeRTOS scheduler has started (so will not appear straight after loading the application when the default breakpoint on `main()` is reached). Debug works in stop mode. In other words, if

execution of a user task is halted either through a user action (halt) or a debug event (breakpoint, watchpoint, fault, etc.), the stopped thread is current and no application thread executes in the background. The register context for any thread is available in the register window. For suspended or blocked threads, the register context is the context in effect when the thread was swapped out, regardless of which thread stack level is examined within the traceback window.

In the below example, the MCU is halted in Thread #2, but a backtrace for Thread #1 is also opened up (and backtrace information for Threads #3, #4, and #5 is also available):



2.2 Required Source Code Changes

MCUXpresso IDE debug is implemented via a GDB remote console application (i.e. a stub). A “remote debug stub” underneath GDB has access to symbolic information (through GDB), but has no direct knowledge of symbol data types. Thread aware debug for FreeRTOS requires 16 bytes of configuration data (symbol FreeRTOSDebugConfig) be added to the application to describe the as-built kernel configuration for a given FreeRTOS project.

The following notes describe the FreeRTOS project modifications required to enable thread aware debug.

Note: Example projects supplied as part of MCUXpresso IDE compatible SDK packages should already have had these changes made to them. And future releases of FreeRTOS are also expected to include the same changes. Thus these changes are generally only required for LPC preinstalled parts with LPCOpen FreeRTOS using projects.

2.2.1 Modify – File `tasks.c`

The MCUXpresso IDE FreeRTOS thread aware debug requires the addition of the following conditional include, and function definition, to the end of the `tasks.c` source file. This code can be placed after the `FREERTOS_MODULE_TEST` conditional include, if it exists:

```
#if ( configINCLUDE_FREERTOS_TASK_C_ADDITIONS_H == 1 )
    #include "freertos_tasks_c_additions.h"
    static void freertos_tasks_c_additions_init( void )
    {
        #ifdef FREERTOS_TASKS_C_ADDITIONS_INIT
            FREERTOS_TASKS_C_ADDITIONS_INIT();
        #endif
    }
#endif
```

Note that the function `freertos_tasks_c_additions_init()` will be called by `vTaskStartScheduler()` in future releases of FreeRTOS, but is not currently used by the MCUXpresso IDE.

2.2.2 Modify – File `FreeRTOSConfig.h`

The `FreeRTOSConfig.h` header file is included in the FreeRTOS source distribution. To enable a FreeRTOS project for thread aware debug, add the following macro definition to this file:

```
#define configINCLUDE_FREERTOS_TASK_C_ADDITIONS_H     1
```

Next, ensure the `configUSE_TRACE_FACILITY` macro is set to 1.

```
#define configUSE_TRACE_FACILITY     1
```

2.2.3 Create – New File `freertos_tasks_c_additions.h`

The text for the one additional header file, `freertos_tasks_c_additions.h`, can be constructed by a cut and paste of the text found below between the “cut here” lines.

For convenience, the `freertos_tasks_c_additions.h` header file you will create can be placed in the same folder as the `FreeRTOSConfig.h` header file.

There is also one edit to `freertos_tasks_c_additions.h` itself that may be required for a particular FreeRTOS project configuration. The macro `configFRTOS_MEMORY_SCHEME` describes the project heap mechanism using a value 1 – 5 according to the following:

1. **heap_1** : The very simplest; does not permit memory to be freed
2. **heap_2** : Permits memory to be freed, but not does coalesce adjacent free blocks
3. **heap_3** : Simply wraps the standard malloc() and free() for thread safety
4. **heap_4** : Coalesces adjacent free blocks to avoid fragmentation. Includes absolute address placement option
5. **heap_5** : As per heap_4, with the ability to span the heap across multiple non-adjacent memory areas

Note: Future versions of FreeRTOS may incorporate the `configFRTOS_MEMORY_SCHEME` macro as a configuration parameter in `FreeRTOSConfig.h`.

```
-------------------------------- cut here --------------------------------
// freertos_tasks_c_additions.h Rev. 1.2
//
#include <stdint.h>

#if (configUSE_TRACE_FACILITY == 0)
#error "configUSE_TRACE_FACILITY must be enabled"
#endif

#define FREERTOS_DEBUG_CONFIG_MAJOR_VERSION 1
#define FREERTOS_DEBUG_CONFIG_MINOR_VERSION 1

// NOTE!!
// Default to a FreeRTOS version which didn't include these macros. FreeRTOS
// v7.5.3 is used here.
#ifndef tskKERNEL_VERSION_BUILD
#define tskKERNEL_VERSION_BUILD 3
#endif
#ifndef tskKERNEL_VERSION_MINOR
#define tskKERNEL_VERSION_MINOR 5
```

```
#endif
#ifndef tskKERNEL_VERSION_MAJOR
#define tskKERNEL_VERSION_MAJOR 7
#endif

// NOTE!!
// The configFRTOS_MEMORY_SCHEME macro describes the heap scheme using a value
// 1 - 5 which corresponds to the following schemes:
//
// heap_1 - the very simplest, does not permit memory to be freed
// heap_2 - permits memory to be freed, but not does coalescence adjacent free
//          blocks.
// heap_3 - simply wraps the standard malloc() and free() for thread safety
// heap_4 - coalesces adjacent free blocks to avoid fragmentation. Includes
//          absolute address placement option
// heap_5 - as per heap_4, with the ability to span the heap across
//          multiple non-adjacent memory areas

#ifndef configFRTOS_MEMORY_SCHEME
#define configFRTOS_MEMORY_SCHEME     3 // thread safe malloc
#endif

#if ((configFRTOS_MEMORY_SCHEME > 5) || (configFRTOS_MEMORY_SCHEME < 1))
#error "Invalid configFRTOS_MEMORY_SCHEME setting!"
#endif

#ifdef __cplusplus
extern "C" {
#endif

extern const uint8_t FreeRTOSDebugConfig[];

// NOTES!!
// IAR documentation is confusing. It suggests the data must be statically
// linked, and the #pragma placed immediately before the symbol definition.
// The IAR supplied examples violate both "rules", so this is a best guess.
//
#if defined(__GNUC__)
const uint8_t FreeRTOSDebugConfig[] __attribute__((section(".rodata"))) =
#elif defined(__CC_ARM)
const uint8_t FreeRTOSDebugConfig[] __attribute__((used)) =
#elif defined(__IAR_SYSTEMS_ICC__)
#pragma required=FreeRTOSDebugConfig
const uint8_t FreeRTOSDebugConfig[] =
#endif
{
    FREERTOS_DEBUG_CONFIG_MAJOR_VERSION,
    FREERTOS_DEBUG_CONFIG_MINOR_VERSION,
    tskKERNEL_VERSION_MAJOR,
    tskKERNEL_VERSION_MINOR,
    tskKERNEL_VERSION_BUILD,
    configFRTOS_MEMORY_SCHEME,
    offsetof(struct tskTaskControlBlock, pxTopOfStack),
#   if (tskKERNEL_VERSION_MAJOR > 8)
    offsetof(struct tskTaskControlBlock, xStateListItem),
#   else
    offsetof(struct tskTaskControlBlock, xGenericListItem),
```

```
#   endif
    offsetof(struct tskTaskControlBlock, xEventListItem),
    offsetof(struct tskTaskControlBlock, pxStack),
    offsetof(struct tskTaskControlBlock, pcTaskName),
    offsetof(struct tskTaskControlBlock, uxTCBNumber),
    offsetof(struct tskTaskControlBlock, uxTaskNumber),
    configMAX_TASK_NAME_LEN,
    configMAX_PRIORITIES,
    0 // pad to 32-bit boundary
};

#ifdef __cplusplus
}
#endif

// end freertos_tasks_c_additions.h
---------------------------------- cut here ----------------------------------
```

3. FreeRTOS Task Aware Debug Views

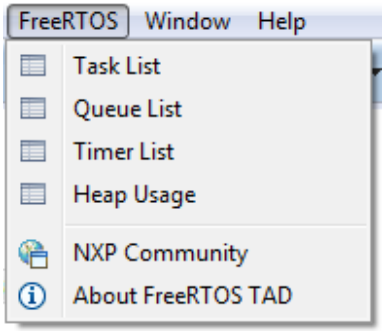
MCUXpresso IDE includes several additional Views to further simplify FreeRTOS application debugging, known collectively as the *FreeRTOS TAD* (Task Aware Debugger for GDB):

- **Task List** : shows list of tasks with status information
- **Queue List** : shows currently active queues, semaphore, and mutex
- **Timer List** : lists the RTOS software timers
- **Heap Usage** : shows current heap usage and memory block allocation

Note: These Views are independent of the debug probe being used, as they just use GDB commands to receive information from the target.

3.1 Showing the FreeRTOS TAD Views

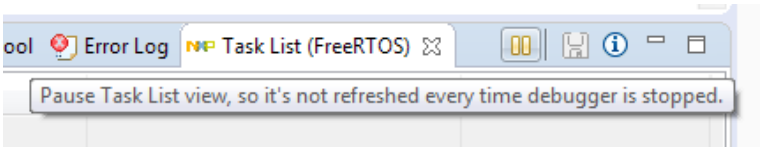
The FreeRTOS Views can be opened using the “FreeRTOS” main menu in the MCUXpresso IDE.



The Views are “stop mode” Views: with the target halted or stopped, the Views will query the device under debug and read the necessary information through the debug connection.

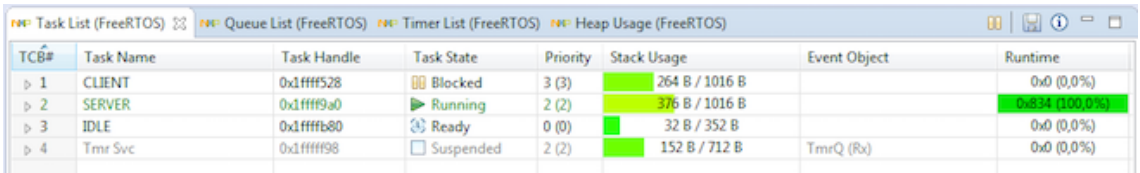
This will also happen during single stepping, so to improve stepping performance it is advisable to:

1. Only have the needed Views in the foreground/visible, or close the Views if they are not used.
2. Make use of the Pause View feature, allowing you to single step without the Views constantly reloading data.



3.2 Task List View

This View shows the tasks in a table:



TCB#

- Task Control Block. `configUSE_TRACE_FACILITY` needs to be set to 1

Task Name

- Name of task. `configMAX_TASK_NAME_LEN` needs to be greater than zero

Task Handle

- Address of the task handle

Task State

- Current task state: blocked, running, ready

Priority

- Task base priority and current task priority

Stack Usage

- Graphical view of current stack usage, with current allocation and stack size available to the task

Event Object

- Lists the object a blocked task is waiting for. Use `vQueueAddToRegistry()` to assign a symbolic name to semaphore, mutex, and queues with `configQUEUE_REGISTRY_SIZE` greater than zero

Runtime

- Task runtime with percentage value. Both `configUSE_TRACE_FACILITY` and `configGENERATE_RUN_TIME_STATS` need to be set to 1

Unfolding a task line item shows the following items:

Stack base

- Stack start address

Stack Top

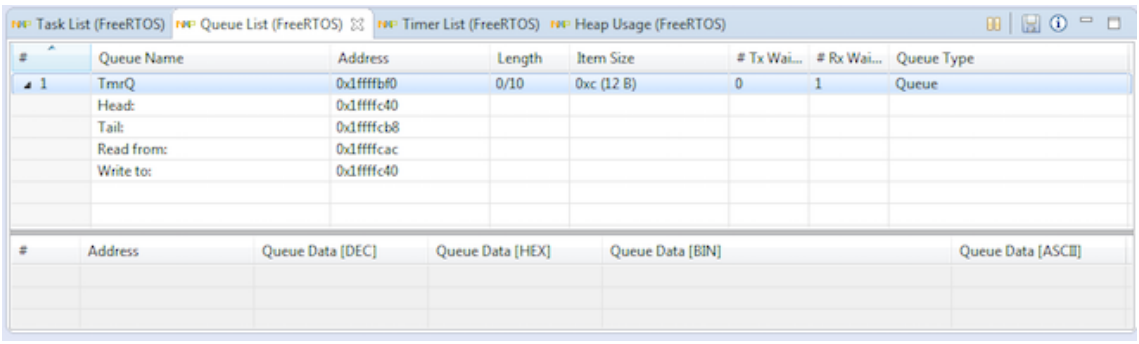
- Stack end address

Stack High Water Mark

- Highest address used by stack at task context switch time

3.3 Queue List View

This View shows the queues, semaphore, and mutex in a table:



The meanings of the columns are as follows.

#

- Number of queue

Queue Name

- Name of queue. Use `configQUEUE_REGISTRY_SIZE` greater than zero and `vQueueAddToRegistry()` to assign a name to a queue, semaphore, or mutex

Address

- Address of queue handle

Length

- Length of queue. The first number indicates the number of elements in the queue, followed by the maximum number of elements possible

Item Size

- Size of an individual element in the queue

 # Tx Waiting

- Number of tasks waiting on a queue until it is not empty

 # Rx Waiting

- Number of tasks waiting until an element is placed into the queue

Queue Type

- Type of queue, either Queue, semaphore, or mutex

Unfolding a queue line item shows the following information:

Head

- Address of queue head item (first item in the queue)

Tail

- Address of queue tail item (last item in the queue)

Read from

- Address of current reading element

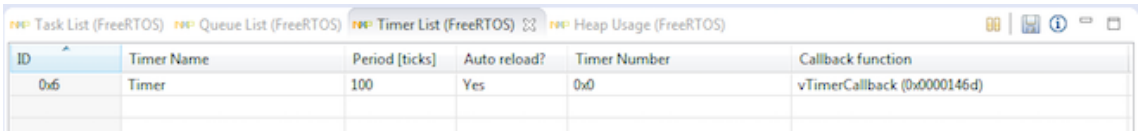
Write to

- Address of next empty item in the queue

Clicking on an element in the queue shows details about it.

3.4 Timer List View

This View shows the software timers in a table:



ID

- ID of timer, assigned by `vTimerSetTimerID()`.

Timer Name

- Name of timer

Period (ticks)

- Period of timer in ticks

Auto reload?

- Whether the timer is automatically restarted after expiration

Timer Number

- Number of timer

Timer callback

- Address and name of callback function

3.5 Heap Usage View

This View provides information about the heap memory used.

3.5.1 Memory Scheme in Use

The Heap Usage View determines the used memory scheme (heap type) from:

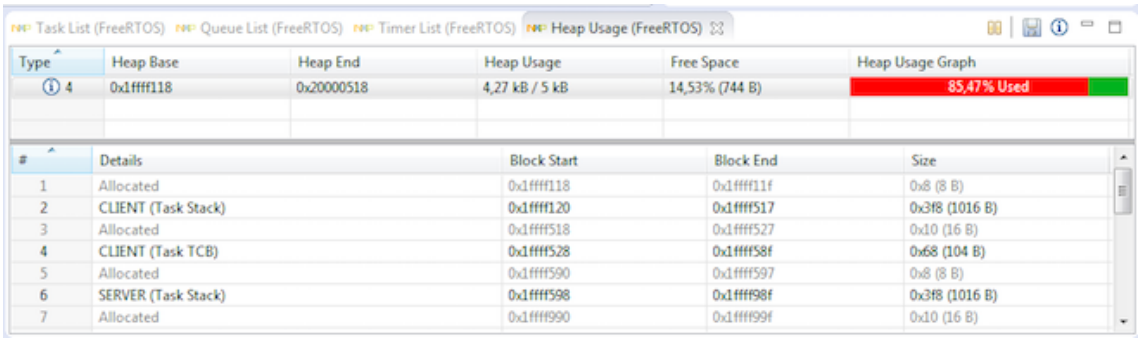
1. The value of the `configFRTOS_MEMORY_SCHEME` in the `FreeRTOSDebugConfig` structure (as described in Required Source Code Changes [5] above)
2. Else, the value of the user-defined variable `freeRTOSMemoryScheme`
3. Else from the details contained in the available FreeRTOS heap related variables (`ucHeap`, `xHeapStructSize` and `heapSTRUCT_SIZE`).

If the `freeRTOSMemoryScheme` variable is to be used, then this can be defined as follows, but you must ensure that there is a reference to this symbol, so that it is not removed by the linker.

```
static const uint8_t freeRTOSMemoryScheme = 2; /* memory scheme 2 used */
```

3.5.2 Heap Usage View Functionality

The Heap Usage View provides the following information.



Type

- Memory scheme number

Heap Base

- Start address of heap

Heap End

- End address of the heap memory

Heap Usage

- Amount of memory used with the total amount of memory

Free Space

- Amount of free memory with percentage

Heap Usage Graph

- Graphical view of percentage used

In the lower part of the View there is information about the heap memory blocks:

#

- Block number

Details

- Allocated, Free or the Task Stack or Task TCB

Block Start

- Start address of memory

Block End

- End address of memory

Size

- Size of memory

4. Thread Aware Debugging with Other Debug Probes

4.1 P&E Micro Probes

FreeRTOS thread aware debugging with P&E Micro debug probes is automatically supported without any special option.

4.2 SEGGER J-Link Probes

FreeRTOS thread aware debugging for SEGGER J-Link debug probes is disabled by default.

To turn it on, enable the “Select RTOS plugin” option for “GDBServer/RTOSPlugin_FreeRTOS” in the Launch Configuration for your project: