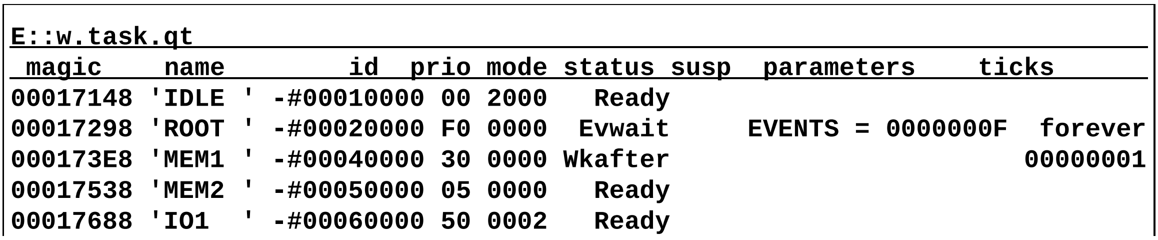
RTOS Debugger for pSOS+

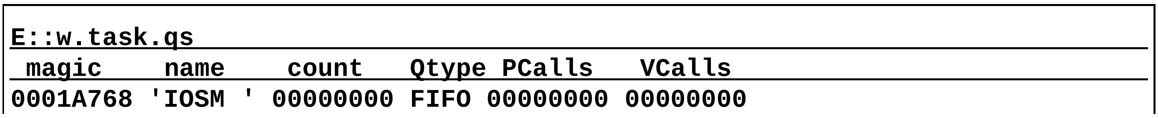
RTOS Debugger for pSOS+

Version December, 01 2009

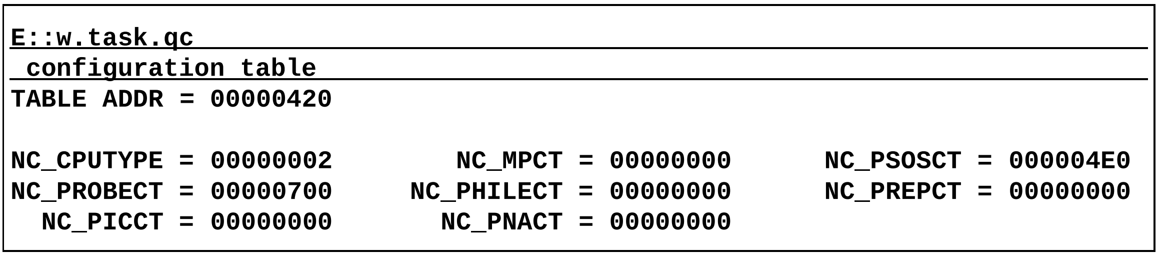
Overview



```
E::w.task.qt
 magic    name        id  prio mode status susp  parameters    ticks
00017148 'IDLE  ' -#00010000 00 2000   Ready
00017298 'ROOT  ' -#00020000 F0 0000  Evwait     EVENTS = 0000000F  forever
000173E8 'MEM1  ' -#00040000 30 0000 Wkafter                         00000001
00017538 'MEM2  ' -#00050000 05 0000   Ready
00017688 'IO1   ' -#00060000 50 0002   Ready
```



```
E::w.task.qs
 magic    name     count    Qtype PCalls   VCalls
0001A768 'IOSM  ' 00000000 FIFO 00000000 00000000
```



```
E::w.task.qc
 configuration table
TABLE ADDR = 00000420

NC_CPUTYPE = 00000002        NC_MPCT = 00000000      NC_PSOSCT = 000004E0
NC_PROBECT = 00000700     NC_PHILECT = 00000000      NC_PREPCT = 00000000
  NC_PICCT = 00000000       NC_PNACT = 00000000
```

The Multitask Debugger for pSOS contains special extensions to the TRACE32 Debugger. This chapter describes the additional features, such as additional commands and statistic evaluations.

For general informations about the In-Circuit Debugger refer to the **”ICD Debugger User’s Guide”** (debugger_user.pdf), **”ICE User’s Guide”** (ice_user.pdf) or **”ICE User’s Guide”** (ice_user.pdf). All general commands are described in **”IDE Reference Guide”** (ide_ref.pdf) and **”General Commands and Functions”**.

Currently pSOS is supported for the following versions:
pSOS 1.1A with pROBE 1.0A on Intel x86, real mode
pSOS 1.1.x, 1.2.x, 2.0.E, 2.1.x on M68k
pSOS 2.0.7 with pROBE 3.0.8 on PowerPC
pSOS 2.1.x on ARM7
pSOS 2.2.3 on PPC
pSOS 2.2.6 on Intel 386, protected mode.
pSOS 2.3.0 on M68K,
pSOS 2.5.x on PPC

Configuration

The TASK.CONFIG command loads an extension definition file called “psos.t32” (directory “demo/*<processor>*/kernel/psos”). It contains all necessary extensions.
Automatic configuration tries to locate the pSOS internals automatically. For this purpose all symbol tables must be loaded and accessable at any time the multitask debugger is used.
If a system symbol is not available or if another address should be used for a specific system variable then the corresponding argument must be set manually with the appropriate address. This can be done by manual configuration which can require some additional arguments, too.
If you want to have dual port access for the display functions (display ’On The Fly’), you have to map emulation memory to the address space of all used system tables.

Manual Configuration

Manual configuration for pSOS multitask debuggers is only necessary for M68k controllers when using the kernel patch.The practice file 'ppsos.cmm' patches pSOS and pROBE and configures the multitask debugger. The macros defined at the beginning of the file specify the address of pROBE+/68k, the address of the current-tcb pointer, and the vectors which are used to enter the kernel.

Format:	**TASK.CONFIG psos** <*magic*> <*sleep*> <*args*>
<*args*>:	**config** <*system_call_gate*>

<*magic*> specifies a memory location, that contains the current running task.
The argument for <sleep> is currently not used. Specify '0'.
The additional arguments of the **TASK.CONFIG** command must be the address of the system call routine, the node configuration table, the pSOS data structures and a flag indicating the main pSOS version.

If the task selective debugging features are not used, the patching of the kernel is not required. The first two arguments are then not required. The PRACTICE program 'demo/m68k/kernel/psos/ppsos.cmm' can make the required patches to pSOS+ and configures the display commands:

```
do ppsos nopatch                ; configures only display functions
                                ; no patches are made (TASK.OFF)

do ppsos noprobe                ; patches pSOS when pROBE is not loaded
                                ; task selective debugging in on

do ppsos notask noprobe         ; patches pSOS when pROBE is not loaded
                                ; task selective debugging in off

do ppsos                        ; patched pSOS and pROBE for task selective
                                ; debugging
```

The batchfile must be modified, when the pSOS node anchor is not at the default location. When patching is required the patch area in the batchfile must be modified to point to an unused memory area.

The demo application 'psos.cmm' in the './demo/m68k/kernel/psos' directory can be started with the same parameters. The application may require modifying the 'psos.cmm' file to load the currently used version of pSOS+.

Automatic Configuration

For system resource display and analyzer functionality, you can do an automatic configuration of the multitask debugger. For this purpose it is necessary, that all system internal symbols are loaded and accessable. Each of the task.config arguments can be substituted by '0', which means, that this argument will be searched and configured automatically. For a full automatic configuration specify '0' to the magic and sleep arguments and omit all other arguments:

Format: **TASK.CONFIG psos** 0 0

If a system symbol is not available, or if another address should be used for a specific system variable, then the corresponding argument must be set manually with the appropriate address.

Quick Configuration Guide

To access all features of the multitask debugger you should follow the following roadmap:

1. Run the demo program (./demo/m68k/kernel/psos/psos.cmm) with your kernel without any patching. This requires that you copy the files from this demo together with your version of the pSOS+ kernel in your directory. Start the demo with 'do psos nopatch'. Add the argument 'noprobe' if you don't want to load pROBE+. The result should be a list of tasks, which change continuously their state.

2. Run the demo program with patching. For CPU32(+) devices you need the argument 'notask', as task selective debugging is not supported on these emulators. (skip for x86 and PPC)

3. Try single stepping, starting and stopping a task. Display kernel resources. On CPU32(+) devices the interrupts during single stepping can be suppressed by **SETUP.IMASKASM**, by a monitor extension (**SYStem.MonFile**) which stops the interrupt source in the chip or by freezing the timer.

4. Try the analyzer demo programs (tasksc, taskstat and taskfunc).

5. Make a copy of the 'ppsos.cmm' batchfile. Modify the file according to your application. This can be changing the pSOS+ node anchor or choosing a different memory area for the patches.

6. Run the modified version in your application **without patching** (with 'nopatch' argument). This should allow you to display the kernel resources and use most of the analyzer features (except the system call display).

7. Run your application with patching, but without task selective debugging ('notask' argument). (skip for x86 and PPC)

8. Run your application with task selective debugging, when required (not CPU32(+) devices). (skip for x86 and PPC)

Hooks & Internals in pSOS

Kernel patching for M68k:
To determine the entry of a task, the patching of pSOS+, and pROBE+ when used, is required. All returns to the task context (usually RTE instructions) are patched to pass control to the multitask monitor. The patch writes the current executing tcb address to the magic-word of the multitask debugger and runs to a breakpoint.The entries to pSOS are patched directly in the vector table. The patches write the value 1 to the magic-word and run to a breakpoint. The 'breakpoint'-trap of pROBE+ should be patched too. This will ease the combination of pROBE+ breakpoints together with the state analyzer.
The task-delete hook of pSOS is used to detect if a task has been deleted. If this hook (KC_DELETECO) is already in use, an additional patch in required.
To stop a task and continue the kernel, the debugger uses the 'manual round robin' feature of PSOS. In this case the debugger will executed the function-call 'tm_wkafter(1)'.

No hooks are used for x86 and PPC.

Features

The multitask debugger for pSOS supports the following features.

Display of Kernel Resources

The extension defines new PRACTICE commands to display various kernel resources. The commands can either give an overview about one resource type or display a single resource in detail. The resource can be defined by it's ID, magic or name. The following information can be displayed:

The following information can be displayed for i386, M68k and PPC:

- configuration	(TASK.QC)	
- objects	(TASK.QO)	
- tasks	(TASK.QT)	
- queues	(TASK.QQ)	
- semaphores	(TASK.QS)	
- regions	(TASK.QR)	
- partitions	(TASK.QP)	
- date and time	(TASK.QD)	
- versions	(TASK.QV)	(only available on PPC)
- system calls	(TASK.SC)	(only available on M68k)

For a detailed description of each command refer to the chapter “pSOS Commands for i386, M68k and PPC”.

The following information can be displayed for x86:

- configuration	(TASK.QC)
- process table	(TASK.QP)
- exchange table	(TASK.QX)
- time	(TASK.QT)
- memory	(TASK.QM)

For a detailed description of each command refer to the chapter “pSOSx86 Commands”.

When working with emulation memory or shadow memory, these resources can be displayed “On The Fly”, i.e. while the target application is running, without any intrusion to the application. If using this dual port memory feature, be sure that emulation memory is mapped to all places, where pSOS holds its tables.

When working only with target memory, the information will only be displayed, if the target application is stopped.

TRACE32 Board Support Package with pROBE+ Terminal Emulation

(only available for PPC) pSOS+ users can call for a TRACE32 board support package. This package is based on the SBC821 BSP. It allows to create applications that run on the in circuit emulator without any target.

The BSP contains a special console driver which connects the pROBE+ with a terminal emulation window of the emulator. The 'break' command pROBE, to stop a running application (default: 'CTRL-C') is changed to 'TAB'. The communication between pROBE+ and the terminal is done via two memory cells, requiring no external interface.

Our demo application was built with this TRACE32 BSP.

Task Runtime Statistics

The time spent in a task can be analyzed by marking the accesses to a word holding the current task descriptor. All kernel activities are added to the calling task. The example program 'taskfunc.cmm' can be used to make the measurement for this analysis.

Analyzer.List List.TASK DEFault	Display trace buffer and task switches
Analyzer.STATistic.TASK	Display task runtime statistic evaluation
Analyzer.Chart.TASK	Display task runtime time chart

Task State Analysis

The time different tasks are is a certain state (running, ready, suspended or waiting) can be displayed as a statistic or in graphical form. This feature is implemented by recording all accesses to the status words of all tasks. Additionally the accesses to the current tcb pointer or the magic word are traced. This is required as the status of a task makes no difference between 'running' and 'ready'. The breakpoints to the tcb status words are set by the **TASK.TASKState** command. The example program 'taskstat.cmm' makes a task state analysis with the demo application.

Analyzer.STATistic.TASKState	Display task state statistic evaluation
Analyzer.Chart.TASKState	Display task state time chart

Function Runtime Statistics

All function related statistic and time chart functions can be used with task specific information. The task switch can be displayed in the analyzer list with the **List.TASK** keyword. The example program 'taskfunc.cmm' makes a taskselective performance analysis for the demo application.

Analyzer.List List.TASK FUNC	Display function nesting
Analyzer.STATistic.TASKFunc	Display function runtime statistic
Analyzer.STATistic.TASKTREE	Display functions as call tree
Analyzer.Chart.TASKFunc	Display function time chart

System Calls

Manually executing system calls requires a small program on the target, which makes the system call and stops execution after the call. Such a program is part of the standard patch procedure (ppsos.cmm). The memory at the system parameter buffer (a part of the patch area) must be mapped internal (only available for M68k).

Task Selective Debugging

Task selective debugging allows to disable or enable the analyzer and the trigger system for specific tasks and to stop one task while others continue to operate. This function has an impact on the response time of the multitask kernel. The feature should not be used when making performance or time measurements or with extremely time critical applications. Task selective debugging not available on x86, PPC, CPU32 and CPU32+ processors.

pSOS specific Menu

The file “psos.men” contains an alternate menu with pSOS specific topics. Load this menu with the **MENU.ReProgram** command.

You will find a new pull down menu called “pSOS+”. The topic 'pROBE Terminal' brings up a terminal emulation window, which communicates with the preconfigured pROBE+ debugger. 'Break to pROBE' performs a special break command inside the terminal emulation to gain control to pROBE (see TRACE32 BSP). The 'Query' topics launch the kernel resource display windows.

The Analyzer->List pull-down menu is changed. You can additionally choose for an analyzer list window showing only task switches (if any) or task switches and defaults.

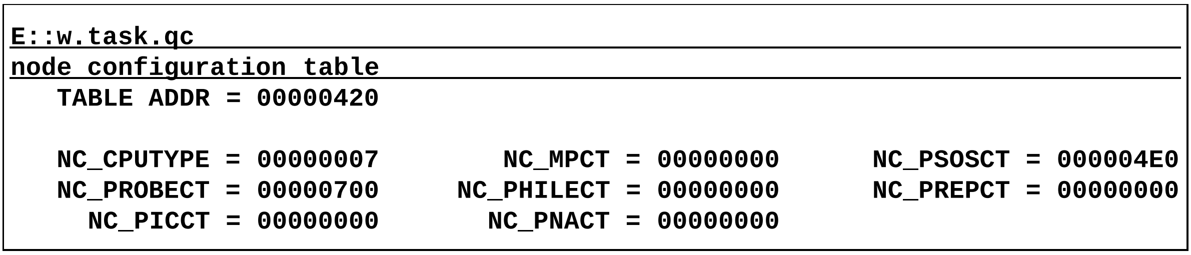
The “Perf” menu contains the additional submenus for task runtime statistics, task related function runtime statistics and statistics on task states. For the function runtime statistics, a prepare command file called “men_ptfp.cmm” is used. This command file must be adapted to your application.

TASK.QC

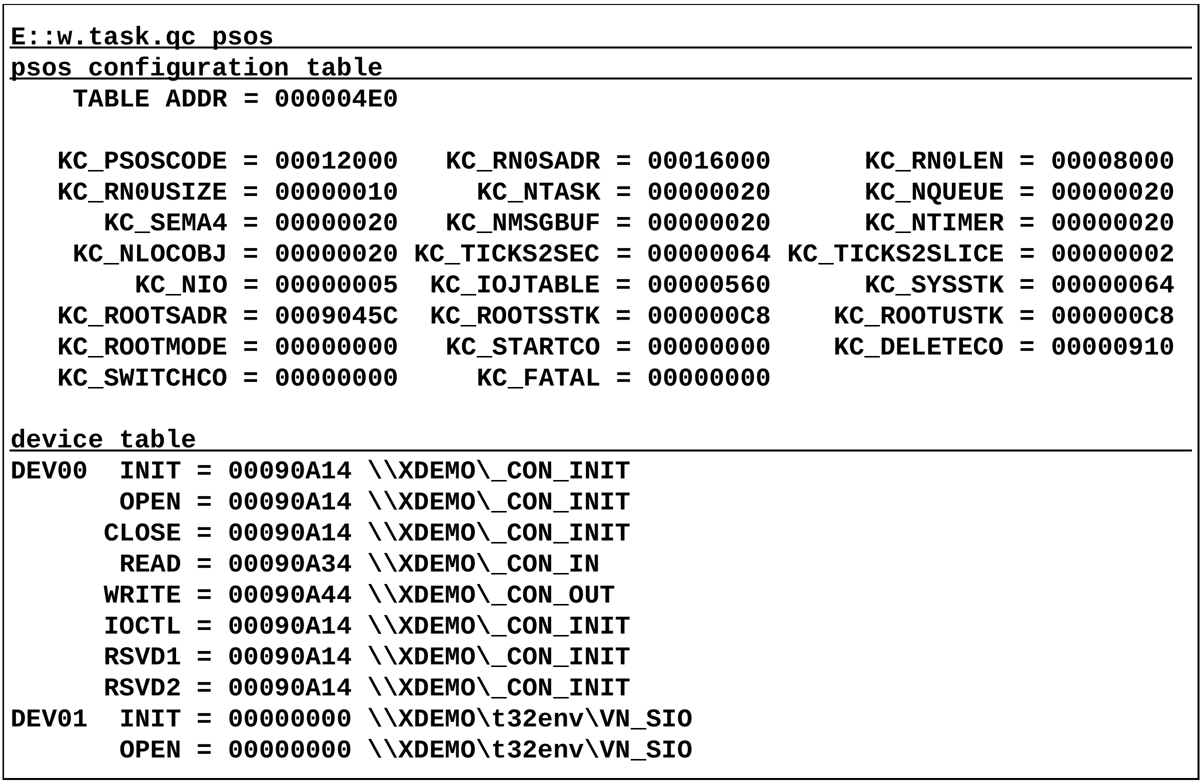
Configuration

Format: **TASK.QC [NODE | PSOS | PROBE]**

Displays the configuration tables of pSOS+. Some of the fields are mouse sensitive. Double clicking on them will show the appropriate information.



```
E::w.task.qc
node configuration table
   TABLE ADDR = 00000420

   NC_CPUTYPE = 00000007        NC_MPCT = 00000000      NC_PSOSCT = 000004E0
   NC_PROBECT = 00000700     NC_PHILECT = 00000000      NC_PREPCT = 00000000
     NC_PICCT = 00000000       NC_PNACT = 00000000
```

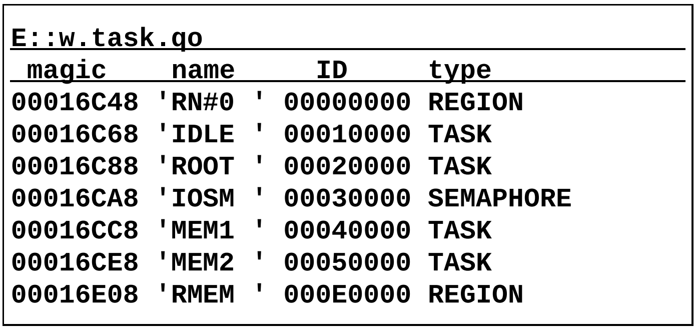


```
E::w.task.qc psos
psos configuration table
    TABLE ADDR = 000004E0

   KC_PSOSCODE = 00012000   KC_RN0SADR = 00016000      KC_RN0LEN = 00008000
   KC_RN0USIZE = 00000010     KC_NTASK = 00000020      KC_NQUEUE = 00000020
      KC_SEMA4 = 00000020   KC_NMSGBUF = 00000020      KC_NTIMER = 00000020
    KC_NLOCOBJ = 00000020 KC_TICKS2SEC = 00000064 KC_TICKS2SLICE = 00000002
        KC_NIO = 00000005  KC_IOJTABLE = 00000560      KC_SYSSTK = 00000064
   KC_ROOTSADR = 0009045C  KC_ROOTSSTK = 000000C8    KC_ROOTUSTK = 000000C8
   KC_ROOTMODE = 00000000   KC_STARTCO = 00000000    KC_DELETECO = 00000910
   KC_SWITCHCO = 00000000     KC_FATAL = 00000000

device table
DEV00  INIT = 00090A14 \\XDEMO\_CON_INIT
       OPEN = 00090A14 \\XDEMO\_CON_INIT
      CLOSE = 00090A14 \\XDEMO\_CON_INIT
       READ = 00090A34 \\XDEMO\_CON_IN
      WRITE = 00090A44 \\XDEMO\_CON_OUT
      IOCTL = 00090A14 \\XDEMO\_CON_INIT
      RSVD1 = 00090A14 \\XDEMO\_CON_INIT
      RSVD2 = 00090A14 \\XDEMO\_CON_INIT
DEV01  INIT = 00000000 \\XDEMO\t32env\VN_SIO
       OPEN = 00000000 \\XDEMO\t32env\VN_SIO
```

Format: **TASK.QO** [<*id*> | <*name*>]

Displays the object-table of pSOS+.

With arguments it displays one object in detail.

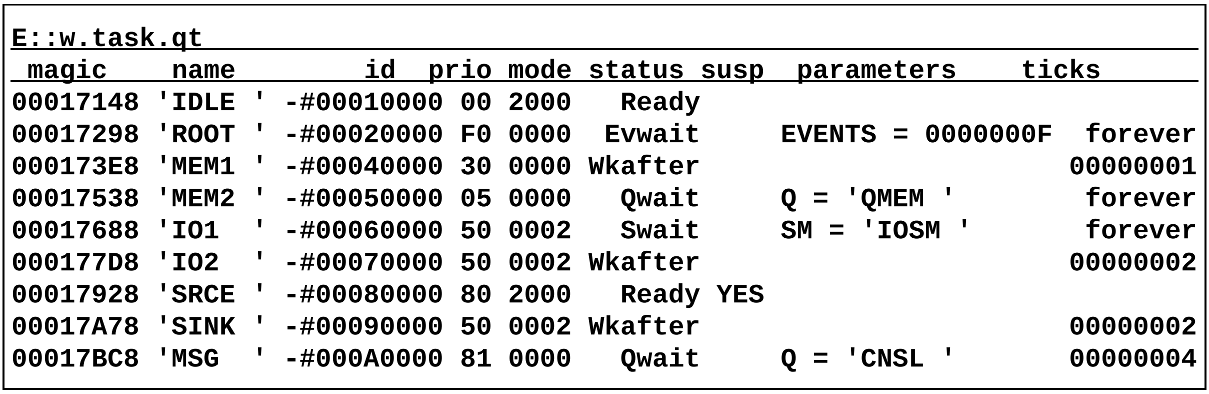


```
E::w.task.qo
  magic     name      ID      type
00016C48 'RN#0 ' 00000000 REGION
00016C68 'IDLE ' 00010000 TASK
00016C88 'ROOT ' 00020000 TASK
00016CA8 'IOSM ' 00030000 SEMAPHORE
00016CC8 'MEM1 ' 00040000 TASK
00016CE8 'MEM2 ' 00050000 TASK
00016E08 'RMEM ' 000E0000 REGION
```

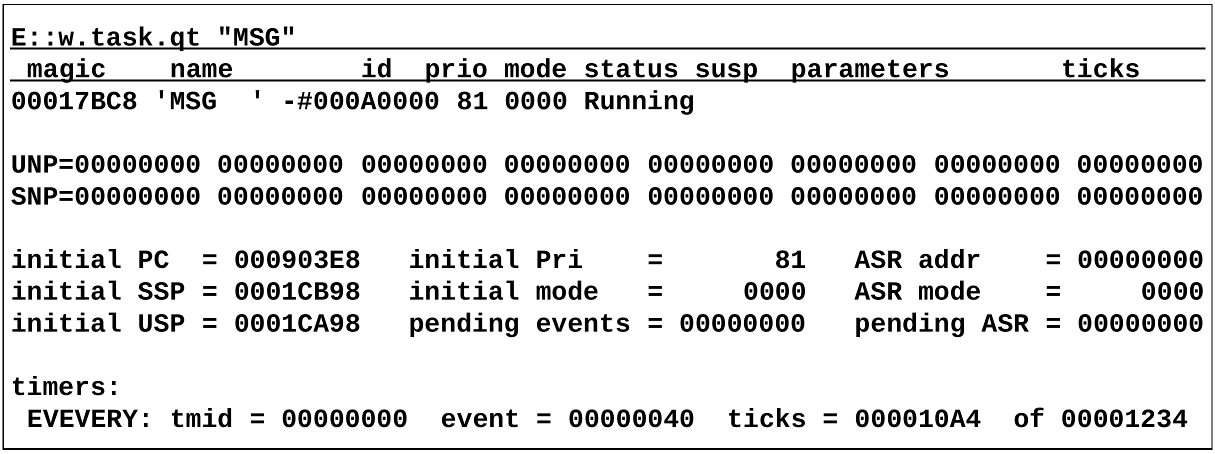
Format: **TASK.QT** [<*task_id*> **|** <*task_name*>]

Displays the task-table of pSOS+.

With arguments it displays one task in detail



```
E::w.task.qt
 magic    name        id  prio mode status susp  parameters    ticks
00017148 'IDLE ' -#00010000 00 2000   Ready
00017298 'ROOT ' -#00020000 F0 0000  Evwait     EVENTS = 0000000F  forever
000173E8 'MEM1 ' -#00040000 30 0000 Wkafter                       00000001
00017538 'MEM2 ' -#00050000 05 0000   Qwait     Q = 'QMEM '        forever
00017688 'IO1  ' -#00060000 50 0002   Swait     SM = 'IOSM '       forever
000177D8 'IO2  ' -#00070000 50 0002 Wkafter                       00000002
00017928 'SRCE ' -#00080000 80 2000   Ready YES
00017A78 'SINK ' -#00090000 50 0002 Wkafter                       00000002
00017BC8 'MSG  ' -#000A0000 81 0000   Qwait     Q = 'CNSL '       00000004
```

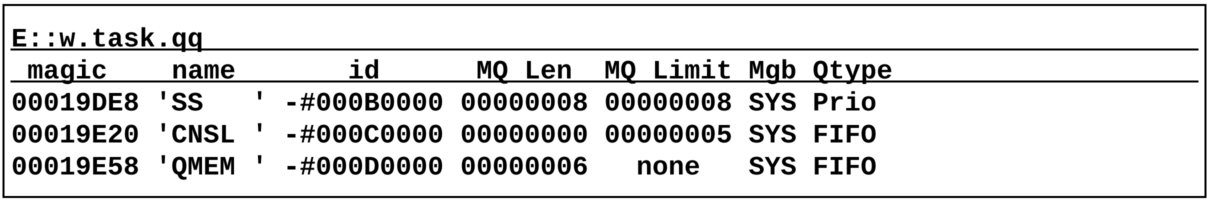


```
E::w.task.qt "MSG"
 magic    name        id  prio mode status susp  parameters       ticks
00017BC8 'MSG  ' -#000A0000 81 0000 Running

UNP=00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
SNP=00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

initial PC  = 000903E8   initial Pri    =       81   ASR addr     = 00000000
initial SSP = 0001CB98   initial mode   =     0000   ASR mode     =     0000
initial USP = 0001CA98   pending events = 00000000   pending ASR = 00000000

timers:
 EVEVERY: tmid = 00000000  event = 00000040  ticks = 000010A4  of 00001234
```

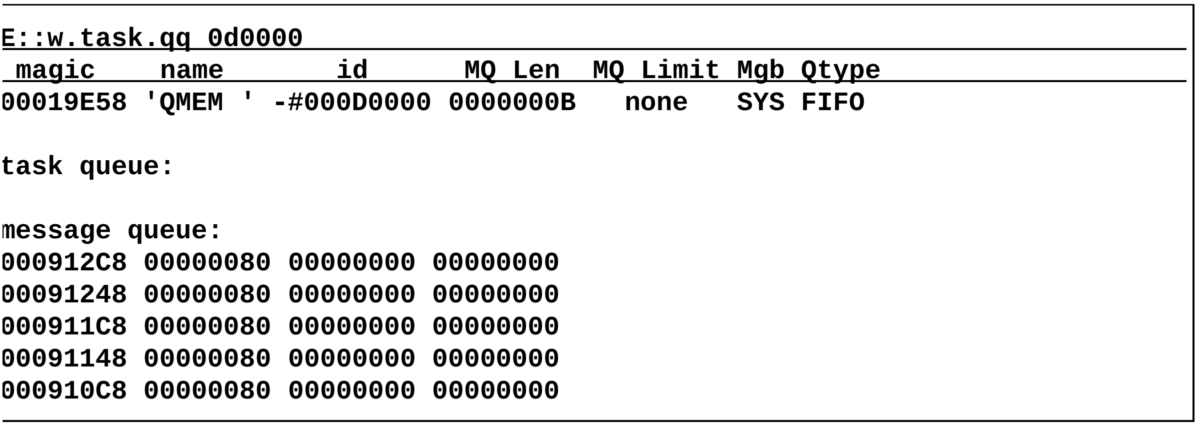
Format: **TASK.QQ** [<*queue_id*> | <*queue_name*>]

Displays the queue table of pSOS+.

With arguments it displays one queue in detail.



```
E::w.task.qq
 magic     name        id       MQ Len   MQ Limit Mgb Qtype
00019DE8 'SS    ' -#000B0000 00000008 00000008 SYS Prio
00019E20 'CNSL  ' -#000C0000 00000000 00000005 SYS FIFO
00019E58 'QMEM  ' -#000D0000 00000006   none   SYS FIFO
```

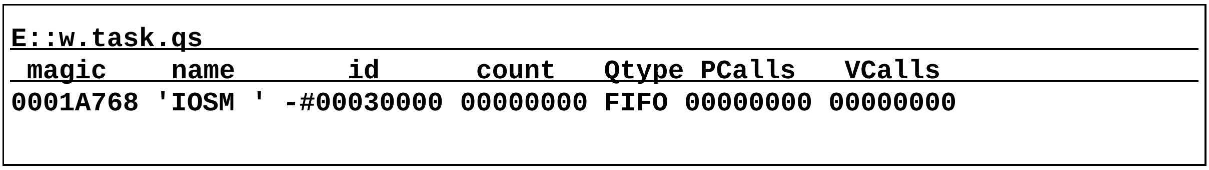


```
E::w.task.qq 0d0000
 magic     name        id       MQ Len   MQ Limit Mgb Qtype
00019E58 'QMEM  ' -#000D0000 0000000B   none   SYS FIFO

task queue:

message queue:
000912C8 00000080 00000000 00000000
00091248 00000080 00000000 00000000
000911C8 00000080 00000000 00000000
00091148 00000080 00000000 00000000
000910C8 00000080 00000000 00000000
```

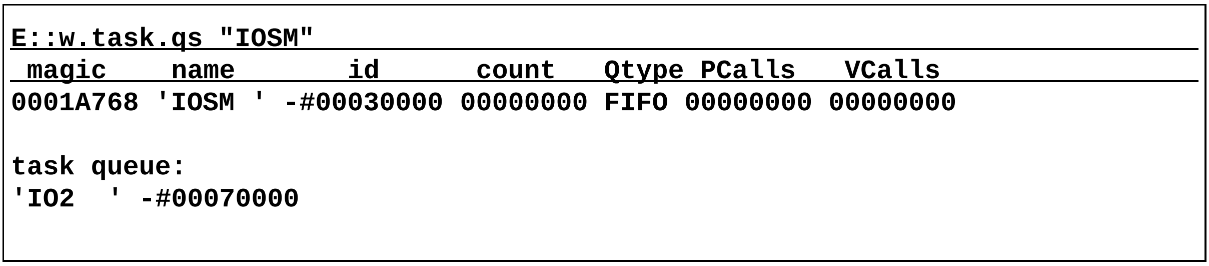
Format: **TASK.QS** [<*semaphore_id*> | <*semaphore_name*>]

Displays the semaphore table of pSOS+.

With arguments it displays one semaphore in detail.



```
E::w.task.qs
 magic    name         id       count    Qtype PCalls   VCalls
0001A768 'IOSM ' -#00030000 00000000 FIFO 00000000 00000000
```



```
E::w.task.qs "IOSM"
 magic    name         id       count    Qtype PCalls   VCalls
0001A768 'IOSM ' -#00030000 00000000 FIFO 00000000 00000000

task queue:
'IO2  ' -#00070000
```

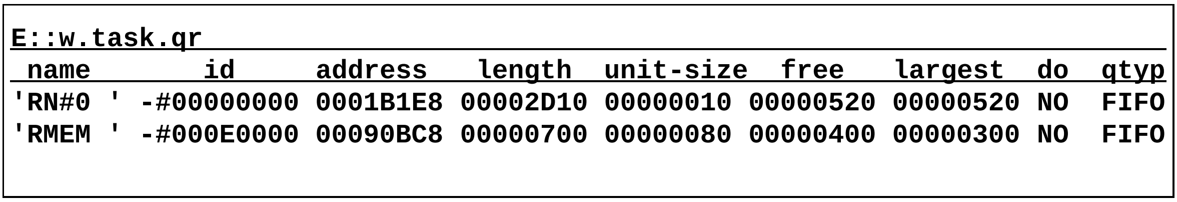
TASK.QR

Regions

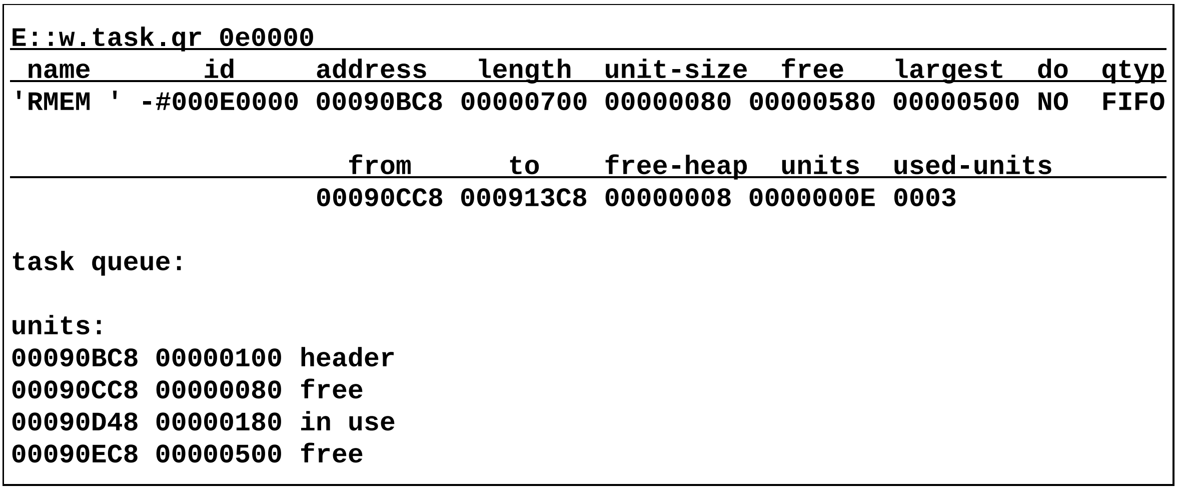
Format: **TASK.QR** [<*region_id*> | <*region_name*>]

Displays the region table of pSOS+.

With arguments it displays one region in detail.



```
E::w.task.qr
 name        id      address   length   unit-size  free    largest  do   qtyp
'RN#0 ' -#00000000 0001B1E8 00002D10 00000010 00000520 00000520 NO   FIFO
'RMEM ' -#000E0000 00090BC8 00000700 00000080 00000400 00000300 NO   FIFO
```



```
E::w.task.qr 0e0000
 name        id      address   length   unit-size  free    largest  do   qtyp
'RMEM ' -#000E0000 00090BC8 00000700 00000080 00000580 00000500 NO   FIFO

                     from       to      free-heap  units   used-units
                         00090CC8 000913C8 00000008 0000000E 0003

task queue:

units:
00090BC8 00000100 header
00090CC8 00000080 free
00090D48 00000180 in use
00090EC8 00000500 free
```

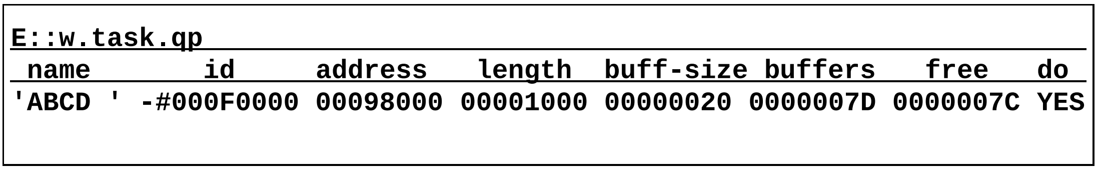
TASK.QP

Partitions

Format: **TASK.QP** [<*partition_id*> | <*partition_name*>]

Displays the partition table of pSOS+.

With arguments it displays one partition in detail.



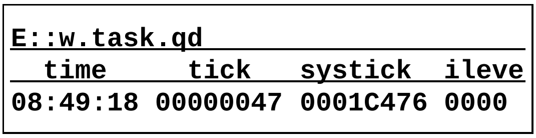
```
E::w.task.qp
 name        id      address   length   buff-size buffers   free    do
'ABCD ' -#000F0000 00098000 00001000 00000020 0000007D 0000007C YES
```

TASK.QD Date and time

Format: **TASK.QD**

Displays the current time and tick.

The 'ilevel' field displays the level of nested interrupts. This value should always be positive and near to zero.

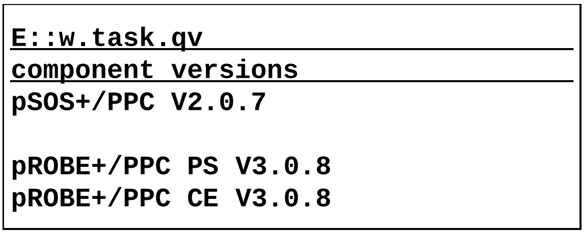


```
E::w.task.qd
  time      tick    systick  ileve
08:49:18 00000047 0001C476 0000
```

TASK.QV Version

Format: **TASK.QV**

Displays the pSOS+ and pROBE+ versions.



```
E::w.task.qv
component versions
pSOS+/PPC V2.0.7

pROBE+/PPC PS V3.0.8
pROBE+/PPC CE V3.0.8
```

TASK.SC System calls

Format: **TASK.SysCall** *<function> <d1> <d2> <d3> <d4> <d5> <a0> <a1>*

<function>: **PT_SGETBU | Q_BROADCA | EV_RECEIV | TM_WKAFTE**
TM_EVEVER | Q_VRECEIV | Q_VBROADC | Q_AVURGEN
<*function***>**

Executes a pSOS system call.

The function can only be executed, when the currently selected task is already stopped or can be stopped by the multitask debugger. When the task selective debugging is not active, the emulation must be stopped (in a regular task) before executing the command. Some functions are abbreviated to nine characters (see above list).

```
task.sc q_create 41424344 4 0 0 7        ; create new queue

task.sc q_send 0c0000 12 34 56 78        ; send message to queue
```

TASK.TASKState Mark task state words

Format: **TASK.TASKState**

This command sets Alpha breakpoints on all task status words.

The statistic evaluation of task states (see **Task State Analysis**) requires recording of the accesses to the task state words. By setting Alpha breakpoints to this words, and selectively recording Alpha’s, you can do a selective recording of task state transitions.

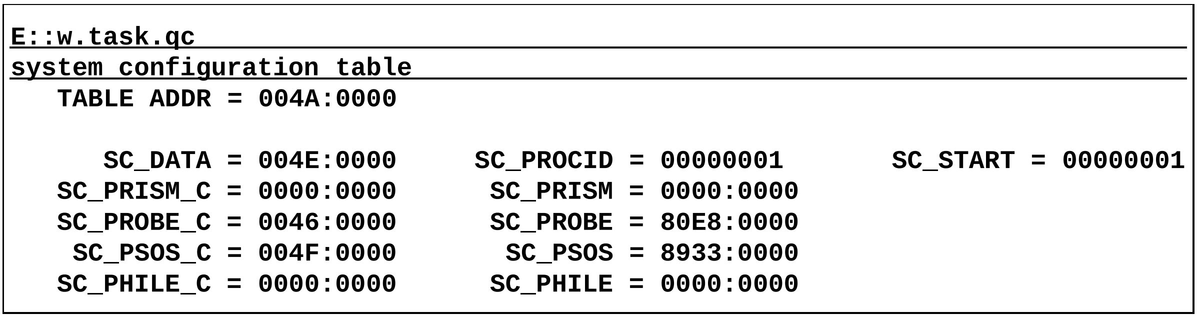
Because setting the Alpha breakpoints by hand is very hard to do, this utility command sets automatically the Alpha’s to the status words of all tasks currently created. It does NOT set breakpoints to tasks, that terminated or haven’t yet been created.

TASK.QC

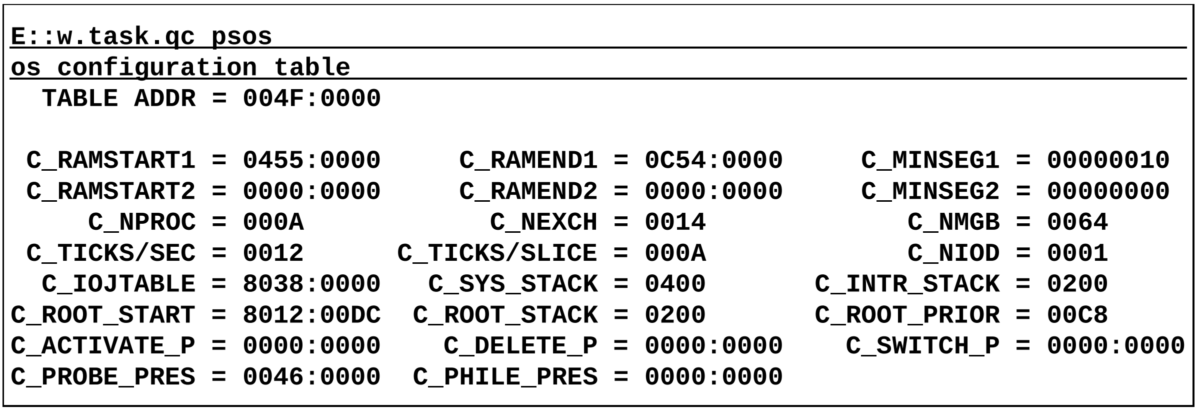
Configuration

Format: **TASK.QC [SYSTEM | PSOS | PROBE]**

Displays the configuration tables of pSOS+.



```
E::w.task.qc
system configuration table
   TABLE ADDR = 004A:0000

      SC_DATA = 004E:0000     SC_PROCID = 00000001       SC_START = 00000001
   SC_PRISM_C = 0000:0000      SC_PRISM = 0000:0000
   SC_PROBE_C = 0046:0000      SC_PROBE = 80E8:0000
    SC_PSOS_C = 004F:0000       SC_PSOS = 8933:0000
   SC_PHILE_C = 0000:0000      SC_PHILE = 0000:0000
```



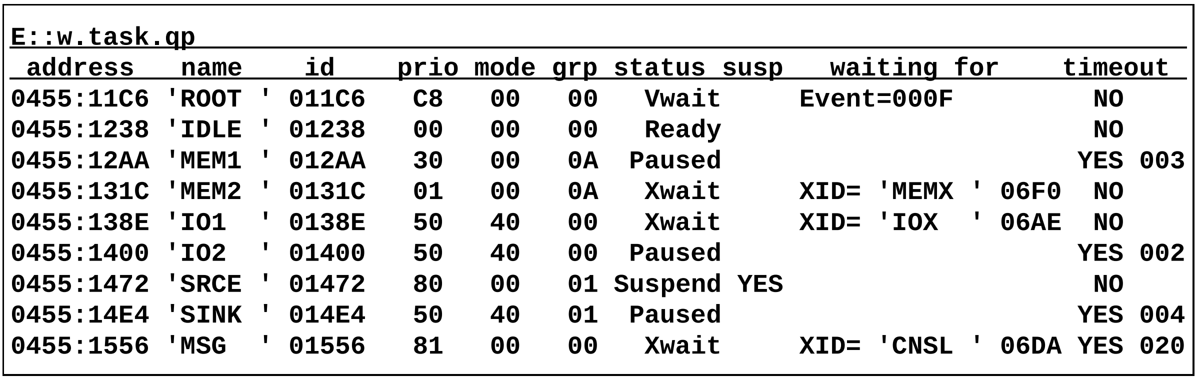
```
E::w.task.qc psos
os configuration table
  TABLE ADDR = 004F:0000

 C_RAMSTART1 = 0455:0000     C_RAMEND1 = 0C54:0000     C_MINSEG1 = 00000010
 C_RAMSTART2 = 0000:0000     C_RAMEND2 = 0000:0000     C_MINSEG2 = 00000000
     C_NPROC = 000A            C_NEXCH = 0014             C_NMGB = 0064
 C_TICKS/SEC = 0012      C_TICKS/SLICE = 000A             C_NIOD = 0001
  C_IOJTABLE = 8038:0000   C_SYS_STACK = 0400       C_INTR_STACK = 0200
C_ROOT_START = 8012:00DC  C_ROOT_STACK = 0200       C_ROOT_PRIOR = 00C8
C_ACTIVATE_P = 0000:0000    C_DELETE_P = 0000:0000    C_SWITCH_P = 0000:0000
C_PROBE_PRES = 0046:0000  C_PHILE_PRES = 0000:0000
```

TASK.QP Process table

Format: **TASK.QP**

Displays the process table of pSOS-86.

The state 'Running' is not displayed.



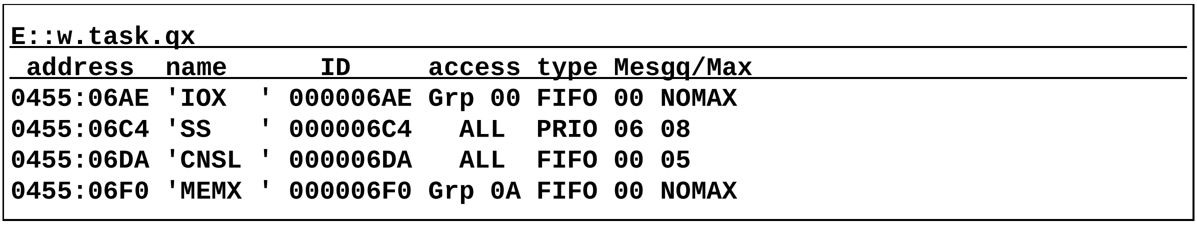
```
E::w.task.qp
 address   name    id    prio mode grp status susp   waiting for    timeout
0455:11C6 'ROOT ' 011C6   C8   00   00   Vwait     Event=000F         NO
0455:1238 'IDLE ' 01238   00   00   00   Ready                        NO
0455:12AA 'MEM1 ' 012AA   30   00   0A  Paused                       YES 003
0455:131C 'MEM2 ' 0131C   01   00   0A   Xwait     XID= 'MEMX ' 06F0  NO
0455:138E 'IO1  ' 0138E   50   40   00   Xwait     XID= 'IOX  ' 06AE  NO
0455:1400 'IO2  ' 01400   50   40   00  Paused                       YES 002
0455:1472 'SRCE ' 01472   80   00   01 Suspend YES                    NO
0455:14E4 'SINK ' 014E4   50   40   01  Paused                       YES 004
0455:1556 'MSG  ' 01556   81   00   00   Xwait     XID= 'CNSL ' 06DA YES 020
```

TASK.QX Exchange table

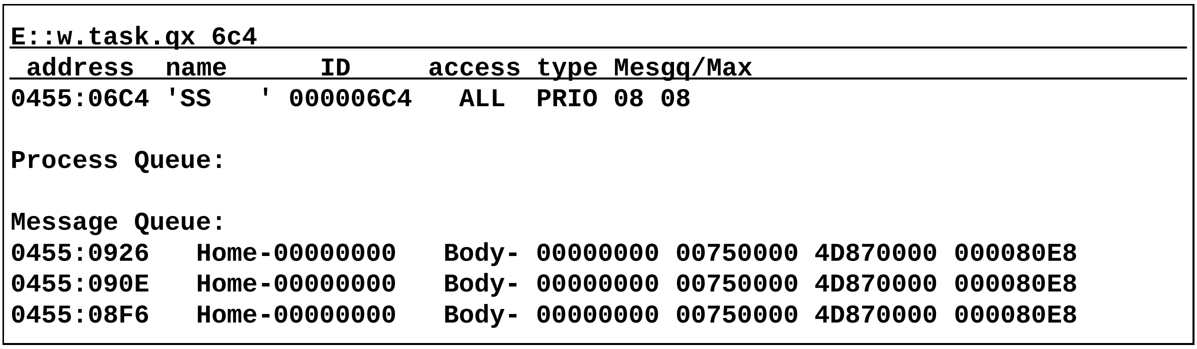
Format: **TASK.QX** [<*id*>]

Displays the exchange table of pSOS-86.

With arguments it displays one exchange in detail.



```
E::w.task.qx
 address  name      ID      access type Mesgq/Max
0455:06AE 'IOX  ' 000006AE Grp 00 FIFO 00 NOMAX
0455:06C4 'SS   ' 000006C4   ALL  PRIO 06 08
0455:06DA 'CNSL ' 000006DA   ALL  FIFO 00 05
0455:06F0 'MEMX ' 000006F0 Grp 0A FIFO 00 NOMAX
```



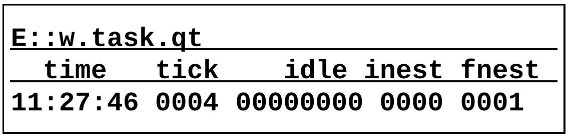
```
E::w.task.qx 6c4
 address  name      ID      access type Mesgq/Max
0455:06C4 'SS   ' 000006C4   ALL  PRIO 08 08

Process Queue:

Message Queue:
0455:0926   Home-00000000   Body- 00000000 00750000 4D870000 000080E8
0455:090E   Home-00000000   Body- 00000000 00750000 4D870000 000080E8
0455:08F6   Home-00000000   Body- 00000000 00750000 4D870000 000080E8
```

TASK.QT

Time

Format: **TASK.QT**

Displays the current time and the nesting levels of system calls and interrupts.



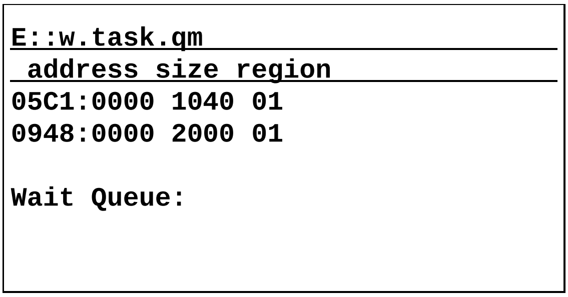
```
E::w.task.qt
  time    tick     idle inest fnest
11:27:46 0004 00000000 0000 0001
```

TASK.QM

Memory

Format: **TASK.QM**

Displays the free memory blocks and processes waiting for free memory.



```
E::w.task.qm
 address size region
05C1:0000 1040 01
0948:0000 2000 01

Wait Queue:
```

TASK.TASKState

Mark task state words

Format: **TASK.TASKState**

This command sets Alpha breakpoints on all task status words.

The statistic evaluation of task states (see **Task State Analysis**) requires recording of the accesses to the task state words. By setting Alpha breakpoints to this words, and selectively recording Alpha’s, you can do a selective recording of task state transitions.

Because setting the Alpha breakpoints by hand is very hard to do, this utility command sets automatically the Alpha’s to the status words of all tasks currently created. It does NOT set breakpoints to tasks, that terminated or haven’t yet been created.

pSOS PRACTICE Functions

There are special definitions for pSOS specific PRACTICE functions.

TASK.CONFIG(<*item*>**)**	Reports configuration parameters.
TASK.CONFIG(magic)	Returns the address for the magic number
TASK.CONFIG(magicsize)	Returns the size of the magic number (1, 2 or 4)