Micriμm, Inc.

μC/OS-II
Kernel Awareness
for C-SPY

User Guide

μC/OS-II: V2.81
μC/OS-II Kernel Awareness Plug-in in C-SPY: V2.10
IAR Embedded Workbench: 4.6

www.Micrium.com

Contents

CONTENTS **2**

INTRODUCTION **3**

µC/OS-II KERNEL AWARENESS IN C-SPY 3
µC/OS-II V2.81 3
IAR EMBEDDED WORKBENCH AND C-SPY DEBUGGER 3

QUICK START **4**

INSTALLATION **5**

REQUIREMENTS 5
SETUP 5
CONFIGURING THE SIMULATOR 5

USING THE PLUG-IN **7**

STARTING THE PLUG-IN 7
UPDATING DATA 7
LIST WINDOW CONTROLS 8
WINDOW MANAGEMENT 8
DISABLED FEATURES 8
KERNEL OBJECT NAMES 9

REFERENCE **10**

DISPLAY CONVENTIONS 10
STATUS 10
TASK LIST 13
TIMER LIST 17
SEMAPHORE LIST 19
MUTEX LIST 20
MAILBOX LIST 21
QUEUE LIST 22
EVENT FLAG GROUPS 23
MEMORY PARTITIONS 24
CONFIG. CONSTANTS 25
OPTIONS 26
ABOUT 27

BIBLIOGRAPHY **28**

CONTACTS **28**

Introduction

This User Guide describes how to use the µC/OS-II kernel awareness capabilities with the IAR Embedded Workbench C-SPY Debugger.

- The *Quick Start* section helps you start using it right away.
- The *Installation* section explains the installation process in detail.
- The *Reference* section provides a comprehensive description of all the features.

µC/OS-II Kernel Awareness in C-SPY

The µC/OS-II Kernel Awareness is added to the C-SPY Debugger as a Plug-in which is automatically loaded when the Debugger is started. The Plug-in can be enabled or disabled in the Project's Options under the Debugger's Plugins tab. Version 2.10 of the µC/OS-II Kernel Awareness Plug-in is compatible with version 4.6 (and higher) of the IAR Embedded Workbench.

Kernel Awareness allows you to display µC/OS-II's internal data structures in a convenient series of Windows integrated with the C-SPY Debugger within the IAR Embedded Workbench. This provides you with information about each of the active tasks in the target application, about each semaphore, mutex, mailbox, queue and event flag group along with a list of all the tasks waiting on these kernel objects, and more. This can become very useful to the embedded developer when testing and debugging applications.

µC/OS-II V2.81

Although previous versions of µC/OS-II provided many features to support kernel awareness, as of µC/OS-II V2.6, a name can be assigned to each kernel object, such as a task, a semaphore, a mutex, a mailbox, a queue, an event flag group, a memory partition and a timer. A kernel aware debugger can thus display the name of these objects, and allow you to quickly see information about these objects. Also, V2.6 allows the debugger to obtain the configuration of an application. V2.81 integrates timer management.

IAR Embedded Workbench and C-SPY Debugger

The IAR Embedded Workbench is a powerful Integrated Development Environment that allows you to develop and manage a complete embedded application project for a variety of target processors in a convenient Windows interface. This IDE is the framework where all necessary tools are integrated: a C/EC++ compiler, an assembler, a linker, an editor, a project manager, and the C-SPY™ Debugger.

The IAR C-SPY Debugger is a high-level language debugger for embedded applications. It is designed for use with the IAR compilers and assemblers, and is completely integrated in the IAR Embedded Workbench IDE, providing seamless switching between development and debugging. Some C-SPY Debuggers are available in a simulator, emulator, and ROM-monitor versions. The simulator version simulates the functions of the target processor entirely in software. The Emulator version provides control over an in-circuit emulator, which is connected to the host computer. The ROM-monitor version provides a low-cost solution to real-time debugging.

C-SPY can be extended with Plug-ins to provide Kernel Awareness capabilities during debugging. This is what the µC/OS-II Kernel Awareness Plug-in for C-SPY provides.

Quick Start

These are the minimum steps necessary to start using the µC/OS-II Kernel Awareness Plug-in with the IAR Embedded Workbench.

1. **Run `Setup.exe`**
   - Follow the setup instructions to install the µC/OS-II KA Plug-in.

2. **Start the IAR Embedded Workbench**

3. **Open your Project**
   - The project should be for a functional µC/OS-II application.

4. **Start the Debugger**
   - The **µC/OS-II** menu will appear in the menu bar, unless there were compiler/linker errors.
   - Make sure the application runs for a few hundred clock cycles to allow µC/OS-II to be initialized.

5. **Setup the Simulator (if applicable)**
   - If you are running the target using a Simulator, you need to simulate the clock tick interrupt so that µC/OS-II will function in a useful way. That is, if there are no clock ticks, then µC/OS-II won't work properly.

     For the ARM processor, for example:
     a. Open **Interrupts...** from the **Low level** menu
     b. Select Interrupt "IRQ 1 0x18 CPSR.I"
     c. Set Repeat Interval = 2500, Latency = 50, Probability = 100, Variance = 2
     d. Click **Install**

     This must be done every time the Debugger is started, but fortunately you can use a C-SPY macro to automate this step. (See *Installation* section for more details.)

6. **Open the µC/OS-II kernel awareness Status window**
   - Note: This is the first item in the **µC/OS-II** menu.
   - Verify that µC/OS-II is detected.
   - Click **Update All** to make the information reflect the current state of µC/OS-II.

7. **Open other µC/OS-II kernel awareness windows**
   - There is a specific window for each type of µC/OS-II kernel object, as well as windows for Configuration Constants, Options and About information.
   - µC/OS-II windows are managed the same way as other C-SPY Debugger windows in the IDE workspace. You can also use the features in the **Window** menu to select or organize the windows.

See *Using the Plug-in* and *Reference* sections for details.

Installation

Requirements

µC/OS-II kernel awareness requires the following:

Software

IAR Embedded Workbench IDE: V4.6 (or higher) for your target processor
µC/OS-II: V2.81 (or higher) with a port for your target processor

Processors

µC/OS-II kernel awareness should work with any processor as long it is supported by C-SPY.

Environment

Windows 95, Windows 98, Windows NT, Window 2000, or Windows XP

Setup

Run `Setup.exe` and follow the instructions.

If you have multiple installations of the IAR Embedded Workbench, the Plug-in must be installed on each one for which you wish to use the µC/OS-II KA Plug-in. It must also be installed on each product family (of a EW installation) for which you wish to use the µC/OS-II KA Plug-in.

The installation process will copy the Plug-in DLL under the `plugins` directory of each selected IAR Embedded Workbench installation path.

Configuring the Simulator

If you are running your application using a Simulator, you need to simulate 'clock tick' interrupts so that µC/OS-II will function in a useful way. That is, if there are no clock ticks, the µC/OS-II tasks won't be able to delay or timeout.

To avoid configuring the clock tick interrupts every time the Debugger is started (as described in *Quick Start* section), you can create a C-SPY macro to perform this task automatically.

1. Create a macro file, which you can name "`irq-clock.mac`", or edit an existing macro file for your target, and add this to it:

```
execUserSetup()
{
  __orderInterrupt("0x18",10000,2500,2,50,100);
}
```

You can set the interrupt parameters that are appropriate for your context. In the example above:

- 0x18 is the interrupt vector (used for the ARM processor)
- 10000 is the Activation Time

- 2500 is the Repeat Interval. A smaller interval may cause high CPU usage, and even prevent user code from being excuted since most of the time would be spent in the OS. The right balance depends on the performance of your computer. A value of 10000 works well on a Pentium IV running at 1GHz.
- 2 is the Variance
- 50 is the Latency
- 100 is the Probability

(See the IAR Embedded Workbench IDE User Guide for more information)

2. Place the macro file in your project directory (or in a more central location for all your projects).

3. In your IAR EW project, under **Project .. Options .. Debugger .. Setup**, enable **Use macro file** and click the browse button to select your macro file. This causes the macro file to execute automatically every time the Debugger is started. This should be indicated by an entry in the Debug Log window stating "Loaded macro file: `<path>\irq-clock.mac`"

Using the Plug-in

This section describes how to use the features of the µC/OS-II Kernel Awareness Plug-in. The detailed description for each window is found in the *Reference* section.

Starting the Plug-in

In the IAR Embedded Workbench, open your project containing a functional µC/OS-II application. When you start the Debugger (i.e. **Project .. Debug**), the **µC/OS-II** menu will appear in the menu bar, unless there were compiler/linker errors.

You can now open any of the µC/OS-II windows from the **µC/OS-II** menu.

You should first check the status of µC/OS-II using the **Status** window, which will indicate if the application is using µC/OS-II and if µC/OS-II is running. See *Status* on page 10 for details.

Typically, the debugger will initially break upon entering the `main()` function of the application. At this point, µC/OS-II is not initialized, so its status should be "Not Running" but you will be able to see its version. Also, you can see the application's configuration constants in the **Config. Constants** window, which are useful if you need to diagnose how µC/OS-II is configured.

Updating Data

The kernel data will not be available until µC/OS-II has had time to initialize it. In other words, µC/OS-II's data structures will be initialized only when `OSInit()` is executed. Because of this, the information windows will be empty until data is available. Depending on your application, it may take a few hundred clock cycles to have all the data.

You can monitor the changes in kernel data in different ways:

1. Set breakpoints in the application. If the **Auto Update** option is enabled (see *Options* on page 26 for details), the kernel data will be re-read each time a breakpoint is reached, and will be used to refresh the contents of the windows.

2. Force a Break while the application is running (i.e. **Debug .. Break**), causing the same effect as a normal breakpoint.

3. Click **Update All** in the **Status** window while the application is running, or select **Update** from the context menu of any of the List windows.

Each time, the contents of the open windows change to reflect the current data in µC/OS-II. For example, you will see the Time (ticks) value increment in the **Status** window.

List Window Controls

Most µC/OS-II windows are List windows showing columns of information for each item in the list. These windows have the following special features:

AutoFit Columns
The width of each column is automatically adjusted to fit its content whenever the window is refreshed. The column's width is set to fit the text in the largest item. This feature can be disabled from the context menu.

Sort
List items can be sorted by clicking on the header of the column to use as sorting criteria. Clicking again toggles the sort order between ascending and descending.

Context Menu
Right-clicking anywhere in the window opens the context menu. Every list window has the following commands in the context menu:

- **AutoFit Columns** : Enables/Disables *AutoFit Columns* feature for this window.

- **Refresh** : Redraws the window contents (using AutoFit if needed) with the currently known target data (i.e. target data is not re-read from the target).

- **Update** : Re-reads all the data from the target, and force an update of all the µC/OS-II windows. This is the same as **Update All** in the **Status** window.

Additional commands may be appended to the end of the menu for specific windows. At this time, only the Task List window has extra context menu commands (See *Task List* on page 12).

Context menu option selections are lost when the window is closed. These are restored to defaults when the window is opened again. However, the options are preserved when the debugger is stopped while the window is open.

Window Management

When the Debugger is re-started, the µC/OS-II windows that were open when it was stopped are restored, and their settings such as sorting and AutoFit are restored.

Disabled features

Most of the features of µC/OS-II can be disabled using Configuration Constants. If major features (Semaphores, Mutexes, Mailboxes, Queues, Event Flag Groups, Memory Partitions, Timers) are disabled, their corresponding window will show a message like this one for the Semaphore List:

```
Semaphore functionality disabled.
To enable, set OS_SEM_EN to 1 in OS_CFG.H
```

Other features can be disabled which will cause some kernel data to be unavailable. In such cases, the information will be left blank, 'n/a' or '?'. For example, if `OS_TASK_STAT_EN` is `0` then **CPU Usage** will remain **n/a** in the **Status** window. Also, most Stack statistics will not be available in the **Task List** window if `OS_TASK_PROFILE_EN` or `OS_TASK_CREATE_EXT_EN` are `0`.

Kernel Object Names

As of V2.6, µC/OS-II allows you to assign names to kernel objects to help identify them in Kernel Awareness tools:

There are five(5) types of kernel object that can be named:

Kernel Object Type	**Name-Size Constant**	**Function used to set the name**
Task	`OS_TASK_NAME_SIZE`	`OSTaskNameSet()`
Event (Sem., Mutex, Mailbox, Queue)	`OS_EVENT_NAME_SIZE`	`OSFlagNameSet()`
Event Flag Group	`OS_FLAG_NAME_SIZE`	`OSFlagNameSet()`
Memory Partition	`OS_MEM_NAME_SIZE`	`OSMemNameSet()`
Timer	`OS_TMR_CFG_NAME_SIZE`	`OSTmrStart()`

You can assign a name to an object if its name-size configuration constant is greater than `0`. This constant actually establishes the number of characters allowed for the object names and must account for a `NUL`-terminated ASCII string. Object names are assigned to each object *after* the object is created by calling its corresponding `OS..NameSet()` function (refer to the µC/OS-II release notes for details).

Task names are shown in the *Name* column of the **Task List** window and in the Tasks Waiting column of the **Semaphore**, **Mutex**, **Mailbox**, **Queue** List windows, as well as in the **Event Flag Group** window. Event names are used in the Name column of their corresponding window and in the Waiting On column of the **Task List** window.

Reference

This section contains the description of each µC/OS-II Kernel Awareness window with a definition for each item of information.

Display Conventions

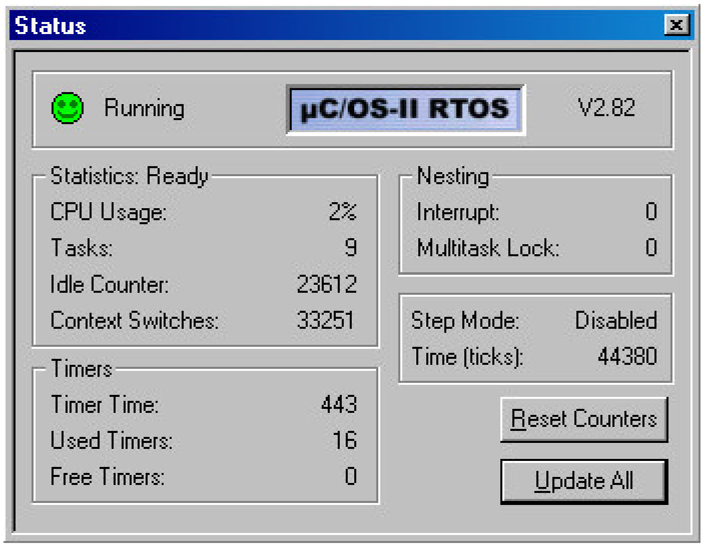
Address values are expressed in hexadecimal format. The width of an address value depends on the target processor's addressing capabilities (2, 3 or 4 bytes), which is indicated by `OSPtrSize`.

Numerical values are displayed with a right-justified alignment, while text values are displayed with left-justification.

Screen output may differ from what is seen here, depending on the system and its configuration.

Status

The Status window shows general information concerning µC/OS-II and contains general controls.



Running V2.82

Statistics: Ready
CPU Usage: 2%
Tasks: 9
Idle Counter: 23612
Context Switches: 33251

Nesting
Interrupt: 0
Multitask Lock: 0

Step Mode: Disabled
Time (ticks): 44380

Timers
Timer Time: 443
Used Timers: 16
Free Timers: 0

Information

Status

The status of µC/OS-II can be one of the following:

Not Detected	❌	µC/OS-II code not present in target application
Debug Disabled	⚠	`OSDebugEn = 0`. Debug Mode must be enabled for kernel awareness
Not Running	⛔	µC/OS-II has not started running (Not enough clock cycles occurred to be initialized)
Running	☺	µC/OS-II is running

Version
Current version of µC/OS-II in target application. If µC/OS-II is not detected or debug mode is disabled, then the version shows V?.??. Version 2.62 and above is necessary for Kernel Awareness to be functional.

Statistics: Ready / Not Ready
Based on the value of OSStatRdy, indicating if the Statistics task is ready. This is only relevant if OS_TASK_STAT_EN is set.

CPU Usage
Percentage of CPU used. (OSCPUUsage)

Tasks
Total number of tasks running, i.e. includes system tasks. (OSTaskCtr)

Idle Counter
Idle counter. (OSIdleCtr)
This counter is reset by the **Reset Counters** feature.

Context Switches
Number of context switches. (OSCtxSwCtr)
This counter is reset by the **Reset Counters** feature.

Nesting – Interrupt
Interrupt nesting level. (OSIntNesting)

Nesting - Multitask Lock
Multitasking lock nesting level. (OSLockNesting)

Step Mode
Indicates the state of the tick step feature:

Value	OSTickStepState	Description
Disabled	0	Stepping is disabled; Tick runs as normal
Waiting	1	Waiting for µC/OS-View to set OSTickStepState to _ONCE
Stepped	2	Process tick once and wait for next command from µC/OS-View
Unknown	*any other value*	Non-supported value

Time (ticks)
Current value of system time (in ticks). (OSTime)

Timer Time
Current value of timer time. (OSTmrTime)
It increments every OS_TICKS_PER_SEC / OS_TMR_CFG_TICKS_PER_SEC.

Used Timers
Number of timers used. (OSTmrUsed)

Free Timers
Number of timers in the list of unused timers. (OSTmrFree)

Controls

Update All

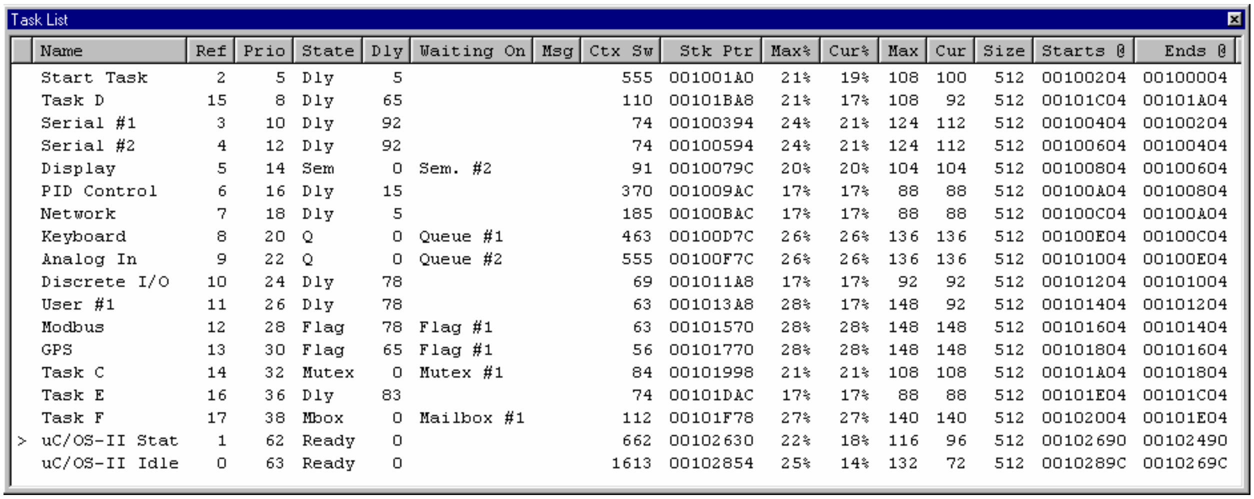
Updates all the µC/OS-II data (by reading from the target) and forces a refresh of all the windows to show the new data. This is automatically done when the target stops at a breakpoint, but it is also permitted to perform an update while the target is running, although you must be aware that the data may be in transition while it is being read, and may produce some inconsistencies.

Reset Counters

Resets (to 0) the global Context Switches counter (`OSCtxSwCtr`) and the global Idle counter (`OSIdleCtr`), as well as the Context Switches counter (`.OSTCBCtxSwCtr`) of every task.

Task List

The Task List window shows information about each task running, as well as its Stack information.



Task List

	Name	Ref	Prio	State	Dly	Waiting On	Msg	Ctx Sw	Stk Ptr	Max%	Cur%	Max	Cur	Size	Starts @	Ends @
	Start Task	2	5	Dly	5			555	001001A0	21%	19%	108	100	512	00100204	00100004
	Task D	15	8	Dly	65			110	00101BA8	21%	17%	108	92	512	00101C04	00101A04
	Serial #1	3	10	Dly	92			74	00100394	24%	21%	124	112	512	00100404	00100204
	Serial #2	4	12	Dly	92			74	00100594	24%	21%	124	112	512	00100604	00100404
	Display	5	14	Sem	0	Sem. #2		91	0010079C	20%	20%	104	104	512	00100804	00100604
	PID Control	6	16	Dly	15			370	001009AC	17%	17%	88	88	512	00100A04	00100804
	Network	7	18	Dly	5			185	00100BAC	17%	17%	88	88	512	00100C04	00100A04
	Keyboard	8	20	Q	0	Queue #1		463	00100D7C	26%	26%	136	136	512	00100E04	00100C04
	Analog In	9	22	Q	0	Queue #2		555	00100F7C	26%	26%	136	136	512	00101004	00100E04
	Discrete I/O	10	24	Dly	78			69	001011A8	17%	17%	92	92	512	00101204	00101004
	User #1	11	26	Dly	78			63	001013A8	28%	17%	148	92	512	00101404	00101204
	Modbus	12	28	Flag	78	Flag #1		63	00101570	28%	28%	148	148	512	00101604	00101404
	GPS	13	30	Flag	65	Flag #1		56	00101770	28%	28%	148	148	512	00101804	00101604
	Task C	14	32	Mutex	0	Mutex #1		84	00101998	21%	21%	108	108	512	00101A04	00101804
	Task E	16	36	Dly	83			74	00101DAC	17%	17%	88	88	512	00101E04	00101C04
	Task F	17	38	Mbox	0	Mailbox #1		112	00101F78	27%	27%	140	140	512	00102004	00101E04
>	uC/OS-II Stat	1	62	Ready	0			662	00102630	22%	18%	116	96	512	00102690	00102490
	uC/OS-II Idle	0	63	Ready	0			1613	00102854	25%	14%	132	72	512	0010289C	0010269C

Information

Current Task

The first column indicates the currently running task, with a '>' symbol.

Name

Name of the task. If it has no name, it is set to '?'. See *Kernel Object Names* on page 9.

Ref

Index of the task in `OSTCBTbl[]`. This also corresponds to the order in which the tasks were created.

Prio

Priority assigned to each task. The default sort criteria for the Task List is the Priority in ascending order. (`.OSTCBPrio`) Priority values range from 0 (highest) to 63 (lowest).

State
State of the task. The possible state values are:

State	Description	Value of `.OSTCBStat`
Ready	Ready to run	`OS_STAT_RDY`
Dly	Waiting for time to expire	`OS_STAT_RDY` but `.OSTCBDly` is non-zero
Sem	Waiting on a semaphore	`OS_STAT_SEM`
Mutex	Waiting on a mutual exclusion semaphore	`OS_STAT_MUTEX`
Flag	Waiting on an event flag group	`OS_STAT_FLAG`
Mbox	Waiting for a message at a mailbox	`OS_STAT_MBOX`
Q	Waiting for a message at a queue	`OS_STAT_Q`
Sem+Suspended	Waiting on a semaphore and task is also suspended	`OS_STAT_SEM + OS_STAT_SUSPEND`
Mutex+Suspended	Waiting on a mutual exclusion semaphore and the task is also suspended	`OS_STAT_MUTEX + OS_STAT_SUSPEND`
Flag+Suspended	Waiting on an event flag group and the task is also suspended	`OS_STAT_FLAG + OS_STAT_SUSPEND`
Mbox+Suspended	Waiting for a message at a mailbox and the task is also suspended	`OS_STAT_MBOX + OS_STAT_SUSPEND`
Q+Suspended	Waiting for a message at a queue and the task is also suspended	`OS_STAT_Q + OS_STAT_SUSPEND`

Dly
Amount of time (in ticks) the task has been delayed (if the State column indicates 'Dly') or, the amount of time left that the task will be waiting for either the semaphore, the mutex, the event flag group, the mailbox or the queue (if the State column indicates an object type). The value is `0` if the task will wait forever for one of the objects. (`.OSTCBDly`)

Waiting On
Name of the object (if any) for which the task is waiting. This can be either an Event Flag Group or an Event (Semaphore, Mutex, Mailbox, or Queue).

Msg
Message received from `OSMboxPost()` or `OSQPost()`. This pointer is shown in hexadecimal format. Typically, it will be empty unless you single-step through the code and a 'Post' call deposits a message either to the mailbox or a queue that the task is waiting for. (`.OSTCBMsg`)

Ctx Sw
Number of times the task was 'switched-in'. This counter can be reset to 0 by selecting **Reset Counters** from the context menu, or by clicking the **Reset Counters** button in the **Status** window. This counter is only available if you set the configuration constant `OS_TASK_PROFILE_EN` to `1` which should be done when you are using the kernel awareness feature of µC/OS-II. (`.OSTCBCtxSwCtr`)

Stk Ptr
Current value of the task's stack pointer (in hexadecimal).

Notes on Stack Statistics:

- The following Stack-related information fields can be disabled with the **Stack Stats** feature in the context menu or in the **Options** window.
- Most of these values assume that tasks were created with `OSTaskCreateExt()`, specifying `OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR` for the `opt` argument. Also, you need to enable the Statistics task (set `OS_TASK_STAT_EN` to `1` in `OS_CFG.H`), and enable stack checking by the Statistics task (set `OS_TASK_STAT_STK_CHK_EN` to `1`).

Max%

Maximum stack space used by the task expressed as a percentage. For example, a value of 47% means that, during execution of the task, the total stack space used never exceeded 47%. This value is reset to 0 by the **Reset StkUsed** feature of the context menu.

```
(.OSTCBStkUsed / (.OSTCBStkSize * OSStkWidth))
```

Cur%

Current stack usage of the task expressed as a percentage. For example, a value of 39% means that the stack pointer is currently located 39% into the stack.

```
(abs(.OSTCBStkPtr - .OSTCBStkBase) / (.OSTCBStkSize * OSStkWidth))
```

Max

Maximum stack space used by the task (in bytes). This value is reset to 0 by the **Reset StkUsed** feature of the context menu. (`.OSTCBStkUsed`)

Cur

Current stack usage of the task (in bytes).

```
(abs(.OSTCBStkPtr - .OSTCBStkBase))
```

Size

Number of bytes allocated for the task stack.

```
(.OSTCBStkSize * OSStkWidth)
```

Starts @

Address of the beginning of the stack. If the stack, on the processor you are using, grows downwards (i.e. `OS_STK_GROWTH` set to `1` in `OS_CPU.H`) then this indicates the *highest* address that the stack pointer can take, otherwise (i.e. `OS_STK_GROWTH` set to `0`), this indicates the *lowest* address the stack pointer can take. (`.OSTCBStkBase`)

Ends @

Address of the end of the stack. If the stack, on the processor you are using, grows downwards (i.e. `OS_STK_GROWTH` set to `1` in `OS_CPU.H`) then this indicates the *lowest* address that the stack pointer can take, otherwise (i.e. `OS_STK_GROWTH` set to `0`), this indicates the *highest* address the stack pointer can take.

Controls

In addition to the standard context menu features (described in *Using the Plug-in* section), the following features are available in the Task List context menu:

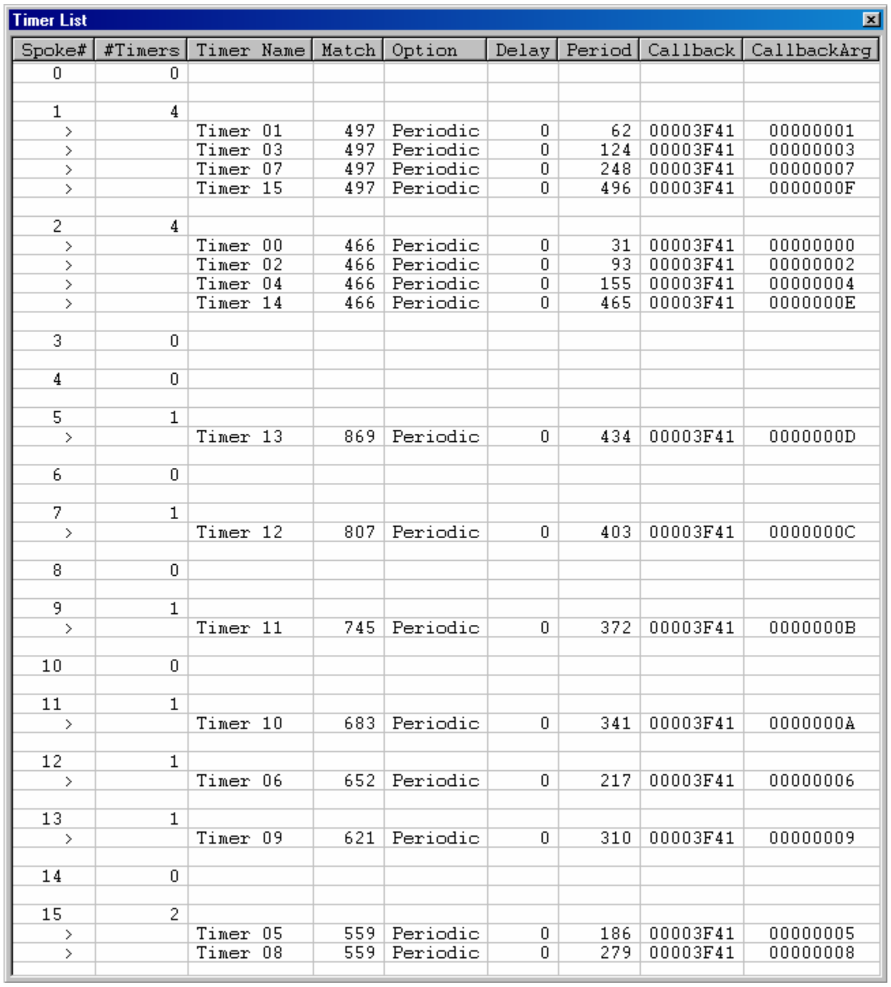
- **StackStats** : Shows/Hides the Stack-related information fields (except Stk Ptr). This is the same as in the **Options** window.

- **Reset Counters** : This has the same effect as the **Reset Counters** button of the **Status** window.

- **Reset StkUsed** : Resets (to 0) the Stack Used counter (`.OSTCBStkUsed`) of every task, represented in the Max% and Max columns.

Timer List

The Timer window shows information about the timers (OS_TMR) in the pool of timers. It is based on a representation of the timer manager 'wheel' (OSTmrWheelTbl[]).



Spoke#	#Timers	Timer Name	Match	Option	Delay	Period	Callback	CallbackArg
0	0							
1	4							
>		Timer 01	497	Periodic	0	62	00003F41	00000001
>		Timer 03	497	Periodic	0	124	00003F41	00000003
>		Timer 07	497	Periodic	0	248	00003F41	00000007
>		Timer 15	497	Periodic	0	496	00003F41	0000000F
2	4							
>		Timer 00	466	Periodic	0	31	00003F41	00000000
>		Timer 02	466	Periodic	0	93	00003F41	00000002
>		Timer 04	466	Periodic	0	155	00003F41	00000004
>		Timer 14	466	Periodic	0	465	00003F41	0000000E
3	0							
4	0							
5	1							
>		Timer 13	869	Periodic	0	434	00003F41	0000000D
6	0							
7	1							
>		Timer 12	807	Periodic	0	403	00003F41	0000000C
8	0							
9	1							
>		Timer 11	745	Periodic	0	372	00003F41	0000000B
10	0							
11	1							
>		Timer 10	683	Periodic	0	341	00003F41	0000000A
12	1							
>		Timer 06	652	Periodic	0	217	00003F41	00000006
13	1							
>		Timer 09	621	Periodic	0	310	00003F41	00000009
14	0							
15	2							
>		Timer 05	559	Periodic	0	186	00003F41	00000005
>		Timer 08	559	Periodic	0	279	00003F41	00000008

Information

Spoke#
Spoke number in the wheel. (0 to OS_TMR_CFG_WHEEL_SIZE-1)

#Timers
Number of timers in this Spoke. (.OSTmrEntries of OS_TMR_WHEEL)

Timer Name
Name of the timer. If it has no name, it is set to '?'. See *Kernel Object Names* on page 9.

Match
Value used by the timer manager to determine if a timer has expired. A timer expires when OSTmrTime == .OSTmrMatch

Option
Type of timer.

Description	**Value of `.OSTmrOpt`**
One-Shot	`OS_TMR_OPT_ONE_SHOT`
Periodic	`OS_TMR_OPT_PERIODIC`

Delay
Time before the first signaling of a periodic timer (in timer time).

Period
Period with which the timer will repeat (in timer time).

Callback
Pointer to the function to call when timer expires.

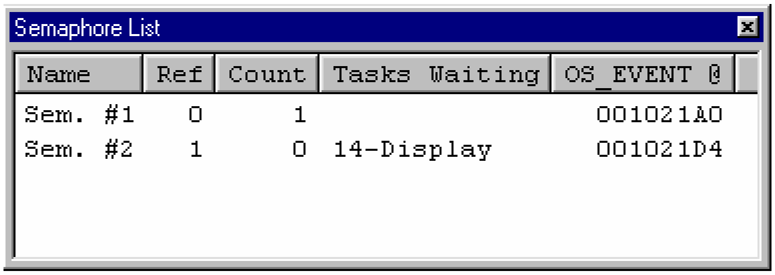
CallbackArg
Argument to pass to the callback function when the timer expires.

Controls

There are no special controls for this window in addition to the standard list window controls. See *List Window Controls* on page 8 for details.

Semaphore List

The Semaphore window shows information about OS_EVENT structures that were created as semaphores.



Name	Ref	Count	Tasks Waiting	OS_EVENT @
Sem. #1	0	1		001021A0
Sem. #2	1	0	14-Display	001021D4

Information

Name
Name of the semaphore. If it has no name, it is set to '?'. See *Kernel Object Names* on page 9.

Ref
Index of the semaphore structure in `OSEventTbl[]`. This also corresponds to the order in which the events were created.

Count
Value of the semaphore interpreted from `.OSEventCnt`.

Tasks Waiting
Tasks waiting on the semaphore. A task is represented by its Priority followed by its Name (if any). If more than one task is waiting on the same semaphore, additional rows are added for the same semaphore with duplicate information in the common columns.

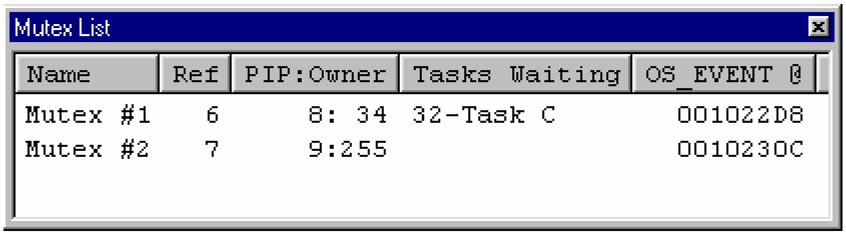
OS_EVENT @
Address of the `OS_EVENT` structure.

Controls

There are no special controls for this window in addition to the standard list window controls. See *List Window Controls* on page 8 for details.

Mutex List

The Mutex window shows information about `OS_EVENT` structures that were created as Mutual exclusion semaphores.



Name	Ref	PIP:Owner	Tasks Waiting	OS_EVENT @
Mutex #1	6	8: 34	32-Task C	001022D8
Mutex #2	7	9:255		0010230C

Information

Name
Name of the mutex. If it has no name, it is set to '?'. See *Kernel Object Names* on page 9.

Ref
Index of the mutex structure in `OSEventTbl[]`. This also corresponds to the order in which the events were created.

PIP-Owner (high-byte - low-byte)
The first value is the PIP (Priority Inheritance Priority) of the mutex, interpreted from the 'upper' eight bits of `.OSEventCnt` . The second value is the priority of the task that owns the mutex or 255 if the mutex is available (i.e. not owned), interpreted from the 'lower' eight bits of `.OSEventCnt`.

Tasks Waiting
Tasks waiting on the mutex. A task is represented by its Priority followed by its Name (if any). If more than one task is waiting on the same mutex, additional rows are added for the same mutex with duplicate information in the common columns.

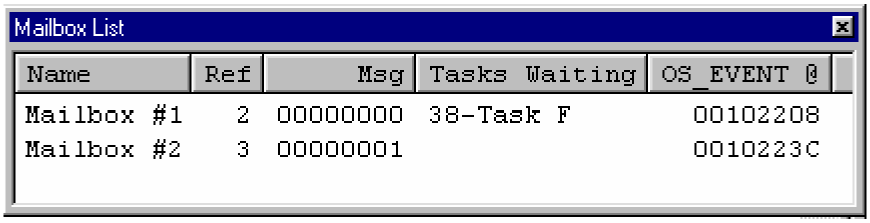
OS_EVENT @
Address of the `OS_EVENT` structure.

Controls

There are no special controls for this window in addition to the standard list window controls. See *List Window Controls* on page 8 for details.

Mailbox List

The Mailbox window shows information about `OS_EVENT` structures that were created as mailboxes.



Mailbox List

Name	Ref	Msg	Tasks Waiting	OS_EVENT @
Mailbox #1	2	00000000	38-Task F	00102208
Mailbox #2	3	00000001		0010223C

Information

Name
Name of the mailbox. If it has no name, it is set to '?'. See *Kernel Object Names* on page 9.

Ref
Index of the mailbox structure in `OSEventTbl[]`. This also corresponds to the order in which the events were created.

Msg
Current contents of the mailbox. This value is the pointer (`.OSEventPtr`) to the message.

Tasks Waiting
Tasks waiting on the mailbox. A task is represented by its Priority followed by its Name (if any). If more than one task is waiting on the same mailbox, additional rows are added for the same mailbox with duplicate information in the common columns.

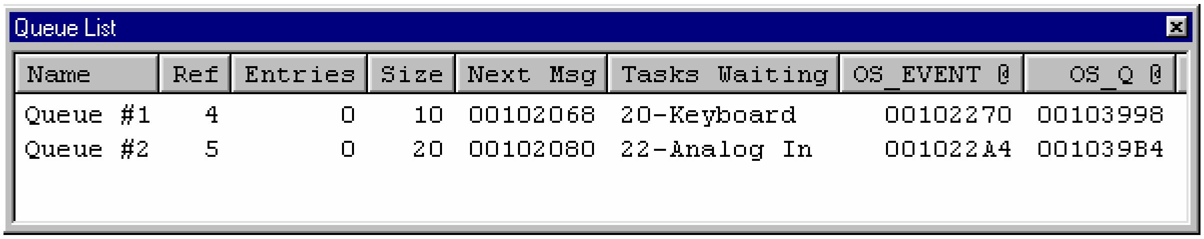
OS_EVENT @
Address of the `OS_EVENT` structure.

Controls

There are no special controls for this window in addition to the standard list window controls. See *List Window Controls* on page 8 for details.

Queue List

The Queue window shows information about `OS_EVENT` structures that were created as message queues.



Name	Ref	Entries	Size	Next Msg	Tasks Waiting	OS_EVENT @	OS_Q @
Queue #1	4	0	10	00102068	20-Keyboard	00102270	00103998
Queue #2	5	0	20	00102080	22-Analog In	001022A4	001039B4

Information

Name
Name of the message queue. If it has no name, it is set to '?'. See *Kernel Object Names* on page 9.

Ref
Index of the message queue in `OSEventTbl[]`. This also corresponds to the order in which the events were created.

Entries
Number of messages currently in the message queue. (`.OSQEntries` in `OS_Q` of the queue)

Size
Maximum number of entries allowed in the message queue. (`.OSQSize` in `OS_Q` of the queue)

Next Msg
Pointer to the next message available from the queue. Note that if there are no messages in the queue (`.OSQEntries` is `0`), then this value is meaningless because it contains whatever was in that message queue position. (`.OSQOut` in `OS_Q` of the queue)

Tasks Waiting
Tasks waiting on the message queue. A task is represented by its Priority followed by its Name (if any). If more than one task is waiting on the same queue, additional rows are added for the same queue with duplicate information in the common columns.

OS_EVENT @
Address of the `OS_EVENT` structure.

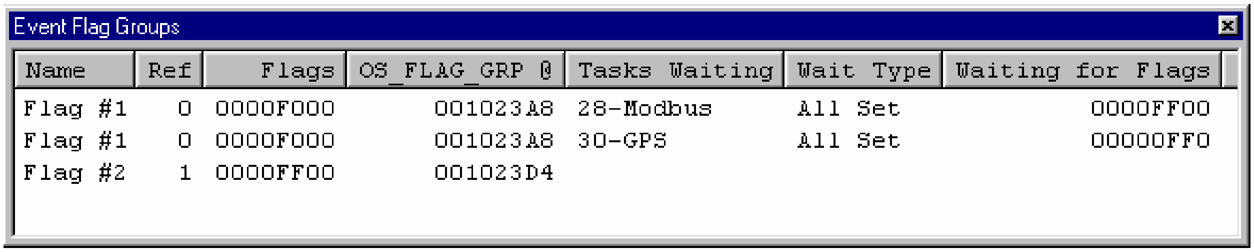
OS_Q @
Address of the `OS_Q` structure of the queue. (`.OSEventPtr`)

Controls

There are no special controls for this window in addition to the standard list window controls. See *List Window Controls* on page 8 for details.

Event Flag Groups

The Event Flag Groups window shows information about the `OS_FLAG_GRP` structures.



Event Flag Groups

Name	Ref	Flags	OS_FLAG_GRP @	Tasks Waiting	Wait Type	Waiting for Flags
Flag #1	0	0000F000	001023A8	28-Modbus	All Set	0000FF00
Flag #1	0	0000F000	001023A8	30-GPS	All Set	00000FF0
Flag #2	1	0000FF00	001023D4			

Information

Name
Name of the event flag group. If it has no name, it is set to '?'. See *Kernel Object Names* on page 9.

Ref
Index of the event flag group in `OSFlagTbl[]`. This also corresponds to the order in which the event flag groups were created.

Flags
Current value stored in the event flag group. The number flag bits used to store flags depends on the definition of `OS_FLAGS` (in `OS_CFG.H`). It could be 8, 16 or 32 bits wide.

OS_FLAG_GRP @
Address of the `OS_FLAG_GRP` structure.

Tasks Waiting
Tasks waiting on the event flag group. A task is represented by its Priority followed by its Name (if any). If more than one task is waiting on the same event, additional rows are added for the same flag with duplicate information in the common columns (as seen in the example above).

Wait Type
Condition for which a task will wait. The possible values are:

Wait Type	Description	Value of `.OSFlagNodeWaitType`
Any Set	A task will wait for ANY of the bits specified in *Waiting For Flags* to be SET in *Flags*.	`OS_FLAG_WAIT_SET_ANY/OR`
All Set	A task will wait for ALL the bits specified in *Waiting For Flags* to be SET in *Flags*.	`OS_FLAG_WAIT_SET_ALL/AND`
Any Clr	A task will wait for ANY of the bits specified in *Waiting For Flags* to be CLEARED in *Flags*.	`OS_FLAG_WAIT_CLR_ANY/OR`
All Clr	A task will wait for ALL the bits specified in *Waiting For Flags* to be CLEARED in *Flags*.	`OS_FLAG_WAIT_CLR_ALL/AND`
+ Consume	Flag will be consumed if condition is satisfied.	`+ OS_FLAG_CONSUME`

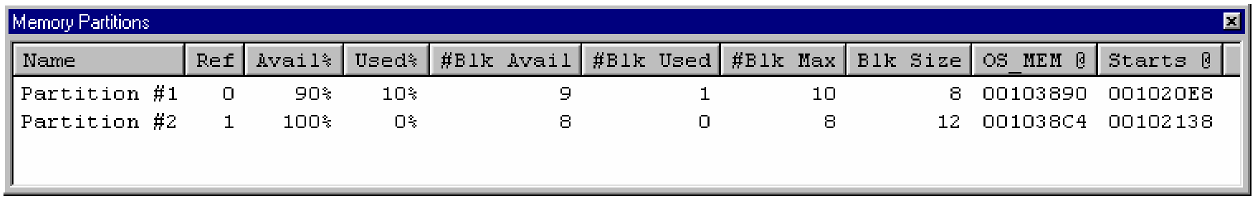
Waiting for Flags
Bits that a task will wait for to be set (or cleared) in the event flag group, depending on *Wait Type*.

Controls

There are no special controls for this window in addition to the standard list window controls. See *List Window Controls* on page 8 for details.

Memory Partitions

The Memory Partitions window shows information about the OS_MEM structures.



Name	Ref	Avail%	Used%	#Blk Avail	#Blk Used	#Blk Max	Blk Size	OS_MEM @	Starts @
Partition #1	0	90%	10%	9	1	10	8	00103890	001020E8
Partition #2	1	100%	0%	8	0	8	12	001038C4	00102138

Information

Name
Name of the memory partition. If it has no name, it is set to '?'. See *Kernel Object Names* on page 9.

Ref
Index of the memory partition in OSMemTbl[]. This also corresponds to the order in which the memory partitions were created.

Avail%
Available memory as a percentage of the memory partition's size.

Used%
Used memory as a percentage of the memory partition's size.

#Blk Avail
Number of memory blocks available in the memory partition. (.OSMemNFree)

#Blk Used
Number of memory blocks in use in the memory partition. (.OSMemNBlks - OSMemNFree)

#Blk Max
Number of memory blocks allocated in the memory partition when it was created. (.OSMemNBlks)

Blk Size
Size (in bytes) of each memory block. (.OSMemBlkSize)

OS_MEM @
Address of the memory partition.

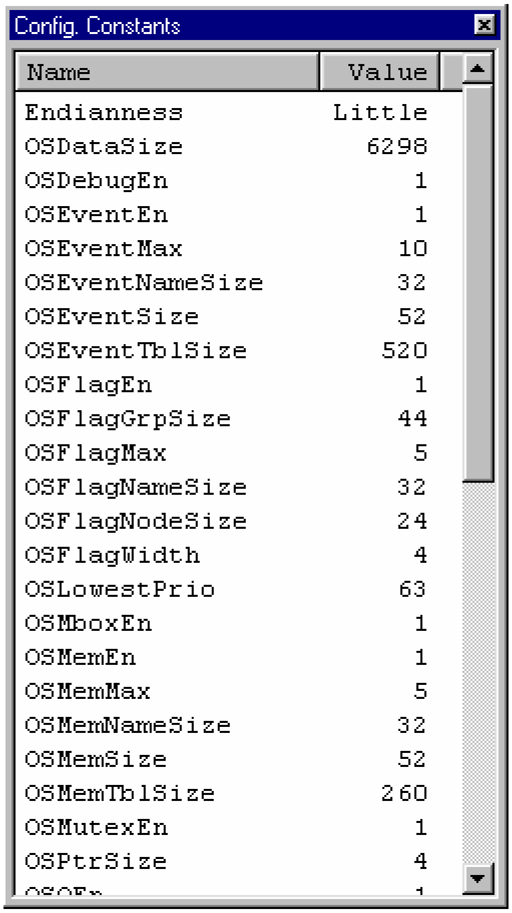
Starts @
Address of the beginning of the memory partition. (.OSMemAddr) This address is typically the base of the storage area and is the *lowest* address of the memory partition. Memory blocks in the memory partition have an address between Starts@ and Starts@ + (#BlkMax * BlkSize).

Controls

There are no special controls for this window in addition to the standard list window controls. See *List Window Controls* on page 8 for details.

Config. Constants

This window lists the µC/OS-II configuration constants in the target application.



Name	Value
Endianness	Little
OSDataSize	6298
OSDebugEn	1
OSEventEn	1
OSEventMax	10
OSEventNameSize	32
OSEventSize	52
OSEventTblSize	520
OSFlagEn	1
OSFlagGrpSize	44
OSFlagMax	5
OSFlagNameSize	32
OSFlagNodeSize	24
OSFlagWidth	4
OSLowestPrio	63
OSMboxEn	1
OSMemEn	1
OSMemMax	5
OSMemNameSize	32
OSMemSize	52
OSMemTblSize	260
OSMutexEn	1
OSPtrSize	4
OSQEn	1

Information

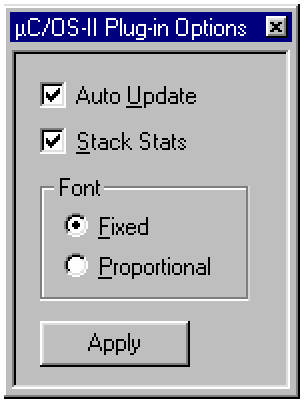
A description of the Configuration Constants can be found in `OS_DEBUG.C` or with their equivalent definitions in `OS_CORE.C`.

Controls

There are no special controls for this window in addition to the standard list window controls. See *List Window Controls* on page 8 for details.

Options

This window controls the general optional features of the µC/OS-II Kernel Awareness Plug-in. More specific options can also be found in the context menu of List windows.



µC/OS-II Plug-in Options

☑ Auto Update

☑ Stack Stats

Font

● Fixed

○ Proportional

Controls

Auto Update

When enabled, the target data is re-read each time a breakpoint is reached, and is used to refresh the contents of the windows. When disabled, the user must manually update the data, using the **Update** command of the context menu in List windows, or the **Update All** button in the **Status** window.

Stack Stats

Shows/Hides the Stack-related information fields in the Task List window. These are Max%, Cur%, Max, Cur, Size, Starts@, and Ends@. This feature is also accessible through the context menu of the Task List window.

Font: Fixed / Proportional

Selects which type of font to use in List windows. The Plug-in uses either the Fixed or Proportional font of the Embedded Workbench's Common Fonts. The Common Fonts are configured under **Tools .. Options .. Common Fonts.** Font changes only take effect the next time a window is re-opened, or when a new font is applied through **Tools** .. **Options .. Common Fonts**. A smaller font means you can have smaller windows, which is important on limited screen space. The following fonts are recommended:

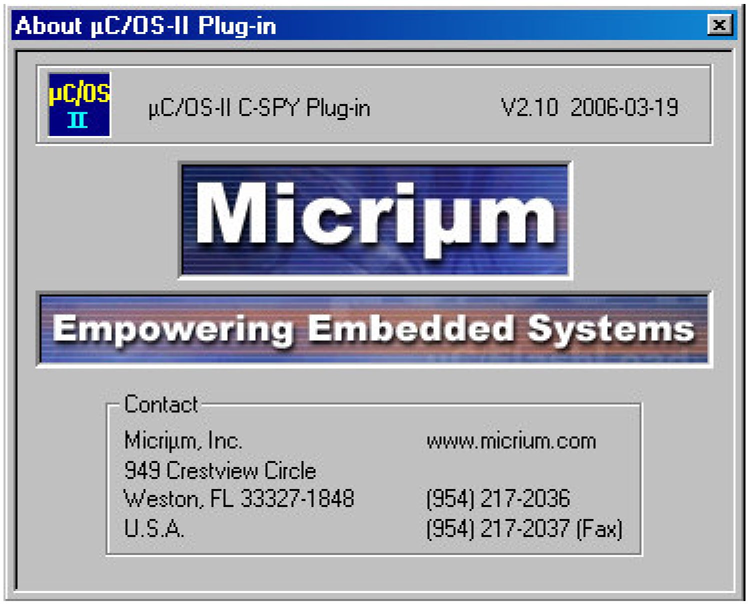
Fixed: `Courier New 9pt`
Proportional: MS Sans Serif 8pt

Apply

Click **Apply** to apply the changes.

About

This window shows information about the µC/OS-II Kernel Awareness Plug-in.



µC/OS-II C-SPY Plug-in V2.10 2006-03-19

Contact

Micriµm, Inc.
949 Crestview Circle
Weston, FL 33327-1848
U.S.A.

www.micrium.com

(954) 217-2036
(954) 217-2037 (Fax)

Information

Version
Current version of the µC/OS-II Kernel Awareness Plug-in.

Contact
How to contact the Micriµm.

Bibliography

µC/OS-II, The Real-Time Kernel, 2nd Edition
Jean J. Labrosse
R&D Technical Books, 2002
ISBN 1-57820-103-9

Contacts

Micriµm, Inc.
949 Crestview Circle
Weston, FL 33327-1848
954-217-2036
954-217-2037 (FAX)
e-mail: Jean.Labrosse@Micrium.com
WEB: www.Micrium.com

IAR Systems, Inc.
Century Plaza
1065 E. Hillsdale Blvd
Foster City, CA 94404
USA
+1 650 287 4250
+1 650 287 4253 (FAX)
WEB: http://www.IAR.com
e-mail: info@IAR.com

CMP Books, Inc.
1601 W. 23rd St., Suite 200
Lawrence, KS 66046-9950
USA
+1 785 841 1631
+1 785 841 2624 (FAX)
WEB: http://www.rdbooks.com
e-mail: rdorders@rdbooks.com