

OS Awareness Manual RTEMS

Release 02.2025



TRACE32 Online Help		
TRACE32 Directory		
TRACE32 Index		
TRACE32 Documents		
OS Awareness Manuals		
OS Awareness Manual RTEMS		1
Overview		4
Terminology		4
Brief Overview of Documents for New Users		5
Supported Versions		5
Configuration		6
Quick Configuration Guide		6
Hooks & Internals in RTEMS		7
Features		8
Display of Kernel Resources		8
Task Stack Coverage		8
Task-Related Breakpoints		9
Dynamic Task Performance Measurement		10
Task Runtime Statistics		11
Task State Analysis		12
Function Runtime Statistics		13
RTEMS specific Menu		15
RTEMS Commands		16
TASK.INFO	Display API information	16
TASK.INTernal.Mutex	Display internal mutexes	16
TASK.INTernal.Thread	Display internal threads	17
TASK.Posix.CondVar	Display POSIX condition variables	17
TASK.Posix.Mutex	Display POSIX mutexes	18
TASK.CLassic.Extension	Display RTEMS extensions	18
TASK.CLassic.MsgQueue	Display RTEMS message queues	19
TASK.CLassic.Partition	Display RTEMS partitions	19
TASK.CLassic.PEriod	Display RTEMS periods	20
TASK.CLassic.POrt	Display RTEMS ports	20
TASK.CLassic.Region	Display RTEMS regions	21
TASK.CLassic.Semaphore	Display RTEMS semaphores	21
TASK.CLassic.Task	Display RTEMS tasks	22
TASK.CLassic.TImer	Display RTEMS timers	23

TASK.Thread	Display all threads	23
RTEMS PRACTICE Functions		25
TASK.CONFIG()	OS Awareness configuration information	25
TASK.CLassic.TASKMAX()	Max. number of tasks	25
TASK.CLassic.TASKLIST()	RTEMS task list	25
TASK.CLassic.TASKNAME()	Name of RTEMS task	26

OS Awareness Manual RTEMS

Version 13-Feb-2025

Overview

▲ Simulator	
File Edit View Var Break Run CPU Misc Trace Perf Cov MPC85XX/QorlQ RTEMS Wind ▶ ▶ 🛺 ↓ ✔ ⊄ ▶ Ⅱ ? №? ◎ ± 20	ow Help 1988 🔲 🚳 📾 🚳 🔹 🔑
B::TERM.gate B::TASK.CLassic.Semapho Meillo, world! More and a second	me Attributes count waiting holder IO PR:BI:IN 0. 0. (none) mi PR:BI:IN 0. 0. (none) ia PR:BI:IN 0. 0. 0004F3A8 SHLL oa PR:BI:IN 0. 0. (none) xa SB 0. 0. (none) MA DEFAULT 0. 2. (none)
IDLE 00054F60 00056F58 00056L88 2% 00056D8 00001E78 4% UI1 00057088 0005880 5% 00058505 0001100 13% TSKA 00059890 00058508 0005850 00001000 13%	B:TASK.INFO
magic object id name state prio api entry intern 0004D458 0901001 IDLE running 255. intern 00001478 Thread_Idle_body int 0004DA08 04010001 UDLE running 255. intern 00001478 Thread_Idle_body int 0004D428 0A010001 UIL wevent 2. RTEMS 00001664 Init 0004E48 0A010003 TSKA wsemaphore 1. RTEMS 00001414 task_c 0004E588 0A010004 TSKD wmutex 1. RTEMS 0000147C task_cd 0004E588 0A010005 TSKE suspended 1. RTEMS 0000147C task_cd 0004F348 0A010006 SHLL wtime 3. RTEMS 0001530 rtems_shell_task 00051FF8 08010001 task_ub wmutex 1. POSIX 00014AC task_b	timers 8. semaphores 15. message queues 8. partitions 8. regions 8. ports (not present) periods 8. extensions 0.
B : : TASK . INFO Thread INTernal CLassic Posix IDLE	stopped MIX UP

The OS Awareness for RTEMS contains special extensions to the TRACE32 Debugger. This manual describes the additional features, such as additional commands and statistic evaluations.

Terminology

RTEMS uses the terms "tasks" and "threads". If not otherwise specified, the TRACE32 term "task" corresponds to both, RTEMS tasks and threads.

Architecture-independent information:

- **"Debugger Tutorial**" (debugger_tutorial.pdf): Get familiar with the basic features of a TRACE32 debugger.
- "General Commands" (general_ref_<x>.pdf): Alphabetic list of debug commands.
- "OS Awareness Manuals" (rtos_<os>.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

Architecture-specific information:

- "Processor Architecture Manuals": These manuals describe commands that are specific for the processor architecture supported by your Debug Cable. To access the manual for your processor architecture, proceed as follows:
 - Choose Help menu > Processor Architecture Manual.

Supported Versions

Currently RTEMS is supported for the following versions:

• RTEMS 4.6 to 5.3 on ARM, ColdFire, NIOS-II, PowerPC and RISC-V

The **TASK.CONFIG** command loads an extension definition file called "rtems.t32" (directory "~~/demo/*<processor*/kernel/rtems"). It contains all necessary extensions.

TASK.CONFIG ~~/demo/<arch>/kernel/rtems/rtems.t32

The configuration tries to locate the RTEMS internals automatically. For this purpose, the kernel symbols must be loaded and accessible at any time the OS Awareness is used (see also "Hooks & Internals").

If you want to have dual port access for the display functions (display "On The Fly"), you have to map emulation or shadow memory to the address space of all used system tables.

If you want to display the OS objects "On The Fly" while the target is running, you need to have access to memory while the target is running. Enable **SYStem.MemAccess**.

Quick Configuration Guide

To get a quick access to the features of the OS Awareness for RTEMS with your application, follow this roadmap:

- 1. Start the TRACE32 Debugger.
- 2. Load your application as normal.
- 3. Execute the command:

TASK.CONFIG ~~/demo/<arch>/kernel/rtems/rtems.t32

See "Configuration".

4. Execute the command:

MENU.ReProgram ~~/demo/<arch>/kernel/rtems/rtems.t32

See "RTEMS Specific Menu".

5. Start your application.

Now you can access the RTEMS extensions through the menu.

In case of any problems, please carefully read the previous Configuration chapters.

No hooks are used in the kernel.

For retrieving the kernel data and structures, the OS Awareness uses the global kernel symbols and structure definitions. Ensure that access to those structures is possible every time when features of the OS Awareness are used.

The OS Awareness for RTEMS supports the following features.

Display of Kernel Resources

The extension defines new commands to display various kernel resources. Information on the following RTEMS components can be displayed:

TASK.CLassic.Extension	RTEMS extensions
TASK.CLassic.MsgQueue	RTEMS message queues
TASK.CLassic.Partition	RTEMS partitions
TASK.CLassic.PEriod	RTEMS periods
TASK.CLassic.POrt	RTEMS ports
TASK.CLassic.Region	RTEMS regions
TASK.CLassic.Semaphore	RTEMS semaphores
TASK.CLassic.Task	RTEMS tasks
TASK.CLassic.Tlmer	RTEMS timers
TASK.INFO	API information
TASK.INTernal.Mutex	Internal mutexes
TASK.INTernal.Thread	Internal threads
TASK.Posix.CondVar	POSIX condition variables
TASK.Posix.Mutex	POSIX mutexes
TASK.Thread	All threads

For a description of the commands, refer to chapter "RTEMS Commands".

If your hardware allows memory access while the target is running, these resources can be displayed "On The Fly", i.e. while the application is running, without any intrusion to the application.

Without this capability, the information will only be displayed if the target application is stopped.

Task Stack Coverage

For stack usage coverage of tasks, you can use the **TASK.STacK** command. Without any parameter, this command will open a window displaying with all active tasks. If you specify only a task magic number as parameter, the stack area of this task will be automatically calculated.

To use the calculation of the maximum stack usage, a stack pattern must be defined with the command **TASK.STacK.PATtern** (default value is zero).

To add/remove one task to/from the task stack coverage, you can either call the **TASK.STacK.ADD** or **TASK.STacK.ReMove** commands with the task magic number as the parameter, or omit the parameter and select the task from the **TASK.STacK.*** window.

It is recommended to display only the tasks you are interested in because the evaluation of the used stack space is very time consuming and slows down the debugger display.

name lo		high	sp	- %	lowest	spare	max 0	10
IDLE 0	07E8AB8	007EAAB8	007EAA98	0%	007EAA59	00001FA1	1% +	
UI1 0	07E6908	007E8908	007E8880	1%	007E8801	00001EF9	3% -	
ST1 0	07E43C8	007E63C8	007E6330	1%	007E62F9	00001F31	2% -	
ST2 0	07E1E88	007E3E88	007E3E10	1%	007E3DD9	00001F51	2% -	
ST3 0	07DF948	007E1948	007E18B0	1%	007E18A1	00001F59	2% -	
MOT1 0	07DD408	007DF408	007DF358	2%	007DF331	00001F29	2% -	
MOT2 0	07DAEC8	007DCEC8	007DCF18	2%	007DCDE1	00001F29	2% -	

Task-Related Breakpoints

Any breakpoint set in the debugger can be restricted to fire only if a specific task hits that breakpoint. This is especially useful when debugging code which is shared between several tasks. To set a task-related breakpoint, use the command:

Break.Set <address>|<range> [/<option>] /TASK <task> Set task-related breakpoint.

- Use a magic number, task ID, or task name for *<task>*. For information about the parameters, see "What to know about the Task Parameters" (general_ref_t.pdf).
- For a general description of the **Break.Set** command, please see its documentation.

By default, the task-related breakpoint will be implemented by a conditional breakpoint inside the debugger. This means that the target will *always* halt at that breakpoint, but the debugger immediately resumes execution if the current running task is not equal to the specified task.

NOTE: Task-related breakpoints impact the real-time behavior of the application.

On some architectures, however, it is possible to set a task-related breakpoint with *on-chip* debug logic that is less intrusive. To do this, include the option **/Onchip** in the **Break.Set** command. The debugger then uses the on-chip resources to reduce the number of breaks to the minimum by pre-filtering the tasks.

For example, on ARM architectures: *If* the RTOS serves the Context ID register at task switches, and *if* the debug logic provides the Context ID comparison, you may use Context ID register for less intrusive task-related breakpoints:

Break.CONFIG.UseContextID ON	Enables the comparison to the whole Context ID register.
Break.CONFIG.MatchASID ON	Enables the comparison to the ASID part only.
TASK.List.tasks	If TASK.List.tasks provides a trace ID (traceid column), the debugger will use this ID for comparison. Without the trace ID, it uses the magic number (magic column) for comparison.

When single stepping, the debugger halts at the next instruction, regardless of which task hits this breakpoint. When debugging shared code, stepping over an OS function may cause a task switch and coming back to the same place - but with a different task. If you want to restrict debugging to the current task, you can set up the debugger with **SETUP.StepWithinTask ON** to use task-related breakpoints for single stepping. In this case, single stepping will always stay within the current task. Other tasks using the same code will not be halted on these breakpoints.

If you want to halt program execution as soon as a specific task is scheduled to run by the OS, you can use the **Break.SetTask** command.

😢 B::Break.List					×
X Delete All 🔿 Disable All 🔘 Enable A	I 🛇 Init 🖉	Method 😤 Store	🔀 Load	🙋 Set	
address type	method task				
C:000005B8 Program	SOFT ST1		V Ø rte	ems_event_send	A
C:000005E8 Program	SOFT ST2			ems_event_receive	
C:0000A788 Program	SOFT		V Ø _Th	nread_Idle_body	
					\sim
<					>
,					

Dynamic Task Performance Measurement

The debugger can execute a dynamic performance measurement by evaluating the current running task in changing time intervals. Start the measurement with the commands **PERF.Mode TASK** and **PERF.Arm**, and view the contents with **PERF.ListTASK**. The evaluation is done by reading the 'magic' location (= current running task) in memory. This memory read may be non-intrusive or intrusive, depending on the **PERF.METHOD** used.

If **PERF** collects the PC for function profiling of processes in MMU-based operating systems (SYStem.Option.MMUSPACES ON), then you need to set **PERF.CONFIG.MMUSPACES**, too.

For a general description of the **PERF** command group, refer to "General Commands Reference Guide **P**" (general_ref_p.pdf).

E B::PER	F.ListTASK							•	×
🌽 Setup	🔛 Config 📭	Goto	🗾 Detailed	🔍 View 🛛	IIIII Profile	🛇 Init	O DISable	A (0)	rm
					runt	ime: 10	0%		
name	ratio	1%	2%	5%	10%	209	*	50%	
IDLE	58.626%							_	~
SHLL	41.374%								
UI1	0.000%	i							
TSKA	0.000%	1							
TSKC	0.000%	1							
TSKD	0.000%	1							
TSKE	0.000%	i							\mathbf{v}
<		Ċ.						>	

Task Runtime Statistics

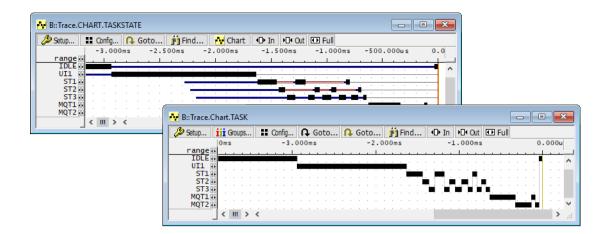
NOTE:	This feature is only available, if your debug environment is able to trace task
	switches (program flow trace is not sufficient). It requires either an on-chip trace
	logic that is able to generate task information (eg. data trace), or a software
	instrumentation feeding one of TRACE32 software based traces (e.g. FDX or
	LOGGER). For details, refer to "OS-aware Tracing" in TRACE32 Concepts,
	page 36 (trace32_concepts.pdf).

Based on the recordings made by the **Trace** (if available), the debugger is able to evaluate the time spent in a task and display it statistically and graphically.

To evaluate the contents of the trace buffer, use these commands:

Trace.List List.TASK DEFault	Display trace buffer and task switches
Trace.STATistic.TASK	Display task runtime statistic evaluation
Trace.Chart.TASK	Display task runtime timechart
Trace.PROfileSTATistic.TASK	Display task runtime within fixed time intervals statistically
Trace.PROfileChart.TASK	Display task runtime within fixed time intervals as colored graph
Trace.FindAll Address TASK.CONFIG(magic)	Display all data access records to the "magic" location
Trace.FindAll CYcle owner OR CYcle context	Display all context ID records

The start of the recording time, when the calculation doesn't know which task is running, is calculated as "(unknown)".



Task State Analysis

NOTE:	This feature is <i>only</i> available, if your debug environment is able to trace task switches and data accesses (program flow trace is not sufficient). It requires either an on-chip trace logic that is able to generate a data trace, or a software instrumentation feeding one of TRACE32 software based traces (e.g. FDX or LOGGER). For details, refer to "OS-aware Tracing" in TRACE32 Concepts,
	page 36 (trace32_concepts.pdf).

The time different tasks are in a certain state (running, ready, suspended or waiting) can be evaluated statistically or displayed graphically.

This feature requires that the following data accesses are recorded:

- All accesses to the status words of all tasks
- Accesses to the current task variable (= magic address)

Adjust your trace logic to record all data write accesses, or limit the recorded data to the area where all TCBs are located (plus the current task pointer).

Example: This script assumes that the TCBs are located in an array named TCB_array and consequently limits the tracing to data write accesses on the TCBs and the task switch.

```
Break.Set Var.RANGE(TCB_array) /Write /TraceData
Break.Set TASK.CONFIG(magic) /Write /TraceData
```

To evaluate the contents of the trace buffer, use these commands:

Trace.STATistic.TASKState	Display task state statistic
Trace.Chart.TASKState	Display task state timechart

The start of the recording time, when the calculation doesn't know which task is running, is calculated as "(unknown)".

🎾 Setup	Confi	g	R	G	oto		Ê	j Fi	nd.		1	C	hart	:	•□•	In	۲D	۹ Ou	t 🖪	Þ	Full									
	-3.	000) m s		-)	2.5	00	ns		- 2	2.0	00	15		-	1.5	00	ns		-1	. 00) () m	s	- !	500	.0	00u	s	0.	0
range	1					1					1					1					1					1			 1	
IDLE 🕕																													-	,
UI1 II ST1 II																	•	-				•	-						-	
ST2 0											1	1	1	1			÷.												_	
ST3 II									1														1			•	•			
MOT1 III														1			1		×			-								
1.061 7.00														1.0																

Function Runtime Statistics

NOTE: This feature is *only* available, if your debug environment is able to trace task switches (program flow trace is not sufficient). It requires either an on-chip trace logic that is able to generate task information (eg. data trace), or a software instrumentation feeding one of TRACE32 software based traces (e.g. **FDX** or **LOGGER**). For details, refer to "**OS-aware Tracing**" in TRACE32 Concepts, page 36 (trace32_concepts.pdf).

All function-related statistic and time chart evaluations can be used with task-specific information. The function timings will be calculated dependent on the task that called this function. To do this, in addition to the function entries and exits, the task switches must be recorded.

To do a selective recording on task-related function runtimes based on the data accesses, use the following command:

```
; Enable flow trace and accesses to the magic location Break.Set TASK.CONFIG(magic) /TraceData
```

To do a selective recording on task-related function runtimes, based on the Arm Context ID, use the following command:

```
; Enable flow trace with Arm Context ID (e.g. 32bit) ETM.ContextID 32
```

To evaluate the contents of the trace buffer, use these commands:

Trace.ListNesting	Display function nesting
Trace.STATistic.Func	Display function runtime statistic
Trace.STATistic.TREE	Display functions as call tree

Trace.STATistic.sYmbol /SplitTASK	Display flat runtime analysis
Trace.Chart.Func	Display function timechart
Trace.Chart.sYmbol /SplitTASK	Display flat runtime timechart

The start of the recording time, when the calculation doesn't know which task is running, is calculated as "(unknown)".

		Nesting	Chart					
s: 30	03. total: 3.779ms 70 workard 3785166	ounds						
je: -5	tree	total	min	max	avr	count	1%	
•	(root)	355.400us	-	355,400us	-	-	4	
	(root)	262.900us	-	262,900us	-	-	1 A	
	(root)	377.700us	-	377.700us	-	-	i i	
		0.800us	0.800us	0.800us	0.800us	1.	+	
	—⊟ _Thread_Handler	361.800us	-	361.800us	-	1.(0/1)	4	
	CPU_ISR_Set_level	1.700us	1.700us	1.700us	1.700us	1.	+	
	— _User_extensions_Thread_begin	6.700us	6.700us	6.700us	6.700us	1.	+	
	—⊕ _Thread_Enable_dispatch	9.600us	9.600us	9.600us	9.600us	1.	+	
	—⊟ semtask3	339.400us	-	339.400us	-	1. (0/1)	+	
	— → rtems_semaphore_release	314.000us	22.600us	59.400us	52.333us	6.	+	
	— mems_task_suspend	22.000us	-	22.000us	-	1.(0/1)	+	
L	🗏 (root)	358.800us		358.800us	-	-	+	
-		0.800us	0.800us	0.800us	0.800us	1.	+	
	- Thread_Handler	342.900us		342.900us		1. (0/1)	•	
-	CPU_ISR_Set_level	1.700us	1.700us	1.700us	1.700us	1.	+	
-	User_extensions_Thread_begin	6.700us	6.700us	6.700us	6.700us	1.	†	
-	— Inread_Enable_dispatch	9.600us	9.600us	9.600us	9.600us	1. 1.(0/1)	+	
L	└── msgqtask1 └── rtems_message_queue_create	320.500us 96.400us	48,200us	320.500us 48.200us	48,200us	2.		
_	—⊕rtems_message_queue_create	129,400us			43.133us	2.	1	
2	₩ B::Trace.Chart.FUNC	🔒 Goto	🛉 Find 🐠	In II Out E	🗈 Full			
2	Setup iii Groups II Config A Goto	Goto -1.000		In ▶□• Out I 900.000u s		00.000u s	-700.000u	
-	Setup jji Goups II Confg Q Goto ange is _semaphore_return_code@512 (s)					00.000us		
_	Setup jii Goups II Config Q Goto Oms range 					00.000us		
_	Setup iii Goups II Config Q Goto Oms semaphore_return_code%51288 eize_interrupt_trylock%51288					00.000us		
	Setup jji Goups II Config G Goto range 80 semaphore_return.code0512 80 eize_interrupt_trylock0512 80 ize_interrupt_blocking0512 80 CORE_muder_0513 80					00.000us		
	Setup iii Goups ii Config G Goto oms range semaphore_return_codeS12 ize_interrupt_trylockes72 LCORE_mutex_Surrender8513 Core_mutex_return_codeS1336					00.000us		
	Setup jji Goups II Config C Goto range 30 semaphore_return_code@s12 80 ize_interrupt_trylockes12 80 CORE_mutex_Surrender@s13 80 core_mutex_return_code@s13 80 core_mutex_return_code@s13 280					00.000u s		
	Setup jii Goups : Config Goto range 88 semaphore_return_code@512 89 eize_interrupt_trylock@512 89 CORE_mutex_Surrender@513 89 core_mutex_return_code@512 89 core_mutex_return_code@512 89 core_mutex_spend@511 89					00.000us		
	Setup jji Goups :: Confg C Goto range:8 semaphore_return_code@ST2 :: eize_interrupt_blocking@ST2 :: CORE_mutex_Surrender@ST3 :: CORE_mutex_return_code@ST3 :: core_mutex_return_code@ST2 :: end\rtems_task_suspend@ST1 :: asksuspend_ThreaG_Get@ST1 ::					00.000us		
	Setup jii Goups : Config C Goto oms range 88 eize_interrupt_trylocks512 88 ize_interrupt_blocking8512 88 					10.000us		
	Setup jii Goups :: Confg C Goto range:8 -semaphore_return_codeSist ize_interrupt_trylocksigeSize -core_mutex_return_codeSist core_mutex_return_codeSist end_rtems_task_suspendSist uspend_Thread_Get85118 uspend_Thread_SuspendESit end_rtems_task_spendESit setup: task_spendESit setup: task_spendESit tasksusp					00.000us		
	Setup jji Goups :: Confg C Goto oms semaphore_return_code@ST2 :: ize_interrupt_trylock@ST2 :: LCORE_mutex_Surrender@ST3 :: core_mutex_return_code@ST3 :: core_mutex_return_code@ST3 :: end_rtems_task_suspend@ST1 :: asksuspend_Thread_Get@ST1 :: end_rtems_task_suspend@ST1 :: end_rtems_task_s					00.000us		
	Setup jii Goups :: Confg C Goto range iX _semaphore_return_codeST2 :: ize_interrupt_trylocksT2 :: _CORE_mutex_SurrenderST3 :: _core_mutex_return_codeST3 :: core_mutex_return_codeST3 :: core_mutex_return_codeST3 :: core_mutex_suprenderST3 :: end_rtems_task_suspendST1 :: uspend_Thread_Get8ST1 :: uspend_Thread_SuppendSST2 :: asksuspend_Thread_Get8ST2 :: uspend_Thread_Get8ST2 :: uspend_Thread_Get8ST2 :: uspend_Thread_Get8ST2 :: uspend_Thread_Get8ST2 ::					00.000us		
	Setup jji Goups :: Config C Goto range ?? semaphore_return_code@s12 ?? ize_interrupt_trylock@s12 ?? CORE_mutex_surrender@s13 ?? core_mutex_return_code@s13 ?? core_mutex_return_code@s12 ?? end/rtems_task_suspend@s11 ?? asksuspend_Thread_Get@s11 ?? uspend_Thread_Get@s11 ?? asksuspend@s12 ?? end/rtems_task_suspend@s12 ?? uspend_Thread_Suspend@s12 ?? uspend_Thread_Suspend@s13 ???					00.000us		
	Setup jii Goups : Confg Costo oms range & eize_interrupt_trylocksT2 & ize_interrupt_blockingST2 & 					10.000us		
	Setup jii Goups :: Confg C Goto range:# semaphore_return_code@ST2 eize_interrupt_trylock@ST2 ize_interrupt_trylocking@ST2 CORE_mutex_Surrender@ST3 CORE_mutex_suspend@ST3 core_mutex_return_code@ST2 end\rtems_task_suspend@ST1 asksuspend_Thread_Get@ST2 asksuspend_Thread_Suspend@ST2 asksuspend_Thread_Suspend@ST2 asksuspend_Thread_Suspend@ST3 asksuspend					00.000us		
	Setup jii Goups : Confg Costo oms range & eize_interrupt_trylocksT2 & ize_interrupt_blockingST2 & 					10.000us		

The menu file "rtems.men" contains a menu with RTEMS specific menu items. Load this menu with the **MENU.ReProgram** command.

You will find a new menu called **RTEMS**.

- The **Display** menu items launch the kernel resource display windows.
- The **Stack Coverage** submenu starts and resets the RTEMS specific stack coverage and provides an easy way to add or remove tasks from the stack coverage window.

In addition, the menu file (*.men) modifies these menus on the TRACE32 main menu bar:

- The **Trace** menu is extended. In the **List** submenu, you can choose if you want a trace list window to show only task switches (if any) or task switches together with default display.
- The **Perf** menu contains additional submenus for task runtime statistics and statistics on task states.

▲ TRACE32 for RTEMS File Edit View Var Break Run CPU Misc Trace Perf Cov MPC8	− □ ×
▶ ₩ ▲ ↓ ← Ċ ▶ Ⅲ № ? №? ▶ ₩ ▲ ↓ ← Ċ ▶ Ⅲ ▲ B::TASK.Thread	Display API Information Display All Threads Display Internal Objects
magic object id name state 007ECLES 0901001 IDLE runni 007EB7D8 0A010001 UI1 suspe 007EB7D8 0A010002 ST1 suspe 007EB7B8 0A010003 ST2 suspe 007EB7B8 0A010004 ST3 suspe 007EB878 0A010005 MQT1 suspe 007EB878 0A010005 MQT1 suspe 007EB058 0A010006 MQT2 suspe 007EB058 0A010006 MQT2 suspe 035play Partitions Display Regions	Display Classic Objects Display POSIX Objects Stack Coverage gqtask1 gqtask2
B : : TASK. Display Ports Display Periods Display Extensions INFO Thread INTernal CLassic Point Stack SP00004788 \\sample\threaddebody IT IDLE	previous MIX UP

TASK.INFO

Format:

TASK.INFO[.INTernal | .CLassic | .Posix | .Itron]

Displays the RTEMS API information tables.

Without any argument, a tree with all available API information will be shown. Specify an API to show the configured objects for this API.

Without any arguments, a table with all created processes will be shown. Specify a process name, ID or magic number to display detailed information on that process.

🖧 B::TASK.INF	b		×
api	objects	max	
internal	threads	1.	~
🗉 classic	mutexes tasks	1.	
-	timers	(not present)	
	semaphores	12.	
	message queues	3.	
	partitions	(not present)	
	regions	1.	
	ports	(not present)	
	periods	(not present)	
- BOCTV	extensions	1.	
POSIX ITRON			
∃ ITRON	1		~
<		>	- 18

The fields "api" and "objects" are mouse sensitive, double clicking on them opens appropriate windows. Right clicking on them will show a local menu.

TASK.INTernal.Mutex

Display internal mutexes

Format:

TASK.INTernal.Mutex [<mutex>]

Displays a table with all mutexes of the internal API or detailed information about one specific mutex.

Without any arguments, a table with all created mutexes will be shown. Specify a mutex name, ID or magic number to display detailed information on that mutex.

"magic" is a unique ID, used by the OS Awareness to identify a specific mutex (address of the mutex control structure).

The fields "magic" and "holder" are mouse sensitive. Double-clicking on them opens appropriate windows. Right clicking on them will show a local menu.

🔏 B::TASK.	INTernal.Mutex	
magic	object id name	waiting holder
0004D820 0004D84C	11010001 11010002	0. (none) 0. (none)
<		×

TASK.INTernal.Thread

Display internal threads



Displays the thread table with all threads of the internal API or detailed information about one specific thread.

The display is identical to TASK.Thread, but shows only internal threads.

🔏 B::TASK.	INTernal.Threa	ad						×
magic	object id	name	state	prio	api	entry		
007EC1E8	09010001	IDLE	running	255.	intern	0000A788	_Thread_Idle_body	^
								\mathbf{v}
<							>	

TASK.Posix.CondVar

Display POSIX condition variables

Format: TASK.Posix.CondVar [<cond_var>]</cond_var>	
---	--

Displays the condition variable table with all condition variables of the POSIX API or detailed information about one specific condition variable.

Without any arguments, a table with all created condition variables will be shown. Specify a condvar name, ID or magic number to display detailed information on that condition variable.

"magic" is a unique ID, used by the OS Awareness to identify a specific condition variable (address of the condition variable control structure).

The fields "magic" and "mutex id" are mouse sensitive. Double-clicking on them opens appropriate windows. Right clicking on them will show a local menu.

Format: TASK.Posix.Mutex [<mutex>]

Displays the mutex table with all mutexes of the POSIX API or detailed information about one specific mutex.

Without any arguments, a table with all created mutexes will be shown. Specify a mutex name, ID or magic number to display detailed information on that mutex.

"magic" is a unique ID, used by the OS Awareness to identify a specific mutex (address of the mutex control structure).

The fields "magic" and "holder" are mouse sensitive. Double-clicking on them opens appropriate windows. Right clicking on them will show a local menu.

🔏 B::TASK.	Posix.Mutex					×
magic 00055540	object id n 33010001	name	waiting 2.	holder 0004E910	тѕкс	\$
<<						>

TASK.CLassic.Extension

Display RTEMS extensions

Format:

TASK.CLassic.Extension [<extension>]

Displays the extension table with all extensions of the RTEMS API or detailed information about one specific extension.

Without any arguments, a table with all created extensions will be shown. Specify a extension name, ID or magic number to display detailed information on that extension.

🔒 B::TASK.CLassic.Extension						
magic 007EAC70	object id	name	switch	callouts create start delete		
00/EAC/0	44010001	1		ference scare acreec	Q	
<				>		

"magic" is a unique ID, used by the OS Awareness to identify a specific extension (address of the extension control structure).

The fields "magic" and "callouts" are mouse sensitive. Double-clicking on them opens appropriate windows. Right clicking on them will show a local menu.

Display RTEMS message queues TASK.CLassic.MsqQueue

```
Format: TASK.CLassic.MsgQueue [<queue>]
```

Displays the message queue table with all message queues of the RTEMS API or detailed information about one specific message queue.

Without any arguments, a table with all created message queues will be shown. Specify a message queue name, ID or magic number to display detailed information on that message queue.

	CLassic.MsgQ							
magic	object id	name	qtype	num	max	msgsize	waiting	
007EB490	22010001	MSQ1	FIFO	0.	2.	16.	0.	
007EB518	22010002	MSQ2	FIFO	0.	2.	16.	0.	1 ^{**}
								· •
-								

"magic" is a unique ID, used by the OS Awareness to identify a specific message queues (address of the message queues control structure).

The fields "magic" and "threads" are mouse sensitive. Double-clicking on them opens appropriate windows. Right clicking on them will show a local menu.

TASK.CLassic.Partition

Display RTEMS partitions

Format: TASK.CLassic.Partition [<partition>]

Displays the partition table with all partitions of the RTEMS API or detailed information about one specific partition.

Without any arguments, a table with all created partitions will be shown. Specify a partition name, ID or magic number to display detailed information on that partition.

"magic" is a unique ID, used by the OS Awareness to identify a specific partition (address of the partition control structure).

The fields "magic" and "address" are mouse sensitive. Double-clicking on them opens appropriate windows. Right clicking on them will show a local menu.

🖧 B::TASK.	CLassic.Partiti	on					×
<u>magic</u> 000516B0	object id 2A010001	name PART	used 0.	max 16.	buffsize 128.	address 00048FB0	•
<<						>	

TASK.CLassic.PEriod

Display RTEMS periods

	Format:	TASK.CLassic.PEriod [<period>]</period>	
--	---------	--	--

Displays the period table with all rate monotonic periods of the RTEMS API or detailed information about one specific period.

Without any arguments, a table with all created periods will be shown. Specify a period name, ID or magic number to display detailed information on that period.

"magic" is a unique ID, used by the OS Awareness to identify a specific period (address of the rate monotonic control structure).

The fields "magic" and "owner" are mouse sensitive. Double-clicking on them opens appropriate windows. Right clicking on them will show a local menu.

🖧 B::TASK.	CLassic.PEriod	ł				, •		2
magic 00051DC0	object id 42010001	name RATE	state incactive	1ength 00000000	owner 0004DE80	UI1	_	\$
<							>	

TASK.CLassic.POrt

Display RTEMS ports

	Format:	TASK.CLassic.POrt [<port>]</port>	
--	---------	------------------------------------	--

Displays the port table with all dual ported memories of the RTEMS API or detailed information about one specific dual ported memory.

Without any arguments, a table with all created ports will be shown.

Specify a port name, ID or magic number to display detailed information on that port.

"magic" is a unique ID, used by the OS Awareness to identify a specific port (address of the dual ported memory control structure).

The fields "magic" and "internal" are mouse sensitive. Double-clicking on them opens appropriate windows. Right clicking on them will show a local menu.

TASK.CLassic.Region

Display RTEMS regions

Format:

TASK.CLassic.Region [<region>]

Displays the region table with all regions of the RTEMS API or detailed information about one specific region.

Without any arguments, a table with all created regions will be shown. Specify a region name, ID or magic number to display detailed information on that region.

	tems.CLassic.	2	ACF0					×
	object id		length	pagesize	used	address	qtype waiting	
007EACF0	32010001	HEAP	007ABC00	0000008	10.	00022C00	FIFO 0.	^
	cond var	id node						
32010001	1.	1.	RTEMS					
Threads wa	aiting							
								~
<								>

"magic" is a unique ID, used by the OS Awareness to identify a specific region (address of the region control structure).

The fields "magic" and "address" are mouse sensitive. Double-clicking on them opens appropriate windows. Right clicking on them will show a local menu.

TASK.CLassic.Semaphore

Display RTEMS semaphores

Format:

TASK.CLassic.Semaphore [<semaphore>]

Displays the semaphore table with all semaphores of the RTEMS API or detailed information about one specific semaphore.

Without any arguments, a table with all created semaphores will be shown.

Specify a semaphore name, ID or magic number to display detailed information on that semaphore.

🔏 B::TASK.	CLassic.Semap	hore							×					
magic	object id			qtype	count	waiting	holder							
007EAE60		LBIO		INHERIT	0.	0.	(none)		~					
007EAED8		TRmi		INHERIT	0.	0.	(none)							
007EAF50		LBI		INHERIT	0.	0.	(none)							
007EAFC8		TRia		INHERIT	0.	0.	(none)							
007EB040	1A010005	TRoa		INHERIT	0.	0.	(none)							
007EB0B8		TRxa		FIFO	1.		007EC1E8	IDLE						
007EB130 007EB1A8				INHERIT	0. 0.	0.	(none)							
007EB1A8		LBI ^S SEM1	binary counting	INHERIT	0.	0.	(none)							
007EB220		SEM2		FIF			1							
007EB310		SEM3		FIF 💑	B::TASK.rf	tems.CLassi	ic.Semaphor	e "SEM3"						x
COT EBSIO	THOTOGOD	52115	simple											
15				mag		object i 1A01000B		type	qtyp FIF0		t waiting 0.	007EB938	ST2	
				- 007	EDDIO	TAOTOOD	SEMP	simple	FIFU	1.	0.	00755329	512	^
				obj	ect id	cond va	r id node	e id ani						
					1000B	11.	1.		15					-
				blo	cked 1	ocked	behaviour							
				2		0.	blocks							-
					y owner	release								-
				no										
				Inr	eads wa	ITTING								
				<									>	

"magic" is a unique ID, used by the OS Awareness to identify a specific semaphore (address of the semaphore control structure).

The fields "magic" and "threads" are mouse sensitive. Double-clicking on them opens appropriate windows. Right clicking on them will show a local menu.

TASK.CLassic.Task

Display RTEMS tasks

Format: TASK.CLassic.Task [<task>]

Displays the task table with all tasks of the RTEMS API or detailed information about one specific task.

The display is identical to **TASK.Thread**, but shows only RTEMS tasks.

Without any arguments, a table with all created tasks will be shown. Specify a task name, ID or magic number to display detailed information on that task.

🖧 B::TASK.	CLassic.Task					[x
magic	object id	name	state	prio	api	entry		
007EB678	0A010001	UI1	suspended	1.	RTEMS	000003A0	Init	~
007EB7D8	0A010002	ST1	suspended	2.	RTEMS	0000008C	semtask1	~
007EB938	0A010003	ST2	suspended	3.	RTEMS	00000188	semtask2	
007EBA98	0A010004	ST3	suspended	4.	RTEMS	000001E8	semtask3	
007EBBF8	0A010005	MQT1	suspended	5.	RTEMS	00000260	msggtask1	
007EBD58	0A010006	MOT2	suspended	6.	RTEMS	00000328	msootask2	
							21	\mathbf{v}
<							<pre>\</pre>	

"magic" is a unique ID, used by the OS Awareness to identify a specific task (address of the thread control structure).

The fields "magic" and "entry" are mouse sensitive. Double-clicking on them opens appropriate windows. Right clicking on them will show a local menu.

Format: **TASK.CLassic.TImer** [<timer>]

Displays the timer table with all timers of the RTEMS API or detailed information about one specific timer.

Without any arguments, a table with all created timers will be shown. Specify a timer name, ID or magic number to display detailed information on that timer.

"magic" is a unique ID, used by the OS Awareness to identify a specific timer (address of the timer control structure).

The fields "magic" and "routine" are mouse sensitive. Double-clicking on them opens appropriate windows. Right clicking on them will show a local menu.

6	B::TASK.(CLassic.Tlmer					×
m	agic	object id	name	state	routine		
0	0050C10	12010001	TIMR	inactive	00010C8C	_Timer_Routine_adaptor	\$
	¢					>	

TASK.Thread

Display all threads

Format: TASK.Thread [<thread>]

Displays the thread table with all threads of all APIs or detailed information about one specific thread.

Without any arguments, a table with all created threads will be shown.

Specify a thread name, ID or magic number to display detailed information on that thread.

🔒 B::TASK.Thread	
magic object id name state prio lapi 007EC1E8 09010001 IDLE running 255. intern 007EB678 0A010001 UIL_ suspended 1. RTEMS 007EB708 0A010002 ST1 suspended 2. RTEMS 007EB938 0A010003 ST2 suspended 3. RTEMS 007EB938 0A010004 ST3 suspended 4. RTEMS	entry 0000A788 _Thread_Idle_body ^ 000003A0 Init 0000008C semtask1 00000188 semtask2 000001E semtask3
007EBBF8 0A010005 MQT1 suspended 5. RTEMS 007EBD58 0A010006 MQT2 suspended 6. RTEMS	B::TASK.rtems.Thread "MQT2"
<	magic object id name state prio api entry 007EBD58 0A010006 MQT2 suspended 6. RTEMS 00000328 msgqtask2
	object id cond var id node id api 0A010006 6. 1. RTEMS
	prio: current real initial 6. 6. 6.
	Waiting for: none
	resource count ticks executed 0. 2.
	timer: initial start stop delta routine
	stack: address size sp pc
	007DAEC0 00002008 007DCE18 0000A6C8

"magic" is a unique ID, used by the OS Awareness to identify a specific thread (address of the thread control structure).

The fields "magic" and "entry" are mouse sensitive. Double-clicking on them opens appropriate windows. Right clicking on them will show a local menu.

There are special definitions for RTEMS specific PRACTICE functions.

TASK.CONFIG() OS Awareness configuration information

C 1	(ntov)	TASK CONFIG(magia magiaaiza)
- Sy	/ntax:	TASK.CONFIG(magic magicsize)

Parameter and Description:

magic	Parameter Type : String (<i>without</i> quotation marks). Returns the magic address, which is the location that contains the currently running task (i.e. its task magic number).
magicsize	Parameter Type : String (<i>without</i> quotation marks). Returns the size of the task magic number (1, 2 or 4).

Return Value Type: Hex value.

TASK.CLassic.TASKMAX()

Max. number of tasks

Syntax: TASK.CLassic.TASKMAX()

Returns the maximum number of RTEMS tasks.

Return Value Type: Hex value.

TASK.CLassic.TASKLIST()

RTEMS task list

Syntax: TASK.CLassic.TASKLIST(<task_magic>)

Returns the task magic number of the first task if *<task_magic>* is zero. Returns the next task magic number in the RTEMS task list. Returns zero if there are no more tasks in the list.

Parameter Type: Decimal or hex or binary value.

Return Value Type: Hex value.

Syntax: TASK.CLassic.TASKNAME(<task_magic>)

Returns the task name for the specified RTEMS task magic number.

Parameter Type: Decimal or hex or binary value.

Return Value Type: String.