LAUTERBACH
DEVELOPMENT TOOLS

OS Awareness Manual RTEMS

Release 02.2025

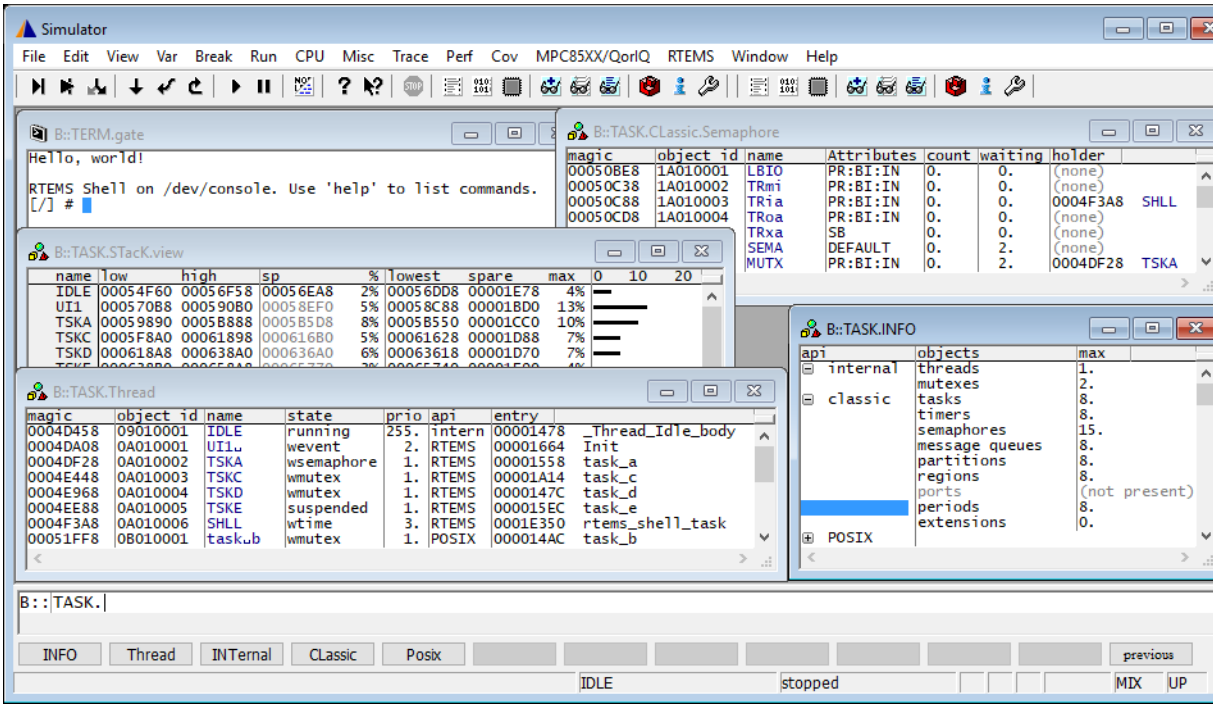
MANUAL

OS Awareness Manual RTEMS

OS Awareness Manual RTEMS

Version 13-Feb-2025

Overview



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.

Brief Overview of Documents for New Users

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

Configuration

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.

Features

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.TImer	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.

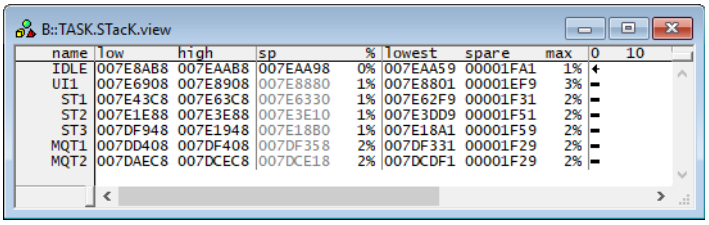
Task Stack Coverage

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.



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.

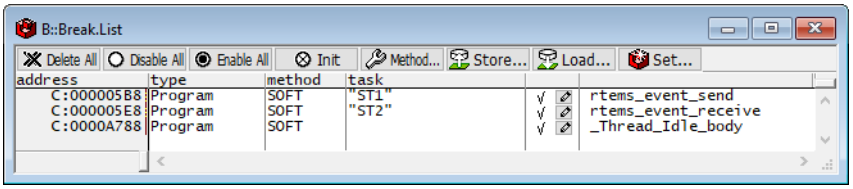
On some architectures, however, it is possible to set a task-related breakpoint with *on-chip* debug logic that is less intrusive. To do this, include the option **/Onchip** in the **Break.Set** command. The debugger then uses the on-chip resources to reduce the number of breaks to the minimum by pre-filtering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

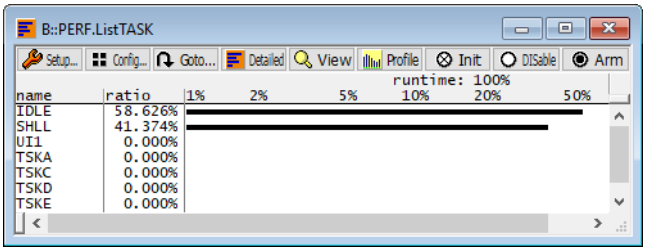


Dynamic Task Performance Measurement

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).



Task Runtime Statistics

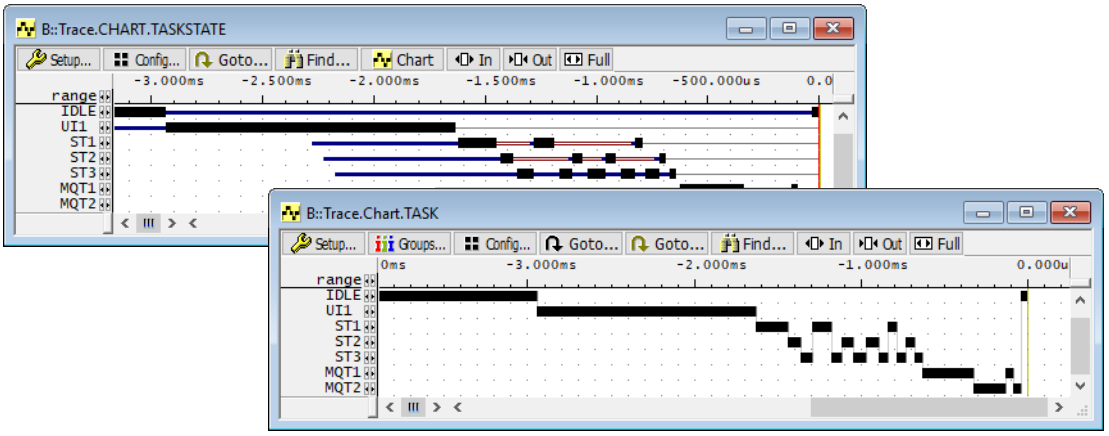
NOTE:	This feature is *only* available, if your debug environment is able to trace task switches (program flow trace is not sufficient). It requires either an on-chip trace logic that is able to generate task information (eg. data trace), or a software instrumentation feeding one of TRACE32 software 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 *only* 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).

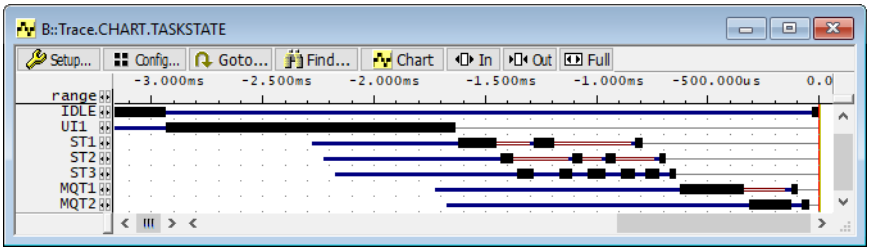
Example: This script assumes that the TCBs are located in an array named TCB_array and consequently limits the tracing to data write accesses on the TCBs and the task switch.

```
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)".



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:

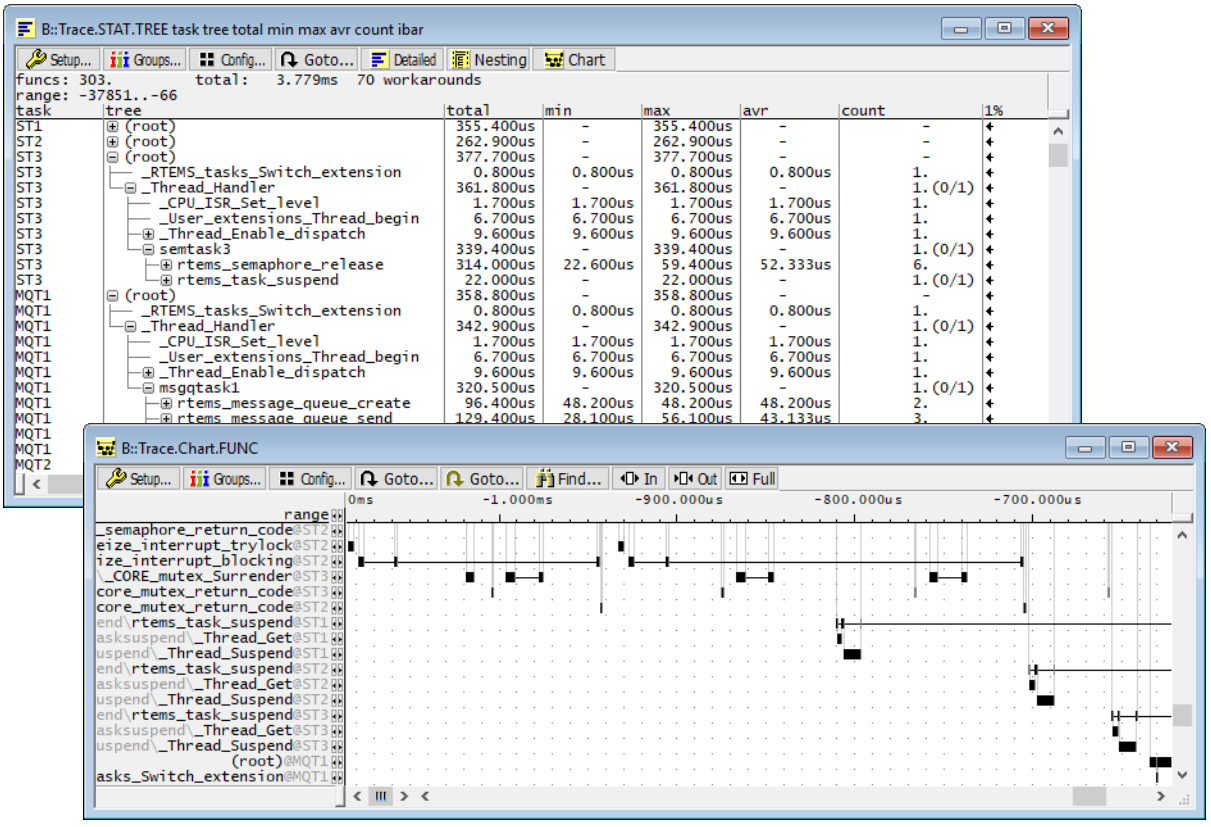
```
; Enable flow trace with Arm Context ID (e.g. 32bit)
ETM.ContextID 32
```

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)".



RTEMS specific Menu

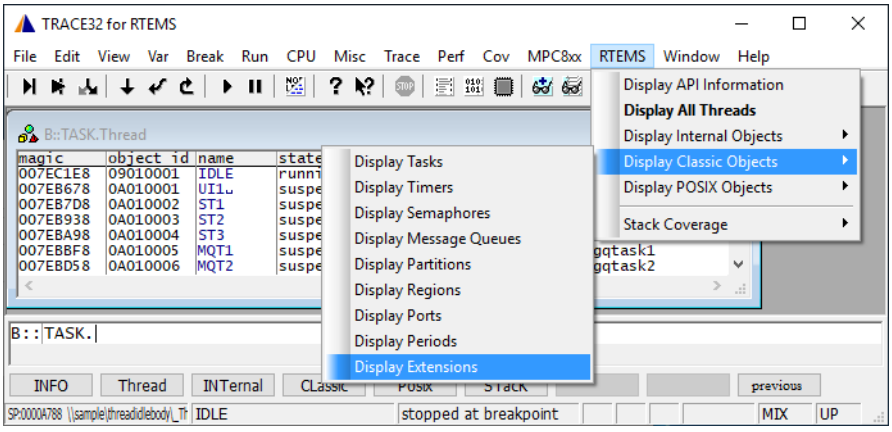
The menu file “rtems.men” contains a menu with RTEMS specific menu items. Load this menu with the **MENU.ReProgram** command.

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.



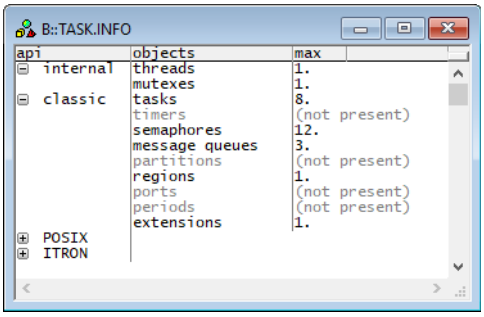
TASK.INFO — Display API information

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.



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

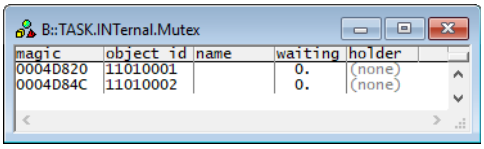
Format: **TASK.INTernal.Mutex** [<*mutex*>]

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.

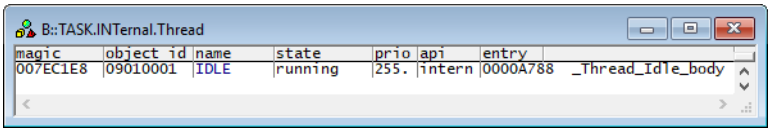


TASK.INTernal.Thread Display internal threads

Format: **TASK.INTernal.Thread** [<*thread*>]

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.



TASK.Posix.CondVar Display POSIX condition variables

Format: **TASK.Posix.CondVar** [<*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.

TASK.Posix.Mutex — Display POSIX mutexes

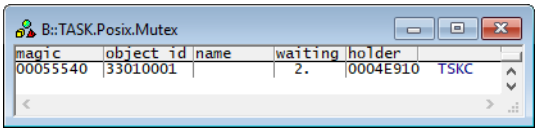
Format: **TASK.Posix.Mutex** [<*mutex*>]

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.

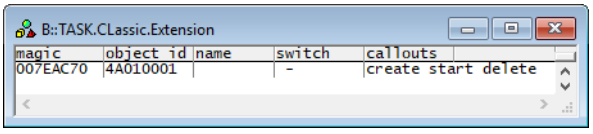


TASK.CLassic.Extension — Display RTEMS extensions

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.



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

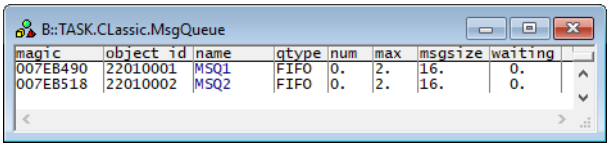
The fields “magic” and “callouts” are mouse sensitive. Double-clicking on them opens appropriate windows. Right clicking on them will show a local menu.

TASK.CLassic.MsgQueue Display RTEMS message queues

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.



“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

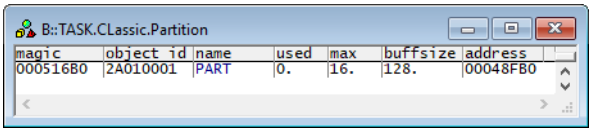
Format: **TASK.CLassic.Partition** [<*partition*>]

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.



TASK.CLassic.PEriod Display RTEMS periods

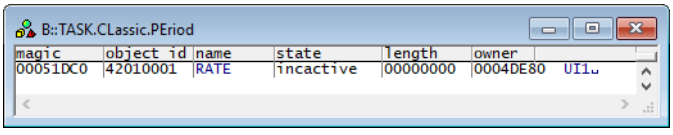
Format: **TASK.CLassic.PEriod** [<*period*>]

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.



TASK.CLassic.POrt Display RTEMS ports

Format: **TASK.CLassic.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).

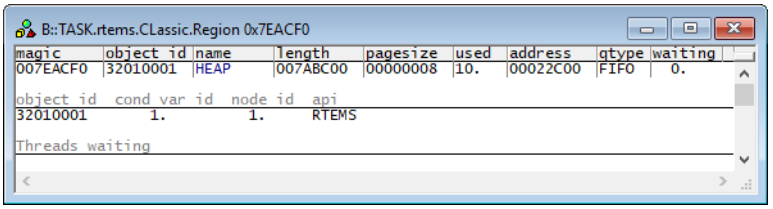
The fields “magic” and “internal” are mouse sensitive. Double-clicking on them opens appropriate windows. Right clicking on them will show a local menu.

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.



“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.

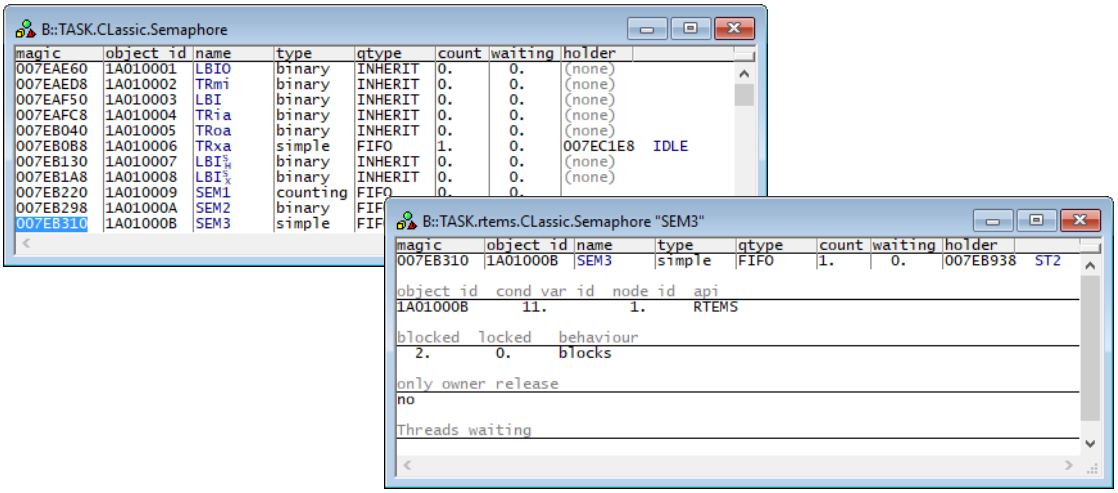
TASK.CLassic.Semaphore Display RTEMS semaphores

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.



“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.

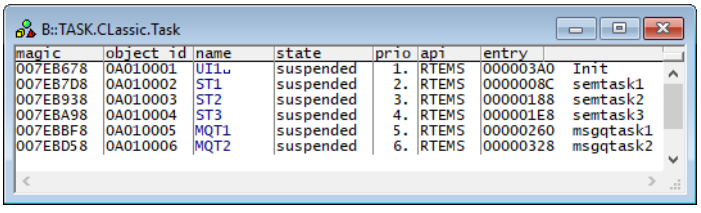
TASK.CLassic.Task — Display RTEMS tasks

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.



“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.

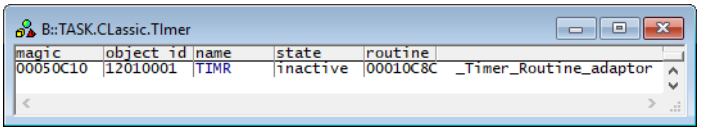
Format: **TASK.CLassic.TImer** [<*timer*>]

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.



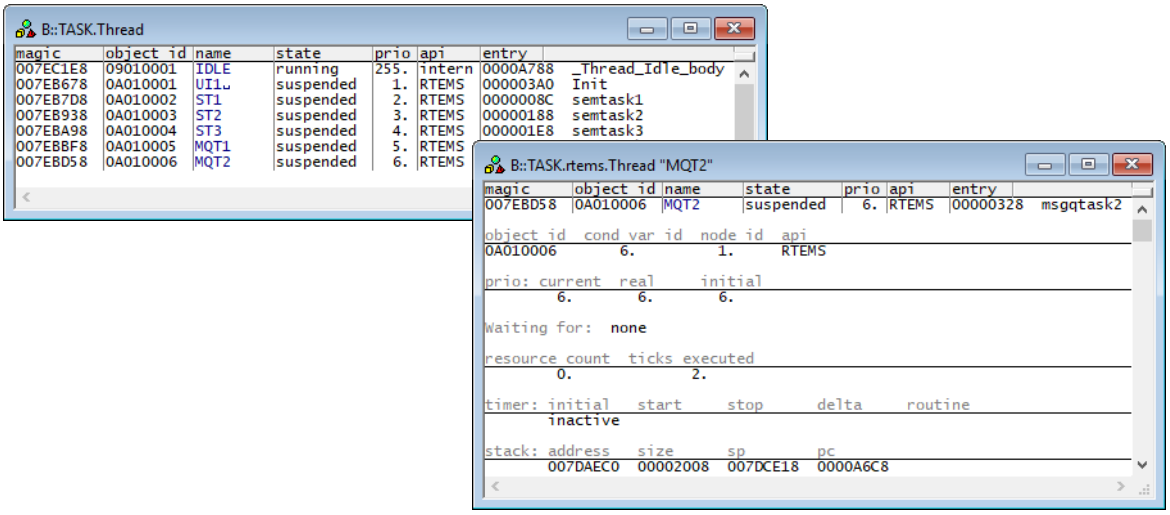
TASK.Thread Display all threads

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.



“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.

RTEMS PRACTICE Functions

There are special definitions for RTEMS specific PRACTICE functions.

TASK.CONFIG() OS Awareness configuration information

Syntax: **TASK.CONFIG(magic | magicsize)**

Parameter and Description:

magic	**Parameter Type**: String (***without*** 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 (***without*** 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.