



# OS Awareness Manual Zephyr

# OS Awareness Manual Zephyr

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

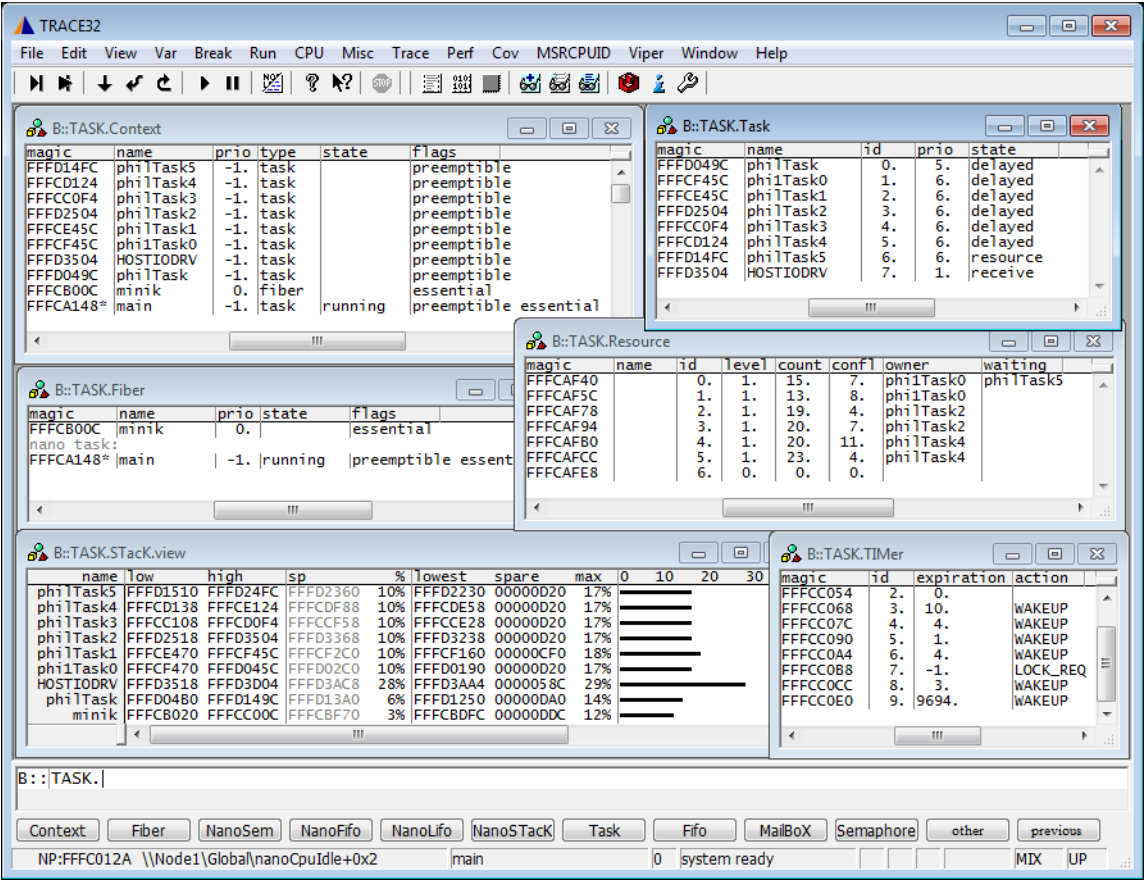
TRACE32 Documents .....	
OS Awareness Manuals .....	
OS Awareness Manual Zephyr .....	1
History .....	4
Overview .....	4
Terminology	5
Brief Overview of Documents for New Users	5
Supported Versions	6
Configuration .....	6
Quick Configuration Guide	7
Hooks & Internals in Zephyr	7
Features .....	9
Display of Kernel Resources	9
Task Stack Coverage	10
Task-Related Breakpoints	11
Dynamic Task Performance Measurement	12
Task Runtime Statistics	12
Function Runtime Statistics	13
Zephyr specific Menu	14
Zephyr Commands for v1.0 .....	15
TASK.Context	Display contexts 15
TASK.Event	Display microkernel events 15
TASK.Fiber	Display fibers 16
TASK.FIFO	Display microkernel FIFOs 16
TASK.MailBoX	Display microkernel mailboxes 17
TASK.Map	Display microkernel maps 17
TASK.MuTeX	Display microkernel mutexes 18
TASK.NanoFifo	Display nanokernel FIFOs 18
TASK.NanoLifo	Display nanokernel LIFOs 19
TASK.NanoSem	Display nanokernel semaphores 19
TASK.NanoSTack	Display nanokernel stacks 20
TASK.PIPE	Display microkernel pipes 21
TASK.Pool	Display microkernel pools 21

TASK.Semaphore	Display microkernel semaphores	22
TASK.Task	Display tasks	22
TASK.TIMER	Display microkernel timers	23
<b>Zephyr Commands for v1.7 .....</b>		<b>24</b>
TASK.ALERT	Display alerts	24
TASK.MailBOX	Display mailboxes	24
TASK.MEMSLAB	Display memslabs	25
TASK.MSGQ	Display msgqs	25
TASK.MUTEX	Display mutexes	25
TASK.SEMaphore	Display semaphores	26
TASK.THREAD	Display threads	26
TASK.TIMER	Display timers	27
TASK.PIPE	Display pipes	27
TASK.QUEUE	Display queues	27
TASK.ZSTACK	Display zstacks	28
<b>Zephyr PRACTICE Functions .....</b>		<b>29</b>
TASK.CONFIG()	OS Awareness configuration information	29

History

28-Aug-18      The title of the manual was changed from “RTOS Debugger for <x>” to “OS Awareness Manual <x>”.

Overview



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

# Terminology

---

Zephyr v1.0 uses the terms “fibers” and “tasks”. If not otherwise specified, the TRACE32 term “task” corresponds to Zephyr fibers *and* tasks.

Zephyr v1.7 onwards uses the term “threads”. If not otherwise specified, the TRACE32 term “task” corresponds to Zephyr threads.

## Brief Overview of Documents for New Users

---

### Architecture-independent information:

- **“Training Basic Debugging”** (training\_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **“T32Start”** (app\_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **“General Commands”** (general\_ref\_<x>.pdf): Alphabetic list of debug commands.

### 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**.
- **“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.
- **Linux Debugging Reference Card** (support.lauterbach.com/downloads/reference-cards)

# Supported Versions

---

Currently Zephyr is supported for the following versions:

- Zephyr 1.0 on ARM and x86.
- Zephyr 1.7 on ARM and x86.  
Special configuration options must be set to the kernel - see [Hooks & Internals](#).
- Zephyr 1.14 on ARM and x86
- Zephyr 2.1 to 2.5 on ARM, RISC-V and x86
- Zephyr 2.6 to 2.7 on ARM, RISC-V and x86.  
Object tables other than threads are **not** supported, because the OS lacks the information for this.
- Zephyr 2.8 onwards on ARM, RISC-V and x86.  
Special configuration options must be set to the kernel - see [Hooks & Internals](#).
- Zephyr 3.x on ARC, ARM, RISC-V and x86.  
Special configuration options must be set to the kernel - see [Hooks & Internals](#).

# Configuration

---

The **TASK.CONFIG** command loads an extension definition file called “zephyr.t32” (directory “~/demo/<arch>/kernel/zephyr/<version>”). It contains all necessary extensions.

Automatic configuration tries to locate the Zephyr internals automatically. For this purpose all symbol tables must be loaded and accessible at any time the OS Awareness is used. Some Zephyr versions need special settings to allow automatic detection of object lists. Please see “[Hooks & Internals](#)”.

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. In case of ICD, you have to enable **SYStem.MemAccess** or **SYStem.CpuAccess** (CPU dependent).

For system resource display, you can do an automatic configuration of the OS Awareness. For this purpose it is necessary that all system internal symbols are loaded and accessible at any time, the OS Awareness is used. Each of the **TASK.CONFIG** arguments can be substituted by '0', which means that this argument will be searched and configured automatically. For a fully automatic configuration omit all arguments:

Format:	<b>TASK.CONFIG zephyr</b>
---------	---------------------------

See also “[Hooks & Internals](#)” for details on the used symbols and how to load object names.

# Quick Configuration Guide

---

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

1. Copy the files `zephyr.t32` and `zephyr.men` to your project directory (from TRACE32 directory “`~/demo/<arch>/kernel/zephyr/<version>`”)
2. Start the TRACE32 Debugger.
3. Load your application as normal.
4. Execute the command `TASK.CONFIG zephyr.t32` (See “[Configuration](#)”).
5. Execute the command `MENU.ReProgram zephyr.men` (See “[Zephyr Specific Menu](#)”).
6. Start your application.

Now you can access the Zephyr extensions through the menu.

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

## Hooks & Internals in Zephyr

---

No hooks are used in the kernel.

For retrieving the kernel data 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 Zephyr kernel must be compiled with debug information.

### Zephyr v1.7:

To be able to support all features mentioned herein, the kernel must be configured with the following options:

```
CONFIG_THREAD_MONITOR=y
CONFIG_THREAD_STACK_INFO=y
CONFIG_OBJECT_TRACING=y
```

### Zephyr v2.8:

To be able to support all features mentioned herein, the kernel must be configured with the following options:

```
CONFIG_THREAD_MONITOR=y
CONFIG_THREAD_STACK_INFO=y
CONFIG_INIT_STACKS=y
CONFIG_OBJECT_TRACING=y
CONFIG_TRACING_OBJECT_TRACKING=y
```

## Zephyr v3.x:

To be able to support all features mentioned herein, the kernel must be configured with the following options:

```
CONFIG_THREAD_MONITOR=y
CONFIG_THREAD_STACK_INFO=y
CONFIG_INIT_STACKS=y
CONFIG_TRACING=y
CONFIG_TRACING_OBJECT_TRACKING=y
CONFIG_TRACING_SYSCALL=y
CONFIG_TRACING_THREAD=y
CONFIG_TRACING_WORK=y
CONFIG_TRACING_ISR=y
CONFIG_TRACING_SEMAPHORE=y
CONFIG_TRACING_MUTEX=y
CONFIG_TRACING_CONDVAR=y
CONFIG_TRACING_QUEUE=y
CONFIG_TRACING_FIFO=y
CONFIG_TRACING_LIFO=y
CONFIG_TRACING_STACK=y
CONFIG_TRACING_MESSAGE_QUEUE=y
CONFIG_TRACING_MAILBOX=y
CONFIG_TRACING_PIPE=y
CONFIG_TRACING_HEAP=y
CONFIG_TRACING_MEMORY_SLAB=y
CONFIG_TRACING_TIMER=y
CONFIG_TRACING_EVENT=y
```

# Features

The OS Awareness for Zephyr supports the following features.

## Display of Kernel Resources

The extension defines new commands to display various kernel resources.

**NOTE:**

The display command listed here apply only for Zephyr 1.0.  
Zephyr 1.7 uses different objects and commands which are not yet documented.

In **Zephyr 1.0**, information on the following components can be displayed:

<b>TASK.Context</b>	Contexts
<b>TASK.Event</b>	Microkernel Events
<b>TASK.Fiber</b>	Nanokernel Fibers
<b>TASK.FIFO</b>	Microkernel Fifos
<b>TASK.MailBoX</b>	Microkernel Mailboxes
<b>TASK.Map</b>	Microkernel Memory Maps
<b>TASK.NanoFifo</b>	Nanokernel Fifos
<b>TASK.NanoLifo</b>	Nanokernel Lifos
<b>TASK.NanoSem</b>	Nanokernel Semaphores
<b>TASK.NanoSTack</b>	Nanokernel Stacks
<b>TASK.PIPE</b>	Microkernel Pipes
<b>TASK.Pool</b>	Microkernel Memory Pools
<b>TASK.Semaphore</b>	Microkernel Semaphores
<b>TASK.Task</b>	Microkernel Tasks
<b>TASK.TIMER</b>	Microkernel Timers

For a description of the commands, refer to chapter “**Zephyr Commands v1.0**”.

In **Zephyr 1.7 onwards**, information on the following components can be displayed:

<b>TASK.ALERT</b>	Alerts
<b>TASK.MailBOX</b>	Mailboxes
<b>TASK.MEMSLAB</b>	Memory Slabs
<b>TASK.MSGQ</b>	Message Queues
<b>TASK.MUTEX</b>	Mutexes

<b>TASK.SEMaphore</b>	Semaphores
<b>TASK.THREAD</b>	Threads
<b>TASK.TIMER</b>	Timers
<b>TASK.PIPE</b>	Pipes
<b>TASK.QUEUE</b>	Queues
<b>TASK.ZSTACK</b>	Zephyr Stacks

For a description of the commands, refer to chapter “[Zephyr Commands v1.7](#)”.

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:

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

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

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](#)” (glossary.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:

<a href="#">Trace.ListNesting</a>	Display function nesting
<a href="#">Trace.STATistic.Func</a>	Display function runtime statistic
<a href="#">Trace.STATistic.TREE</a>	Display functions as call tree
<a href="#">Trace.STATistic.sYmbol /SplitTASK</a>	Display flat runtime analysis
<a href="#">Trace.Chart.Func</a>	Display function timechart
<a href="#">Trace.Chart.sYmbol /SplitTASK</a>	Display flat runtime timechart

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

## Zephyr specific Menu

---

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

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

- The **Display** menu items launch the kernel resource display windows.
- The **Stack Coverage** submenu starts and resets the Zephyr 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 the default display.
- The **Perf** menu contains additional submenus for task runtime statistics.

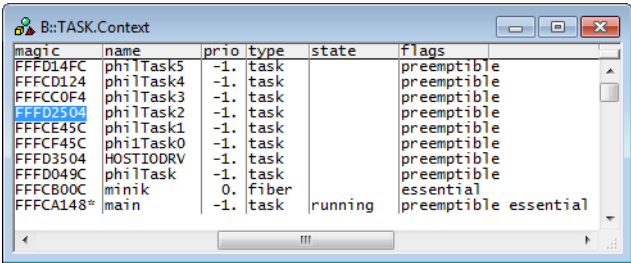
## TASK.Context

Display contexts

Format:

TASK.Context

Displays the context table of Zephyr.



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

The field “magic” is mouse sensitive, double clicking on it opens an appropriate window. Right clicking on it will show a local menu.

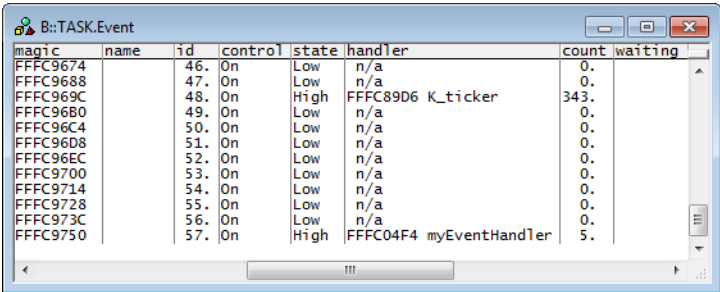
## TASK.Event

Display microkernel events

Format:

TASK.Event

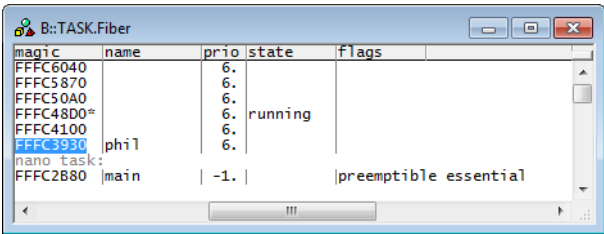
Displays the event table of Zephyr Microkernel.



“magic” is a unique ID, used by the OS Awareness to identify a specific event (address of the event control structure). The field “magic” is mouse sensitive, double clicking on it opens an appropriate window. Right clicking on it will show a local menu.

Format: TASK.Fiber

Displays the fiber table of Zephyr.

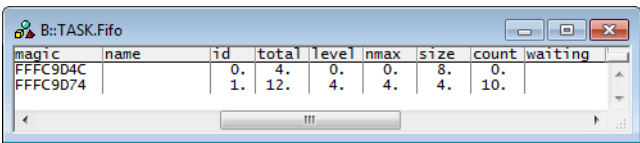


“magic” is a unique ID, used by the OS Awareness to identify a specific fiber (address of the fiber control structure).

The field “magic” is mouse sensitive, double clicking on it opens an appropriate window. Right clicking on it will show a local menu.

Format: TASK.FIFO

Displays the FIFO table of Zephyr Microkernel.

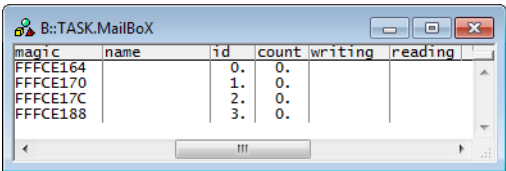


“magic” is a unique ID, used by the OS Awareness to identify a specific FIFO (address of the FIFO control structure).

The field “magic” is mouse sensitive, double clicking on it opens an appropriate window. Right clicking on it will show a local menu.

Format:           **TASK.MailBoX**

Displays the mailbox table of Zephyr Microkernel.

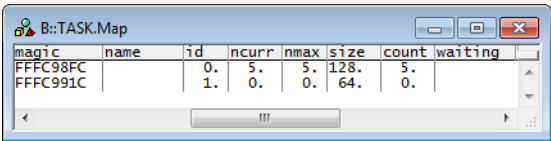


“magic” is a unique ID, used by the OS Awareness to identify a specific mailbox (address of the mailbox control structure).

The field “magic” is mouse sensitive, double clicking on it opens an appropriate window. Right clicking on it will show a local menu.

Format:           **TASK.Map**

Displays the memory map table of Zephyr Microkernel.



“magic” is a unique ID, used by the OS Awareness to identify a specific memory map (address of the map control structure).

The field “magic” is mouse sensitive, double clicking on it opens an appropriate window. Right clicking on it will show a local menu.

Format:

TASK.MuTeX

Displays the mutex table of Zephyr Microkernel.

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

The field “magic” is mouse sensitive, double clicking on it opens an appropriate window. Right clicking on it will show a local menu.

TASK.NanoFifo

Display nanokernel FIFOs

Format:

TASK.NanoFifo <symbol> [/Struct | /Ptr | /Array <size>]

Displays FIFOs of Zephyr Nanokernel. Specify the symbol name of a FIFO to display its contents.

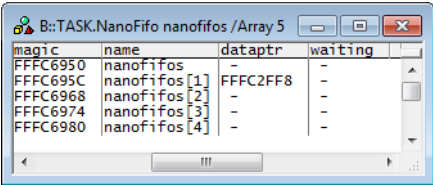
Optional Parameters:

- Struct

<symbol> refers to a variable holding a NANO\_FIFO structure (default)
- Ptr

<symbol> is a pointer to a NANO\_FIFO structure
- Array <size>

<symbol> is an array with <size> NANO\_FIFO entries



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

The field “magic” is mouse sensitive, double clicking on it opens an appropriate window. Right clicking on it will show a local menu.

Format:

TASK.NanoLifo <symbol> [/Struct | /Ptr | /Array <size>]

Displays LIFOs of Zephyr Nanokernel. Specify the symbol name of a LIFO to display its contents.

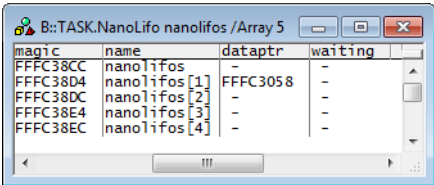
Optional Parameters:

- Struct

<symbol> refers to a variable holding a NANO\_LIFO structure (default)
- Ptr

<symbol> is a pointer to a NANO\_LIFO structure
- Array <size>

<symbol> is an array with <size> NANO\_LIFO entries



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

The field “magic” is mouse sensitive, double clicking on it opens an appropriate window. Right clicking on it will show a local menu.

TASK.NanoSem

Display nanokernel semaphores

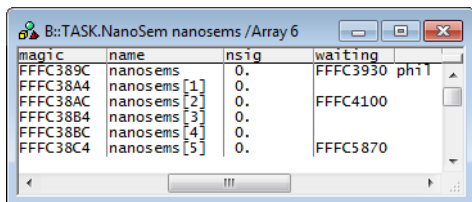
Format:

TASK.NanoSem <symbol> [/Struct | /Ptr | /Array <size>]

Displays semaphores of Zephyr Nanokernel. Specify the symbol name of a semaphore to display its contents.

Optional Parameters:

<b>Struct</b>	<code>&lt;symbol&gt;</code> refers to a variable holding a NANO_SEM structure (default)
<b>Ptr</b>	<code>&lt;symbol&gt;</code> is a pointer to a NANO_SEM structure
<b>Array &lt;size&gt;</b>	<code>&lt;symbol&gt;</code> is an array with <code>&lt;size&gt;</code> NANO_SEM entries



magic	name	nsig	waiting
FFFC389C	nanosems	0.	FFFC3930 ph1
FFFC38A4	nanosems [1]	0.	
FFFC38AC	nanosems [2]	0.	FFFC4100
FFFC38B4	nanosems [3]	0.	
FFFC38BC	nanosems [4]	0.	
FFFC38C4	nanosems [5]	0.	FFFC5870

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

The field “magic” is mouse sensitive, double clicking on it opens an appropriate window. Right clicking on it will show a local menu.

## TASK.NanoSTack

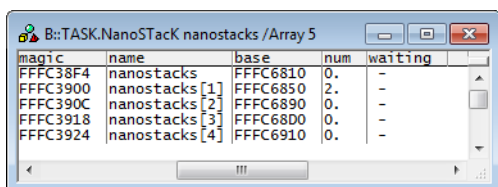
## Display nanokernel stacks

Format: **TASK.NanoSTack** `<symbol>` [/Struct | /Ptr | /Array `<size>`]

Displays stacks of Zephyr Nanokernel. Specify the symbol name of a stack to display its contents.

Optional Parameters:

<b>Struct</b>	<code>&lt;symbol&gt;</code> refers to a variable holding a NANO_STACK structure (default)
<b>Ptr</b>	<code>&lt;symbol&gt;</code> is a pointer to a NANO_STACK structure
<b>Array &lt;size&gt;</b>	<code>&lt;symbol&gt;</code> is an array with <code>&lt;size&gt;</code> NANO_STACK entries



magic	name	base	num	waiting
FFFC38F4	nanostacks	FFFC6810	0.	-
FFFC3900	nanostacks [1]	FFFC6850	2.	-
FFFC390C	nanostacks [2]	FFFC6890	0.	-
FFFC3918	nanostacks [3]	FFFC68D0	0.	-
FFFC3924	nanostacks [4]	FFFC6910	0.	-

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

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

TASK.PIPE

Display microkernel pipes

Format: TASK.PIPE

Displays the pipe table of Zephyr Microkernel.

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

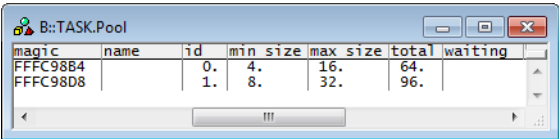
The field “magic” is mouse sensitive, double clicking on it opens an appropriate window. Right clicking on it will show a local menu.

TASK.Pool

Display microkernel pools

Format: TASK.Pool

Displays the memory pool table of Zephyr Microkernel.



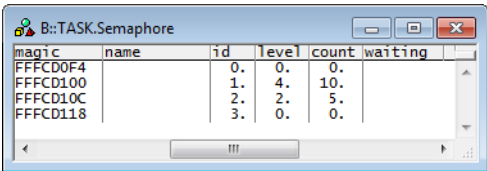
magic	name	id	min size	max size	total	waiting
FFFC98B4		0.	4.	16.	64.	
FFFC98D8		1.	8.	32.	96.	

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

The field “magic” is mouse sensitive, double clicking on it opens an appropriate window. Right clicking on it will show a local menu.

Format: TASK.Semaphore

Displays the semaphore table of Zephyr Microkernel.



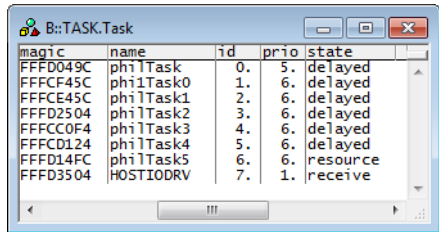
magic	name	id	level	count	waiting
FFFC00F4		0.	0.	0.	
FFFC0100		1.	4.	10.	
FFFC010C		2.	2.	5.	
FFFC0118		3.	0.	0.	

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

The field “magic” is mouse sensitive, double clicking on it opens an appropriate window. Right clicking on it will show a local menu.

Format: TASK.Task

Displays the task table of Zephyr.



magic	name	id	prio	state
FFFD049C	philTask	0.	5.	delayed
FFFCF45C	philTask0	1.	6.	delayed
FFFCF45C	philTask1	2.	6.	delayed
FFFD2504	philTask2	3.	6.	delayed
FFFC00F4	philTask3	4.	6.	delayed
FFFD124	philTask4	5.	6.	delayed
FFFD14FC	philTask5	6.	6.	resource
FFFD3504	HOSTIODRV	7.	1.	receive

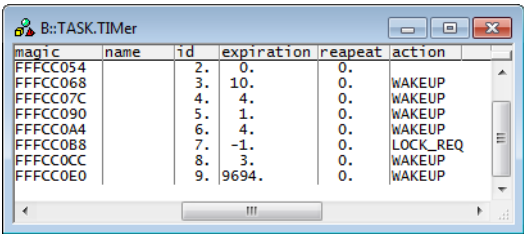
“magic” is a unique ID, used by the OS Awareness to identify a specific task (address of the task control structure).

The field “magic” is mouse sensitive, double clicking on it opens an appropriate window. Right clicking on it will show a local menu.

Format:

TASK.TIMER

Displays the timer table of Zephyr Microkernel.



The screenshot shows a window titled "B::TASK.TIMER" containing a table of timer data. The table has columns for magic, name, id, expiration, repeat, and action. The data rows are as follows:

magic	name	id	expiration	repeat	action
FFFFC054		2.	0.	0.	
FFFFC068		3.	10.	0.	WAKEUP
FFFFC07C		4.	4.	0.	WAKEUP
FFFFC090		5.	1.	0.	WAKEUP
FFFFC0A4		6.	4.	0.	WAKEUP
FFFFC0B8		7.	-1.	0.	LOCK_REQ
FFFFC0CC		8.	3.	0.	WAKEUP
FFFFC0E0		9.	9694.	0.	WAKEUP

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

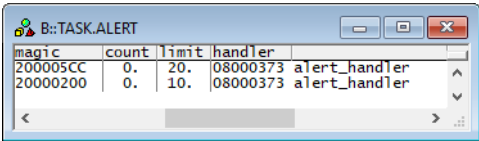
The field “magic” is mouse sensitive, double clicking on it opens an appropriate window. Right clicking on it will show a local menu.

TASK.ALERT

Display alerts

Format: TASK.ALERT

Displays the alert table of Zephyr.

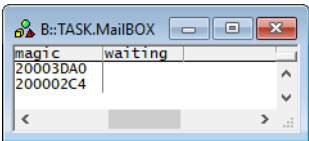


TASK.MailBOX

Display mailboxes

Format: TASK.MailBOX

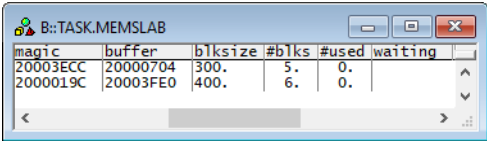
Displays the mailbox table of Zephyr.



Format:

TASK.MEMSLAB

Displays the memory slab table of Zephyr.



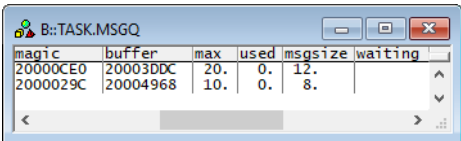
The screenshot shows a terminal window titled "B::TASK.MEMSLAB" displaying a table of memory slab information. The table has columns: magic, buffer, blksize, #blks, #used, and waiting. There are three rows of data.

magic	buffer	blksize	#blks	#used	waiting
20003ECC	20000704	300.	5.	0.	
2000019C	20003FE0	400.	6.	0.	

Format:

TASK.MSGQ

Displays the message queue table of Zephyr.



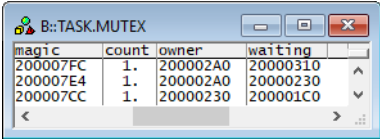
The screenshot shows a terminal window titled "B::TASK.MSGQ" displaying a table of message queue information. The table has columns: magic, buffer, max, used, msgsize, and waiting. There are three rows of data.

magic	buffer	max	used	msgsize	waiting
20000CE0	20003DDC	20.	0.	12.	
2000029C	20004968	10.	0.	8.	

Format:

TASK.MUTEX

Displays the mutex table of Zephyr.

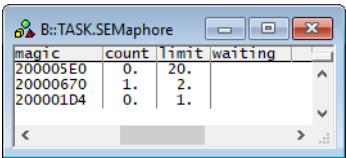


The screenshot shows a terminal window titled "B::TASK.MUTEX" displaying a table of mutex information. The table has columns: magic, count, owner, and waiting. There are three rows of data.

magic	count	owner	waiting
200007FC	1.	200002A0	20000310
200007E4	1.	200002A0	20000230
200007CC	1.	20000230	200001C0

Format: TASK.SEMaphore

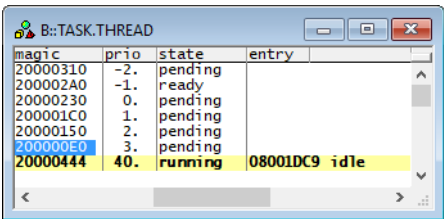
Displays the semaphore table of Zephyr.



magic	count	limit	waiting
200005E0	0.	20.	
20000670	1.	2.	
200001D4	0.	1.	

Format: TASK.THREAD

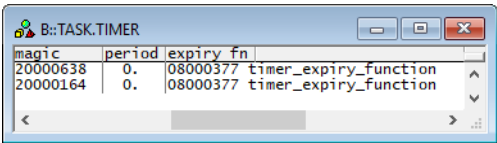
Displays the thread table of Zephyr.



magic	prio	state	entry
20000310	-2.	pending	
200002A0	-1.	ready	
20000230	0.	pending	
200001C0	1.	pending	
20000150	2.	pending	
200000E0	3.	pending	
20000444	40.	running	08001DC9 idle

Format: TASK.TIMER

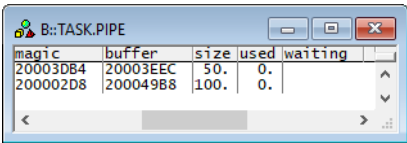
Displays the timer table of Zephyr.



magic	period	expiry fn
20000638	0.	08000377 timer_expiry_function
20000164	0.	08000377 timer_expiry_function

Format: TASK.PIPE

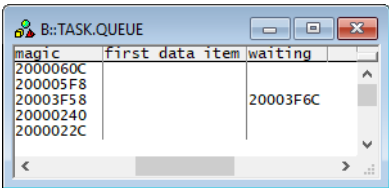
Displays the pipe table of Zephyr.



magic	buffer	size	used	waiting
20003DB4	20003EEC	50.	0.	
200002D8	200049B8	100.	0.	

Format: TASK.QUEUE

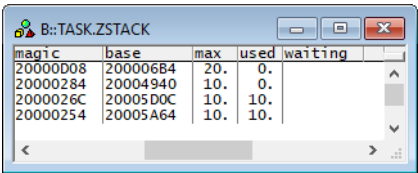
Displays the queue table of Zephyr.



magic	first data item	waiting
2000060C		
200005F8		
20003F58		20003F6C
20000240		
2000022C		

Format: TASK.ZSTACK

Displays the Zephyr stack table.



The screenshot shows a terminal window with the title 'B::TASK.ZSTACK'. It displays a table with five columns: 'magic', 'base', 'max', 'used', and 'waiting'. The table contains four rows of data, each representing a different stack entry. The 'magic' column contains hexadecimal values, 'base' contains memory addresses, 'max' contains decimal values, 'used' contains decimal values, and 'waiting' contains decimal values.

magic	base	max	used	waiting
20000D08	200006B4	20.	0.	
20000284	20004940	10.	0.	
2000026C	20005D0C	10.	10.	
20000254	20005A64	10.	10.	

There are special definitions for Zephyr specific PRACTICE functions.

TASK.CONFIG()

OS Awareness configuration information

---

Syntax:

TASK.CONFIG(magic | magicsize)

Parameter and Description:

magic	<b>Parameter Type:</b> 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	<b>Parameter Type:</b> String ( <i>without</i> quotation marks). Returns the size of the task magic number (1, 2 or 4).

Return Value Type: Hex value.