

RTXC™ Quadros™ Kernel Awareness
User Manual
for
Metrowerks™ CodeWarrior® Debugger,
ARM® RealView® Debugger,
Keil™ μVision™,
IAR™ Embedded Workbench Debugger,
and
Intel® XDB Debugger

Revision History:

Draft 15	07/18/08	Added instructions for RealView with Eclipse
Draft 14	05/09/08	Added instructions for Keil μ Vision
Draft 13	03/13/08	Clarified IAR EWB installation instructions
Draft 12	08/04/06	Corrected installation instructions for ARM RealView Debugger
Draft 11	05/24/06	Add instructions for Intel XDB Debugger
Draft 10	10/19/04	Update instructions for version 4.x of IAR Embedded Workbench
Draft 9	04/06/03	Add instructions for IAR Embedded Workbench
Draft 8	11/25/03	Add instructions for ARM RealView Debugger
Draft 7	07/17/02	Update screen snapshots, add Kernel Information Display section
Draft 6	07/09/02	Minor cleanups.
Draft 5	11/13/01	Update screen snapshots, minor cleanups.
Draft 4	3/27/01	Fix TOC, trademarks; Minor cleanups; Add menu snapshots.
Draft 3	3/27/01	Update screen snapshots, add chapter on Task Semaphores display.
Draft 2	3/22/01	Form chapters, add Table of Contents, Cover Page, standardize terminology
Draft 1	3/21/01	Display descriptions only

NOTE: Example screens in this document were generated from a PowerPC target application, but will be illustrative for all supported targets.

Table of Contents

- [Introduction](#)
- [Installing Kernel Awareness](#)
- [Enabling Kernel Awareness](#)
- [Initializing Kernel Awareness](#)
- [Using Kernel Awareness](#)
- [The Displays](#)
 - [The **Threads** Display](#)
 - [The **Level Information** Display](#)
 - [The **Thread Schedule Information** Display](#)
 - [The **Thread Gate Information** Display](#)
 - [The **Exception Backtrace** Display](#)
 - [The **Tasks** Display](#)
 - [The **Task Register Information** Display](#)
 - [The **Task Stacks Information** Display](#)
 - [The **Task Semaphores Information** Display](#)
 - [The **Task Ready List** Display](#)
 - [The **Semaphores** Display](#)
 - [The **Mailboxes** Display](#)
 - [The **Message Entry Information** Display](#)
 - [The **Mailbox Semaphores** Display](#)
 - [The **Partitions** Display](#)
 - [The **Partition Block Information** Display](#)
 - [The **Partition Semaphores** Display](#)
 - [The **Queues** Display](#)
 - [The **Queue Entry Information** Display](#)
 - [The **Queue Semaphores** Display](#)
 - [The **Mutexes** Display](#)
 - [The **Mutex Semaphores** Display](#)
 - [The **Pipes** Display](#)
 - [The **Full Buffer Addresses** Display](#)
 - [The **Empty Buffer Addresses** Display](#)
 - [The **Exceptions** Display](#)
 - [The **Event Sources** Display](#)
 - [The **Counters** Display](#)
 - [The **Alarms** Display](#)
 - [The **Alarm Semaphores** Display](#)
 - [The **Kernel Information** Display](#)

Introduction

RTXC Quadros Kernel Awareness is a plug-in extension to the Metrowerks CodeWarrior IDE, ARM's RealView debugger, Keil's μ Vision IDE, IAR's Embedded Workbench, and Intel's XDB Debugger.

The respective debugging environments, being general purpose, have no idea which Real-Time Operating System (RTOS) kernel is running on the target, only the processor type, and therefore cannot display kernel-specific data structures to you in a meaningful way.

Quadros Kernel Awareness allows you to view Quadros-specific objects in an easy-to-read fashion, rather than leaving you to interpret the raw memory and register data that the debugger provides.

When you begin a debug session for your application on the target processor, Kernel Awareness makes itself available to you as either as a menu selection on the IDE's main menu bar (for CodeWarrior and XDB), or as a floating "control panel" (for RealView, μ Vision, and Embedded Workbench). Once it and the kernel have been initialized, then at any time the application comes to a breakpoint, you can use Kernel Awareness to view Quadros-specific objects such as threads, tasks, queues, etc., in addition to using all the standard debugging features.

Installing Kernel Awareness

For CodeWarrior:

The DLL file that you were supplied (whose name depends on the target processor) must be copied into the appropriate folder in the \Program Files\Metrowerks\CodeWarrior directory tree. Usually, the subdirectory is called Rtos, and can be located as {CodeWarrior}\Bin\Plugins\Debugger\Rtos. In many RTX Quadros installations, the DLL is automatically installed.

The DLL names for the various Metrowerks versions are listed below:

Metrowerks version	DLL name
ARM	ARMrtos_Quadros.dll
ColdFire	CFrtos_Quadros.dll
DSP56800 (Hawk1)	Quadros_K3_Plugin.dll
MCore	MCORErtos_Quadros.dll
PowerPC	EPPCrtos_Quadros.dll
StarCore	SC100rtos_Quadros.dll

For RealView:

Copy the following modules from your {Quadros}\bin directory into the RealView directory tree:

File	RealView directory
wl_armex.dll	{RVD}\lib\custom
e1_quaka.dll	{RVD}\lib\custom

If RealView is running, you must restart it for the new files to be recognized.

For Embedded Workbench:

The Quadros installation program installs Quadros_EWB5_Plugin.dll and Quadros_EWB5_Plugin.ewplugin into your {Quadros}\bin directory. Copy these files to the {IAR}\ARM\plugins\rtos\Quadros directory (you will have to create the Quadros directory).

Quadros_EWB5_Plugin.ewplugin contains XML code describing the Kernel Awareness module for the EWB IDE. You may need to edit this file, changing the <dllFile> entry to point to the location of your Quadros_EWB5_Plugin.dll.

If Embedded Workbench is running, you must restart it for the file to be recognized.

For **Keil μ Vision:**

The Quadros installation program installs uVision-ARMrtos_Quadros.DLL into your {Quadros}\bin directory. Copy this file to the {Keil}\ARM\bin directory.

Open the {Keil}\tools.ini file in a text editor. Locate the section of the file beginning with the [ARMADS] section header. To this section, add the following entry:

```
RTOSn=uVision-ARMrtos_Quadros.DLL ("RTXC Quadros")
```

where 'n' in 'RTOSn' is the smallest unused digit from 1 to 6. For example, if entries already exist for RTOS0 and RTOS1, your entry should be

```
RTOS2= uVision-ARMrtos_Quadros.DLL ("RTXC Quadros")
```

Save this file and restart μ Vision.

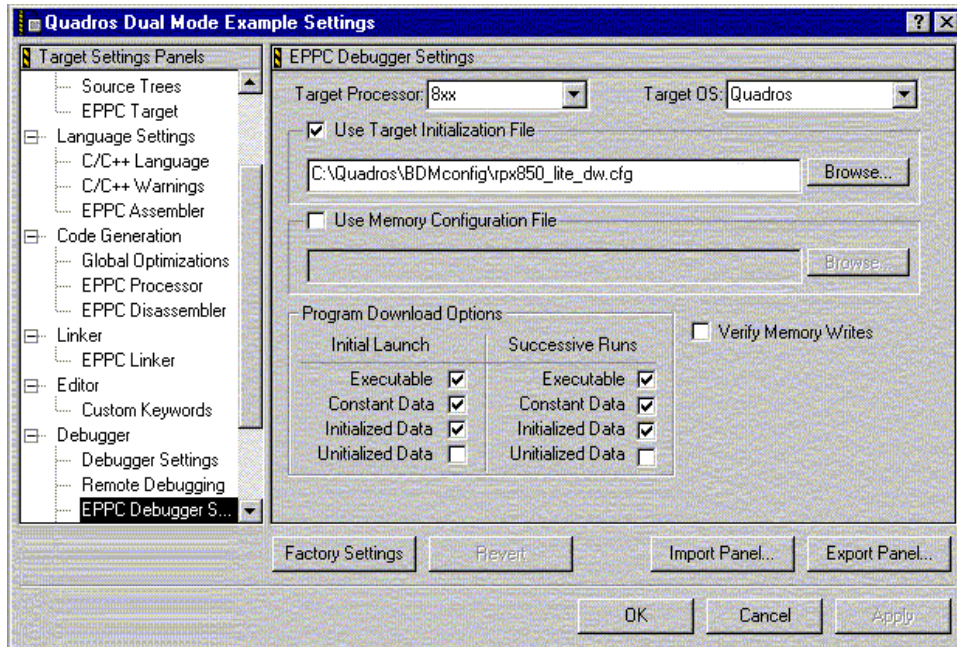
For **XDB:**

The DLL file that you were supplied does not need to be installed in any particular folder. Follow the steps in the next section to enable Kernel Awareness in XDB.

Enabling Kernel Awareness

For CodeWarrior:

In CodeWarrior, open the project for the application you wish to debug, then select its Settings panel. Select the **XXXX Debugger Settings** (where "XXXX" represents the target processor type) item under **Debugger**. That brings up the following display:

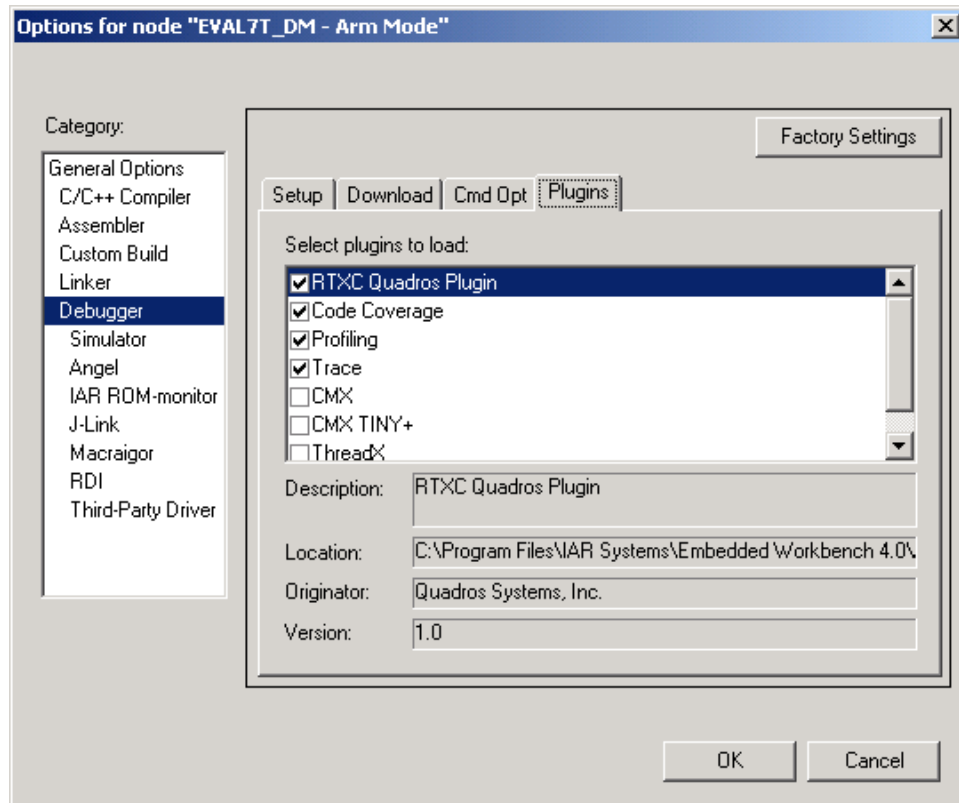


For **Target OS**, select **Quadros**.

You are now ready to debug your application using Kernel Awareness.

For Embedded Workbench:

Select Project/Options, and then highlight "Debugger". The dialog should appear as follows:

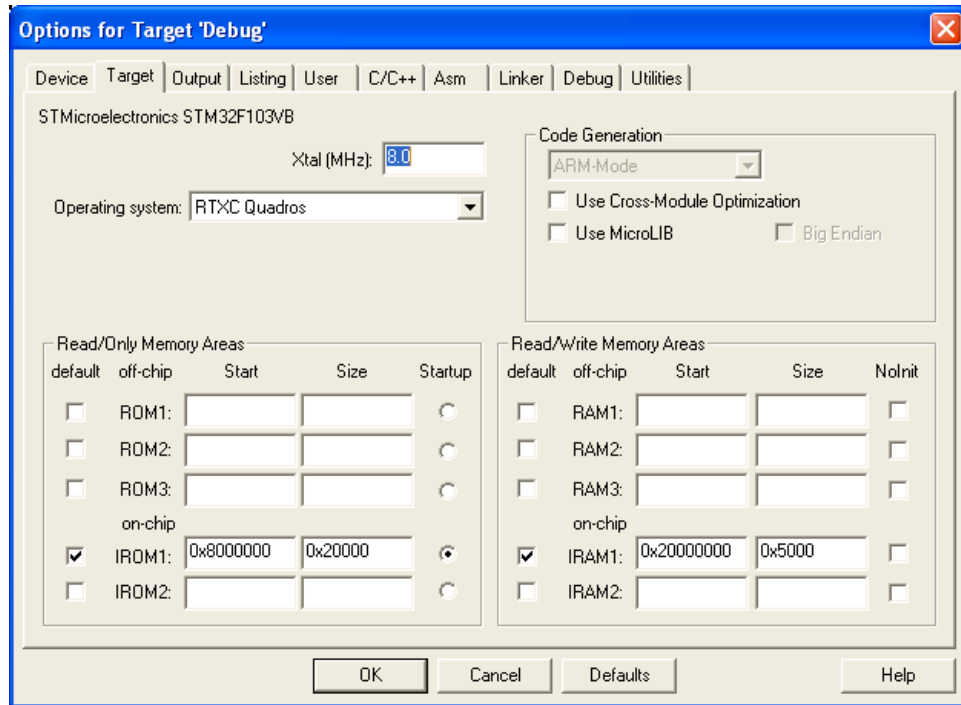


Select the box labeled “RTXC Quadros Plugin”. You are now ready to begin using Kernel Awareness.

For Keil μ Vision:

With your project open in μ Vision, select the root of the project tree in the Project Workspace window, select *Project->Options for Target ‘*, and then select the *Target* tab. Select *RTXC Quadros* from the *Operating system* drop-down box.

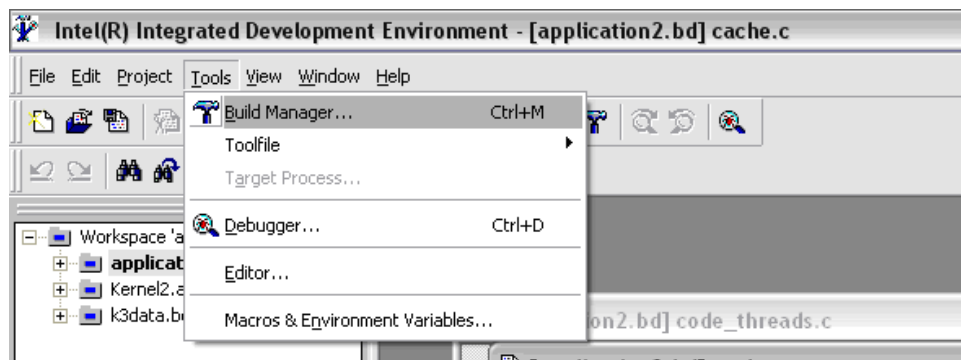
The *Operating system* selection should appear as follows:



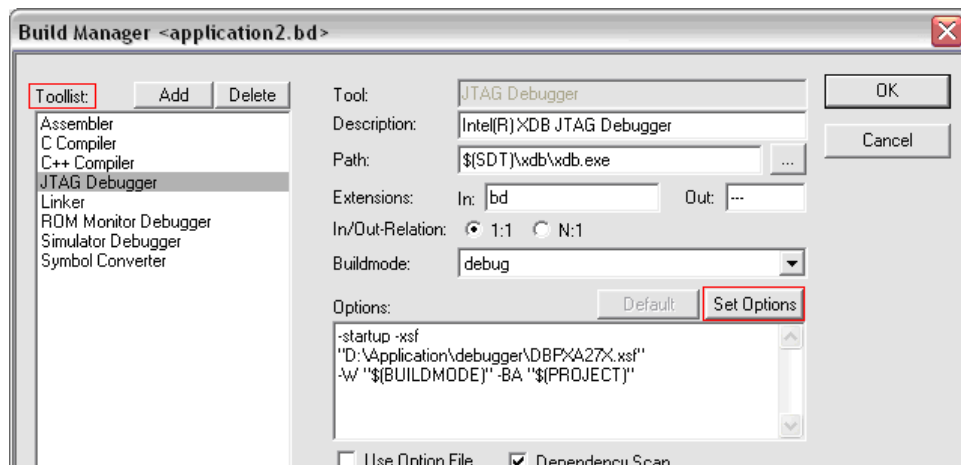
Select OK.

For **XDB**:

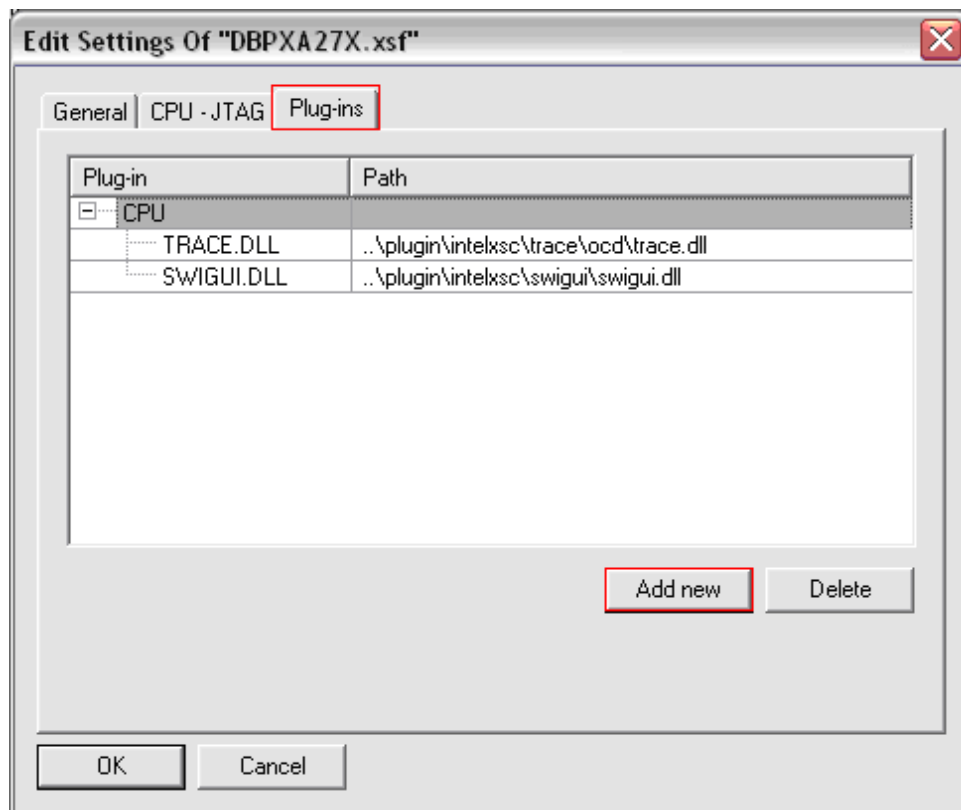
The DLL file that you were supplied must be added to the **Plug-ins** list in the debugger options window of the Intel Integrated Development Environment. To access the debugger options, first select the **Tools** menu option, and then select **Build Manager**.



Next, select the appropriate debugger from the **Toollist** and press the **Set Options** button.



In this window, select the **Plug-ins** tab and press the **Add new** button.



Now locate and select the supplied Kernel Awareness DLL file.

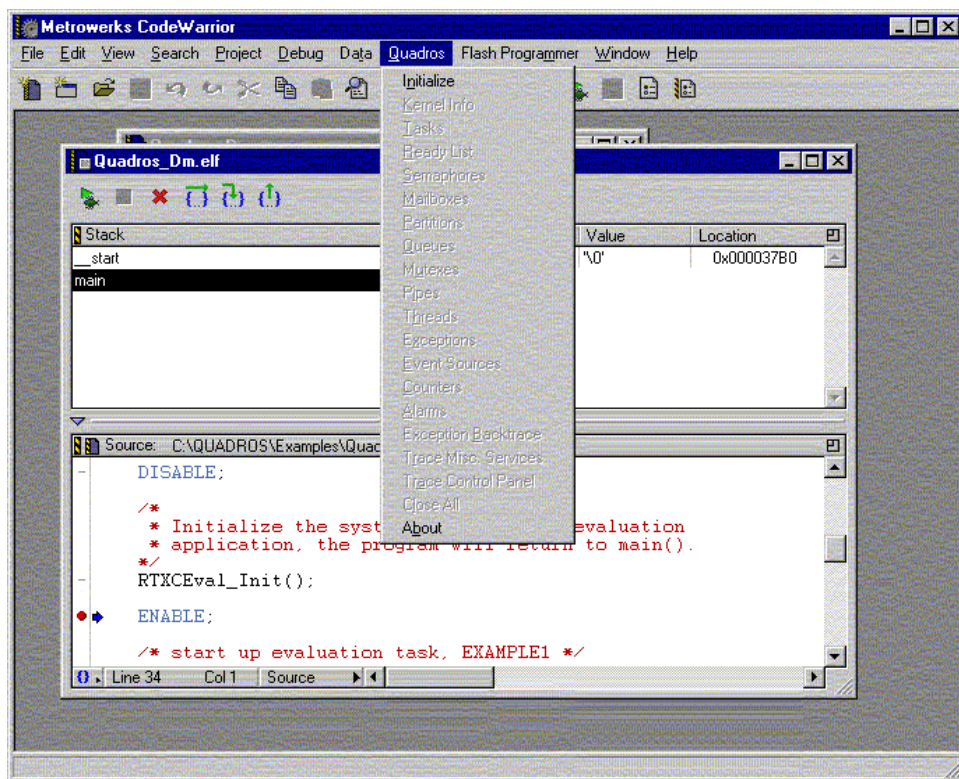
Initializing Kernel Awareness

For CodeWarrior:

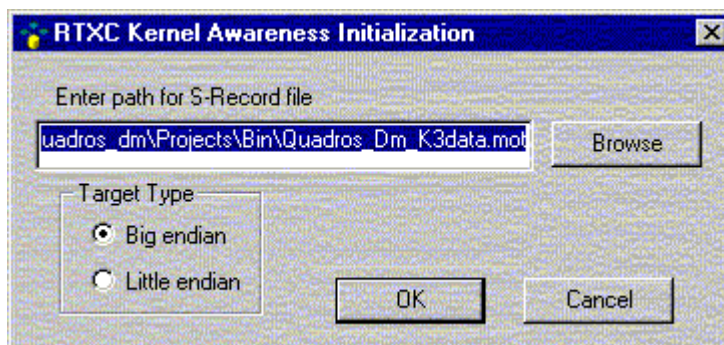
With your project open in CodeWarrior, click the Debug button (it looks like a rightward-pointing arrow). You must let your application run to some point after the call to `rtxcinit()`.

Once you are at a breakpoint, and the initialization routine has already been called, you are ready to begin using Kernel Awareness.

To do so, click the **Quadros** menu item, then select **Initialize**, as shown in the following figure:



The following dialog will appear:



In this dialog, you must select the S-record file (k3data.*, NOTE: file name may be preceded with target description, e.g. PowerPcK3data.*) which is appropriate to your application. Among other things, this file tells Kernel Awareness where to locate the Quadros kernel's anchor point within the target, among other data.

Either type in the full path of file name of the file, or use the **Browse** button to navigate to it. Note: the file extension used for the k3data file is specific to the version of CodeWarrior being used. Consult your binding manual.

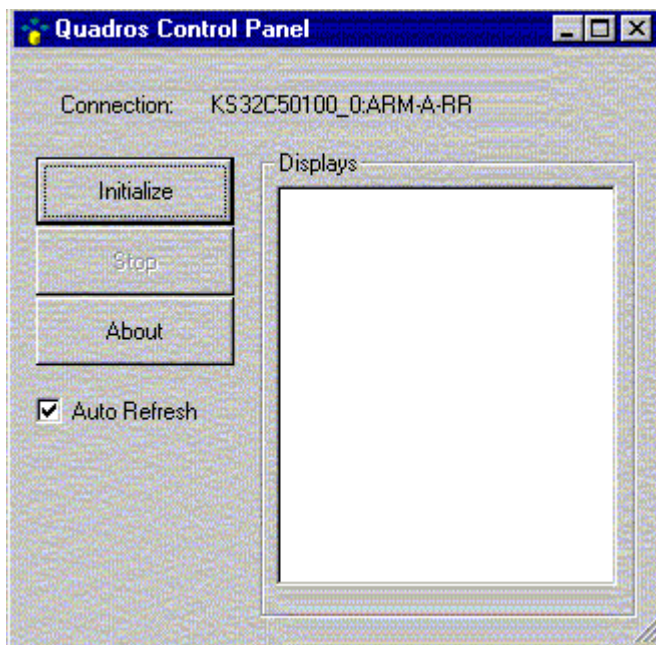
For bi-endian targets, you must also specify whether the target is big endian or little endian.

Click the **OK** button to complete the initialization.

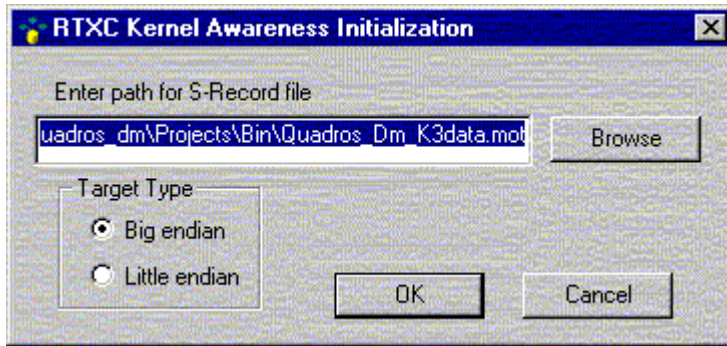
Once initialization has completed successfully, you are ready to begin using the Quadros Kernel Awareness displays, described in the section, [Using Kernel Awareness](#).

For RealView (with Eclipse):

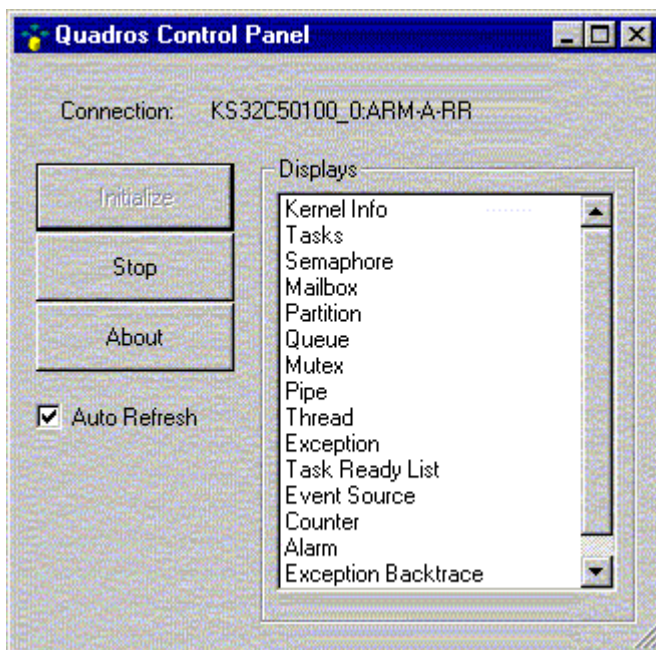
After starting the RealView debugger from the Eclipse IDE, start execution of your application. On the first breakpoint you hit after rtxcinit() is called, you will be able to initialize the Kernel Awareness plug-in. A Quadros Kernel Awareness control panel will exist for the connection, as shown:



Click the "Initialize" button, and the following dialog will appear:



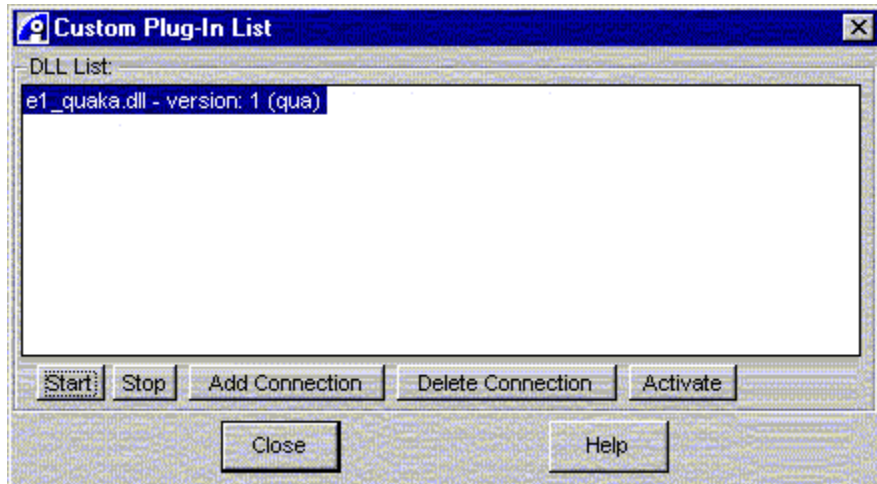
When you have selected an S-record file, the "Display" box will be populated with the valid displays for your Quadros kernel, as follows:



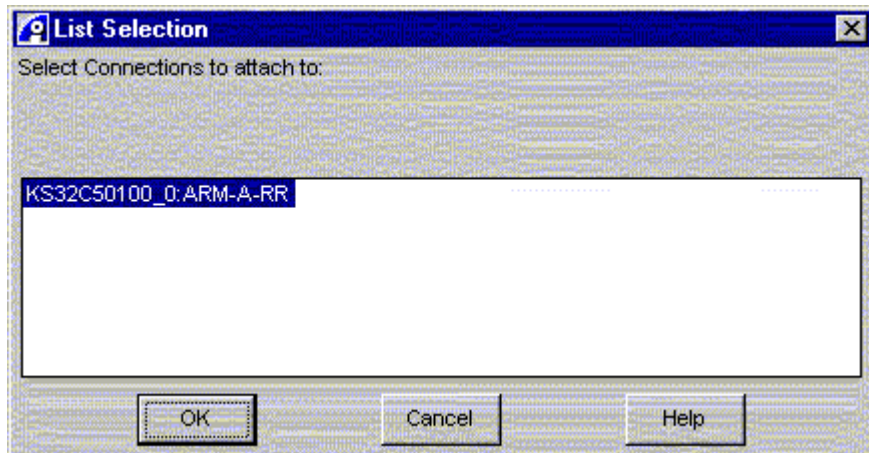
For **RealView (without Eclipse):**

After starting the Multi-ICE Server, or other connection method, start the RealView debugger, and then load your application image into the target and start execution. On the first breakpoint you hit after `rtxcinit()` is called, select RealView's Custom Plug-In control panel, which floats independently of RealView's main window. (Note that the Custom Plug-in control panel does not appear on the Windows taskbar and can sometimes be hidden by the RealView debugger main window.)

This appears as shown:



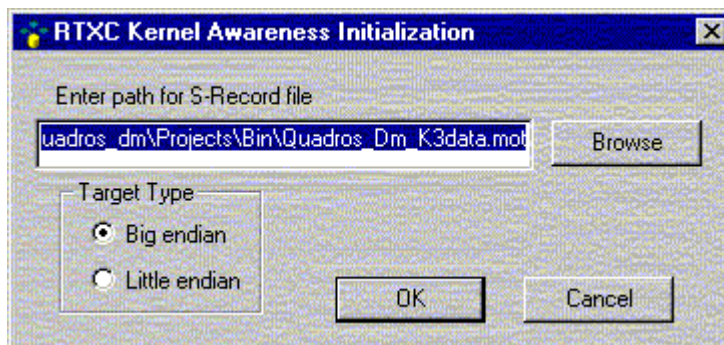
You can now highlight the entry for "e1_quaka.dll", click "Start", then click "Add Connection." The following dialog will appear:



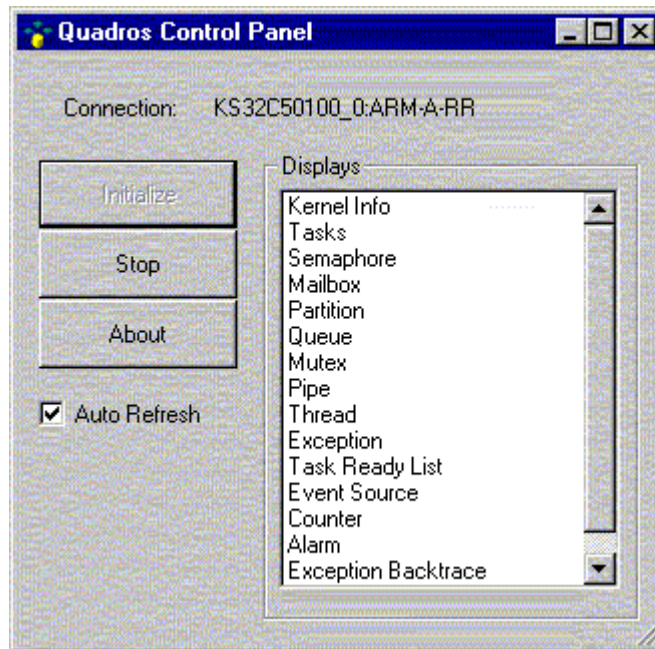
Highlight the desired connection, and then click "OK". (Note: the example shown is for an Eval7T board). A Quadros Kernel Awareness control panel will appear for that connection, as follows:



Click the "Initialize" button, and the following dialog will appear:

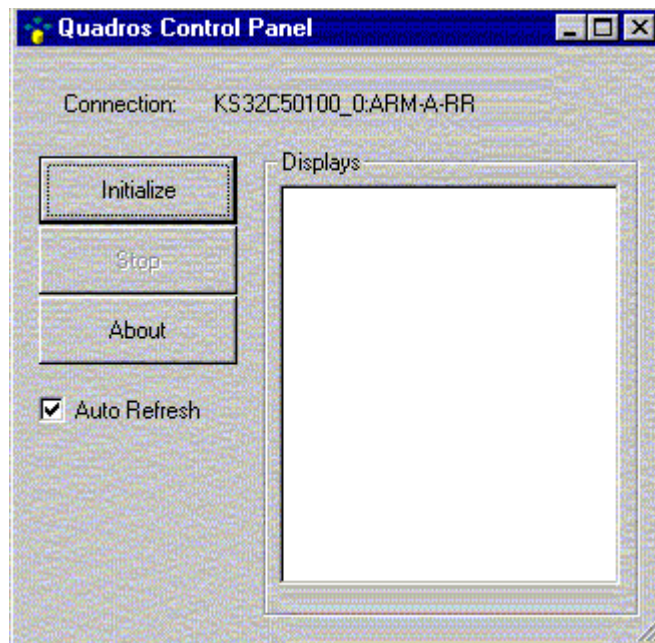


When you have selected an S-record file, the "Display" box will be populated with the valid displays for your Quadros kernel, as follows:

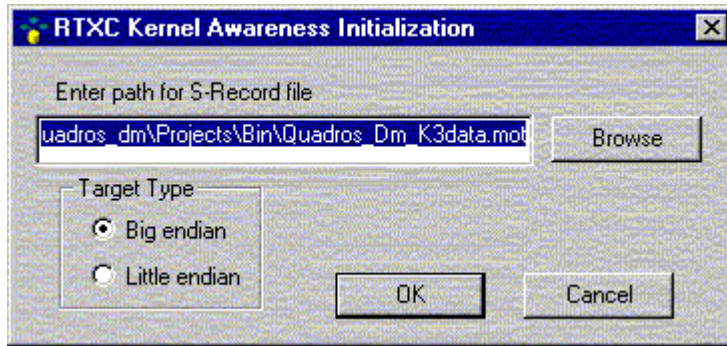


For **Embedded Workbench**:

Start the Embedded Workbench debugger, then load your application image into the target and start execution. The following dialog will appear (it may initially be behind the IDE):

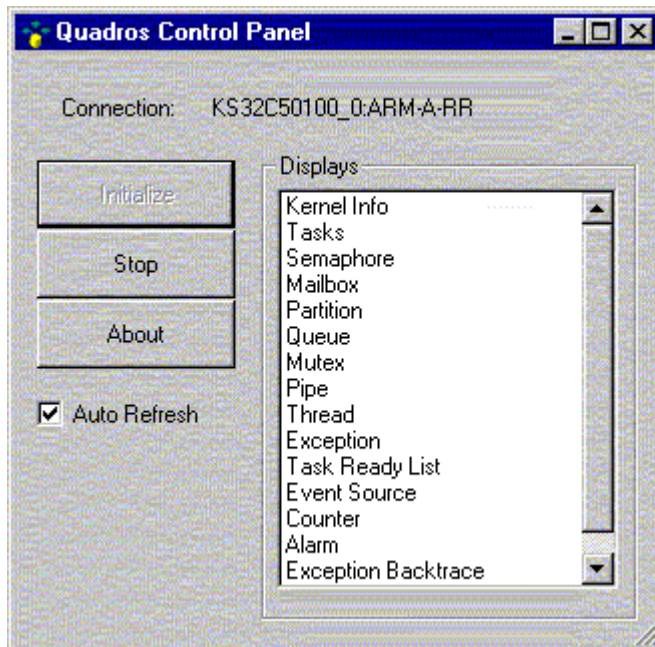


After you have run the application to some point after the call to `rtxcinit()`, click the "Initialize" button, and then the following dialog will appear:



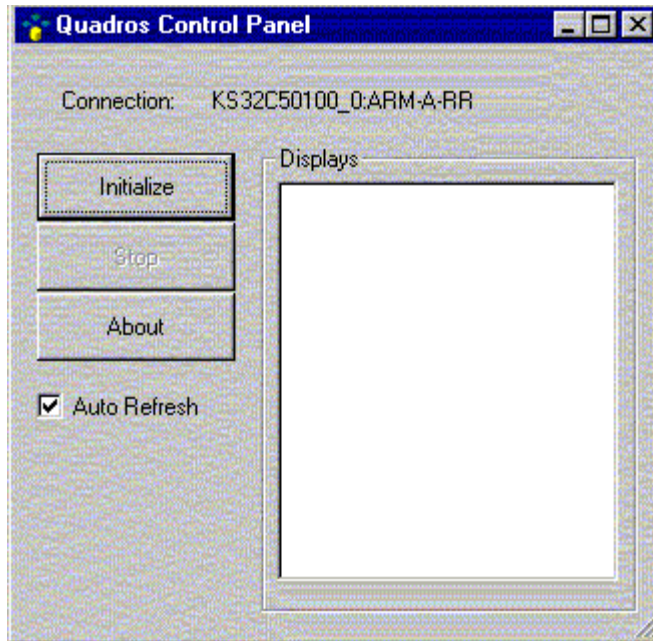
Enter or browse to the path of the appropriate k3data file for your application, choose the target processor's endianness (if applicable), and then click "OK". (Note: The k3data file must be built alongside the main application binary. See the Quadros SDK documentation for instructions).

After this, the **Display** box will be populated with the valid displays for your Quadros kernel, as follows:

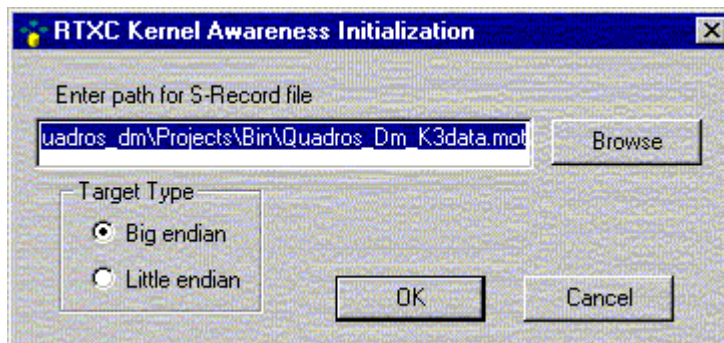


For Keil μ Vision:

Start the Keil μ Vision IDE, then load your application image into the target and start execution. The following dialog will appear:

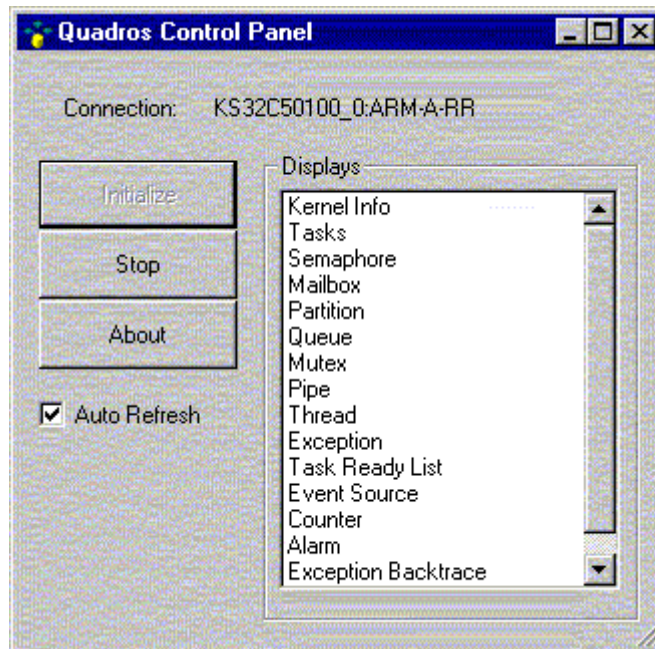


After you have run the application to some point after the call to `rtxcinit()`, click the "Initialize" button, and then the following dialog will appear:



Enter or browse to the path of the appropriate k3data file for your application, choose the target processor's endian-ness (if applicable), and then click "OK" (Note: The k3data file must be built alongside the main application binary. See the Quadros SDK documentation for instructions).

After this, the **Display** box will be populated with the valid displays for your Quadros kernel, as follows:

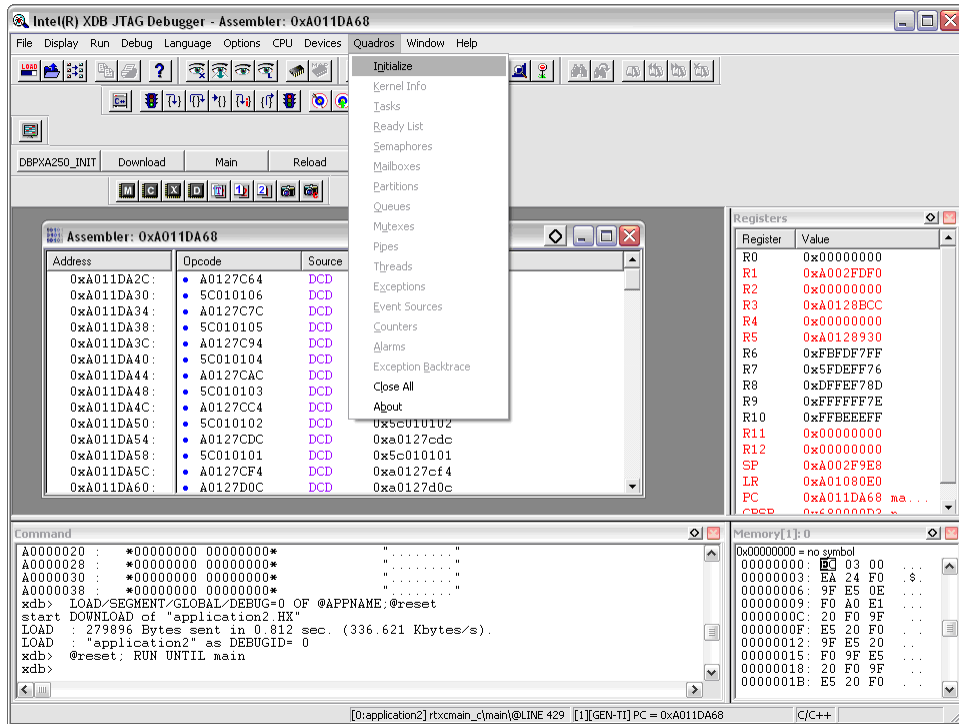


For XDB:

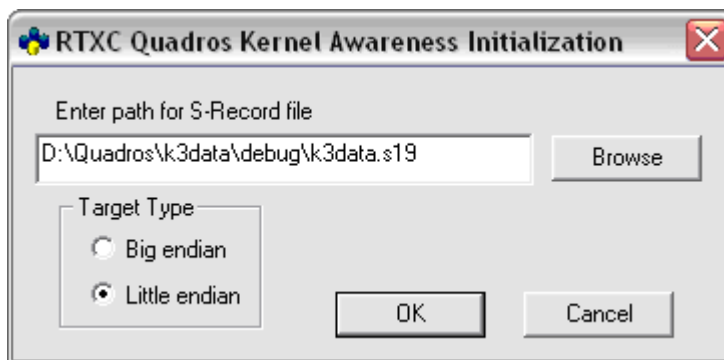
With your project open in the Intel Integrated Development Environment, click the Debugger button (it looks like a bug beneath a magnifying glass). You must let your application run to some point after the call to `rtxcinit()`.

Once you are at a breakpoint and the initialization routine has already been called, you are ready to begin using Kernel Awareness.

To call the initialization routine, click the **Quadros** menu item and select **Initialize**, as shown in the following figure:



The following dialog will appear:



In this dialog, you must select the S-record file (k3data.*, NOTE: file name may be preceded with target description) which is appropriate to your application. Among other things, this file tells Kernel Awareness where to locate the Quadros kernel's anchor point within the target.

Either type in the full path of file name of the file, or use the **Browse** button to navigate to it. For bi-endian targets, you must also specify whether the target is big endian or little endian.

Click the **OK** button to complete the initialization.

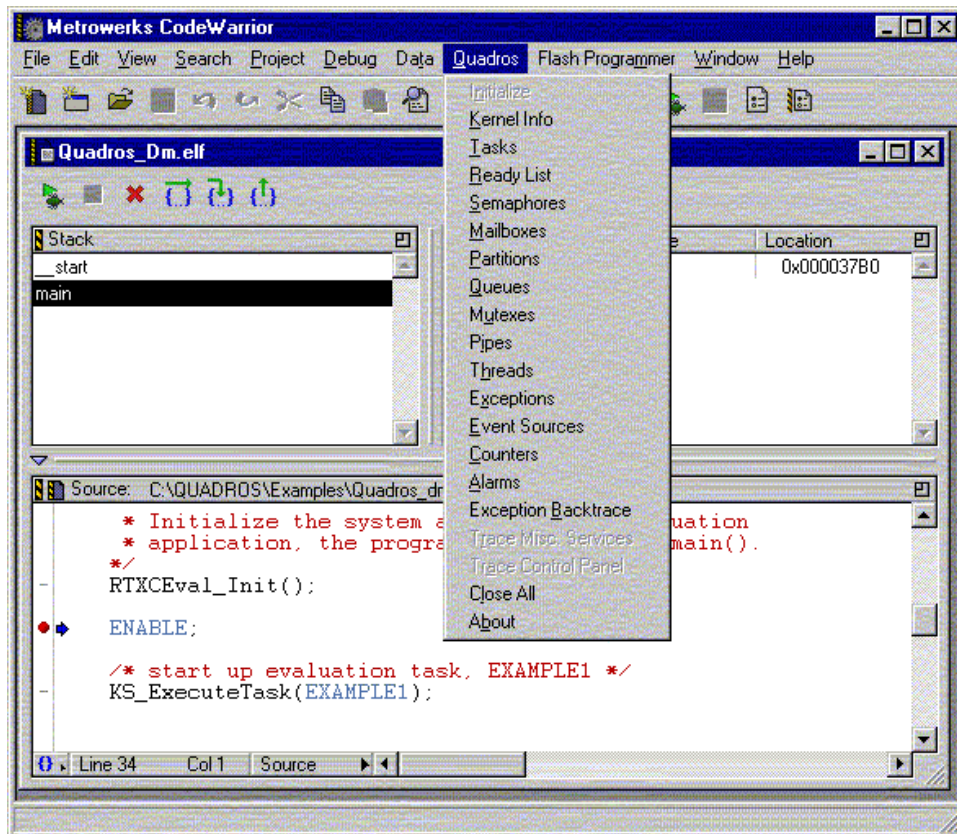
Once initialization has completed successfully, you are ready to begin using the Quadros Kernel Awareness displays, described in the section, [Using Kernel Awareness](#).

Using Kernel Awareness

Once you have initialized Kernel Awareness (see [Initializing Kernel Awareness](#)), you can begin viewing Kernel Awareness displays, which are the same for the all debuggers. The only difference is in the way they are started.

For CodeWarrior:

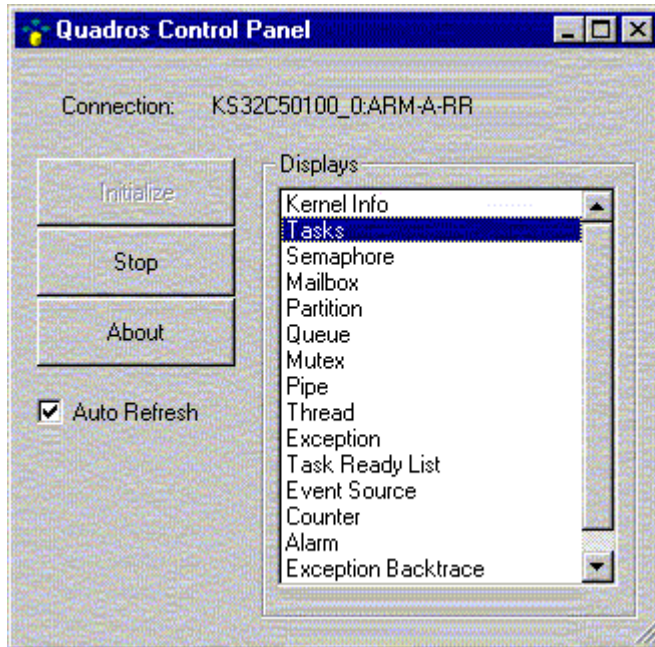
Select **Quadros** from the CodeWarrior's main menu bar. You will see that at least some of the items for launching displays are now enabled. Items that correspond to application elements which are not enabled for your application will remain disabled. An example menu follows:



While a debug session is in progress, and the application is at a breakpoint, you can launch these displays by opening the Quadros menu in the CodeWarrior IDE and selecting the desired display.

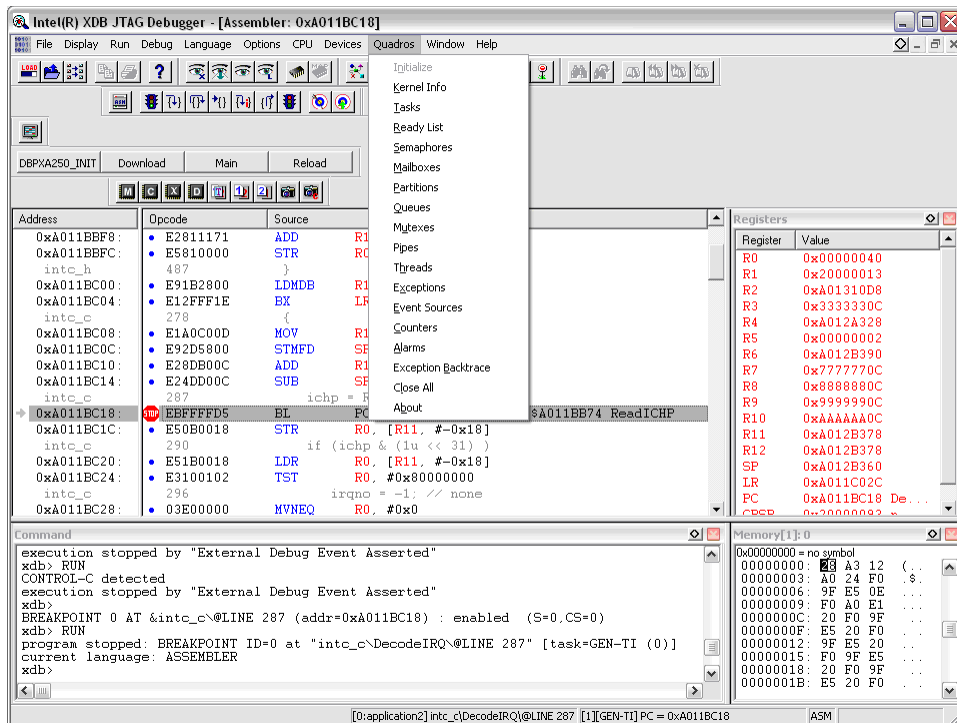
For RealView, Embedded Workbench, and μ Vision:

Bring the Quadros Control Panel to the foreground, then double-click the display of your choice.



For XDB:

Select **Quadros** from XDB's main menu bar. You will see that at least some of the items for launching displays are now enabled. Items that correspond to application elements which are not enabled for your application will remain disabled. An example menu follows:

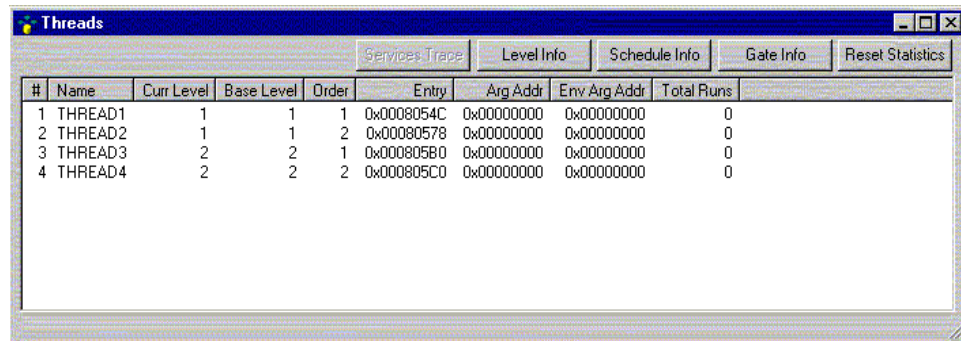


While a debug session is in progress, and the application is at a breakpoint, you can launch these displays by opening the Quadros menu in XDB and selecting the desired display.

The Displays

The displays described in the remaining sections are identical for all debugger environments.

The Threads Display



#	Name	Curr Level	Base Level	Order	Entry	Arg Addr	Env Arg Addr	Total Runs
1	THREAD1	1	1	1	0x0008054C	0x00000000	0x00000000	0
2	THREAD2	1	1	2	0x00080578	0x00000000	0x00000000	0
3	THREAD3	2	2	1	0x000805B0	0x00000000	0x00000000	0
4	THREAD4	2	2	2	0x000805C0	0x00000000	0x00000000	0

This display shows one line entry for every thread in the application.

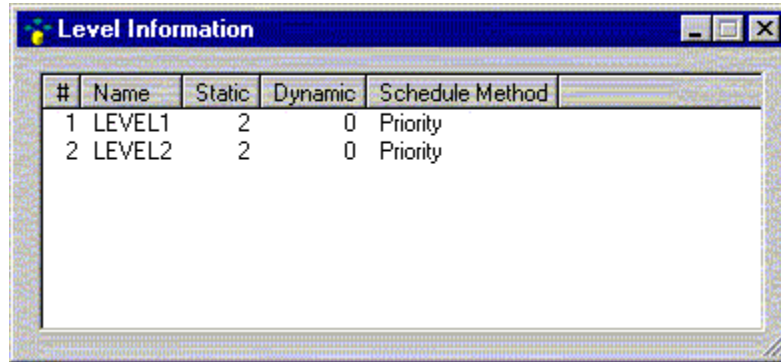
The information shown is as follows:

- The **# (pound sign)** column shows the object handle for the thread.
- The **Name** column shows the name of the thread.
- The **Curr Level** column shows the current level number for the thread.
- The **Base Level** column shows the base level number for the thread (corresponding to the level object to which the thread belongs).
- The **Order** column shows the thread's order within its level.
- The **Entry** column shows the thread's entry point address (in hexadecimal).
- If *Thread Arguments* are enabled for the application, the **Arg Addr** column shows the thread's argument address (in hexadecimal), otherwise this column does not appear.
- If *Thread Environment Arguments* are enabled for the application, the **Env Arg Addr** column shows the thread's environment argument address (in hexadecimal).
- If *Thread Statistics* are enabled for the application, the **Total Runs** column shows the number of times that the thread has run, otherwise the column does not appear.

A number of buttons are available to request additional actions. They are as follows:

- The **Level Info** button launches the **Level Information** display.
- The **Schedule Info** button launches the **Thread Schedule Information** display.
- If *Thread Gates* are enabled for the application, the **Gate Info** button launches the **Thread Gate Information** display, otherwise this button is disabled.
- If *Thread Statistics* are enabled for the application, the **Reset Statistics** button causes the **Total Runs** count for each thread to be reset, otherwise this button is disabled.

The Level Information Display



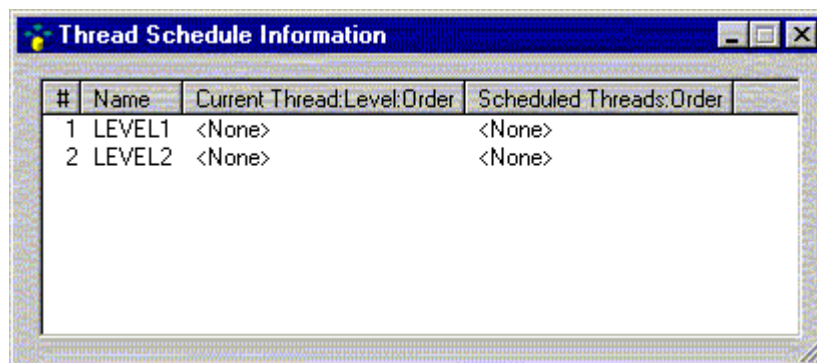
#	Name	Static	Dynamic	Schedule Method
1	LEVEL1	2	0	Priority
2	LEVEL2	2	0	Priority

This display shows one line entry for every level in the application.

The information shown is as follows:

- The # (**pound sign**) column shows the object handle for the level.
- The **Name** column shows the name of the level.
- The **Static** column shows the number of static threads belonging to the level.
- The **Dynamic** column shows the number of dynamic threads belonging to the level.
- The **Schedule Method** column shows the scheduling algorithm used by the level. It will either be "Priority" or "Round Robin".

The Thread Schedule Information Display



#	Name	Current Thread:Level:Order	Scheduled Threads:Order
1	LEVEL1	<None>	<None>
2	LEVEL2	<None>	<None>

This display shows one line entry for every level in the application.

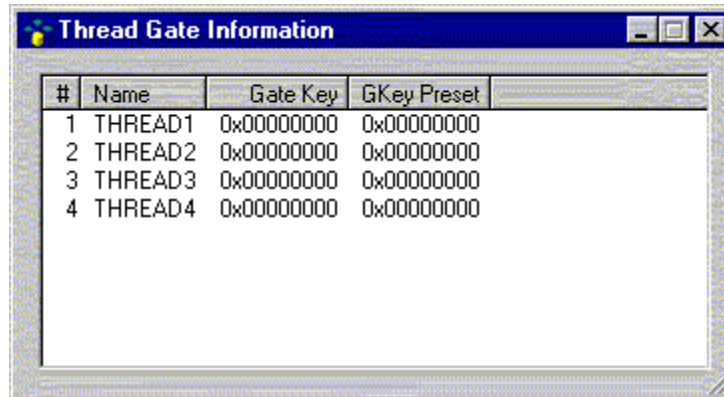
The information shown is as follows:

- The # (**pound sign**) column shows the object handle for the level.
- The **Name** column shows the name of the level.
- The **Current Thread:Level:Order** column shows the thread that is currently executing for the level, with its current level and order appended to the name

(for example, in the figure above, Thread4:L2:O1 indicates that Thread4 is running at Level 2, Order 1).

- The **Scheduled Threads:Order** column shows the threads which are currently scheduled for execution at that level, with their order numbers appended to the names.

The Thread Gate Information Display



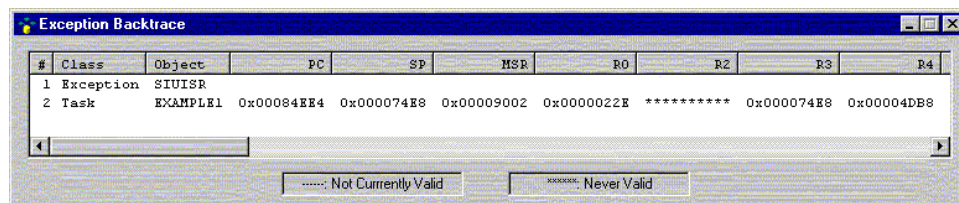
#	Name	Gate Key	GKey Preset
1	THREAD1	0x00000000	0x00000000
2	THREAD2	0x00000000	0x00000000
3	THREAD3	0x00000000	0x00000000
4	THREAD4	0x00000000	0x00000000

This display shows one line entry for every thread in the application.

The information shown is as follows:

- The # (**pound sign**) column shows the object handle for the thread.
- The **Name** column shows the name of the thread.
- The **Gate Key** column shows the thread's gate key.
- The **GKey Preset** column shows thread's gate key preset value.

The Exception Backtrace Display



#	Class	Object	PC	SP	MSR	R0	R2	R3	R4
1	Exception	SIUISR							
2	Task	EXAMPLE1	0x00084EE4	0x000074E8	0x00009002	0x0000022E	*****	0x000074E8	0x00004DB8

----- Not Currently Valid ***** Never Valid

This display shows a backtrace of the exception frames that are currently in the system. Each line represents a schedulable software component (Thread, Exception, Task or Kernel) that is either executing, or was preempted by the item above it. The topmost line shows the active component, which preempted the component listed on the second line, which in turn preempted the third, and so on.

The information shown is as follows:

- The # (**pound sign**) column shows the stack depth for the component, a value of 1 being most recent.

- The **Class** column shows the type of the component. The possible values are Thread, Exception, Task, and Kernel.
- The **Object** column shows the name of the component. Thread objects have their order and level numbers appended to them. For example, in the figure above, Thread1 is executing at Level 1 and Order 1.
- The remaining columns show, where possible, the saved register context for the component, in hexadecimal. The number, names, and types of registers are processor-specific. The pattern "-----" indicates that that register's context is not valid at the current time. The pattern "*****" indicates that the register's context is never valid in this configuration.

The Tasks Display

#	Name	Priority	Entry	Arg Address	State
1	EXAMPLE1	5	0x00080658	0x00000000	(Current)
2	CONITASK	3	0x000813DC	0x000048E8	Semaphore CONAISEM
3	CONOTASK	4	0x0008144C	0x000048E8	Semaphore CONAQSEM
4	<not in use>				

This display shows one line entry for every task in the application.

The information shown is as follows:

- The **# (pound sign)** column shows the object handle for the task.
- The **Name** column shows the name of the task.
- The **Priority** column shows the priority for the task. 1 is the highest priority, and 126 the lowest.
- The **Entry** column shows the task's entry point address (in hexadecimal).
- If *Task Environment Arguments* are enabled for the application, the **Arg Address** column shows the task's environment argument address (in hexadecimal), otherwise this column does not appear.
- The **State** column shows the task's current state. This can be Current, Ready, Suspended, Aborted, Inactive, or one of several possible waiting states:
 - Acknowledge (waiting for message acknowledgement)
 - Sleep
 - Semaphore
 - Queue (Queue Empty and/or Queue Full)
 - Mailbox (Message Received)
 - Partition
 - Mutex
 - Alarm

- Queue Semaphore (Queue Not Empty <QNE> or Queue Not Full <QNF>)
- Mailbox Semaphore (Mailbox Not Empty <MNE>)
- Partition Semaphore (Partition Not Empty <PNE>)
- Mutex Semaphore (mUtex Not Busy <UNB>)
- Alarm Semaphore (Alarm Expired <AE> or Alarm Aborted <AA>).

Note that a task may be Suspended, and also in a waiting state at the same time. Also, a task can be waiting on more than one semaphore. For the waiting states that may involve timeouts, the ticks remaining on the wait are shown as part of the state information. If a task is timesliced, the time remaining and period are shown in the form "<remaining/period ticks>", for example, "<1/5 ticks>".

A number of buttons are available to request additional actions. They are as follows:

- The **Register Info** button launches the **Task Registers Information** display.
- The **Stack Info** button launches the **Task Stacks Information** display.
- If *Task Semaphores* are enabled for the application, **Semaphores** button launches the **Task Semaphores Information** display.

The Task Register Information Display

#	Name	Status	PC	SP	MSR	R0	R2
0	<Null Task>		0x00084EE4	0x0007FF00	0x00009002	0x00000003	***** ----
1	EXAMPLE1	<current>					
2	CONITASK		0x00084EE4	0x000047E8	0x00009002	0x00000216	***** ----
3	CONOTASK		0x00084EE4	0x000043E8	0x00009002	0x00000216	***** ----
4	<not in use>						

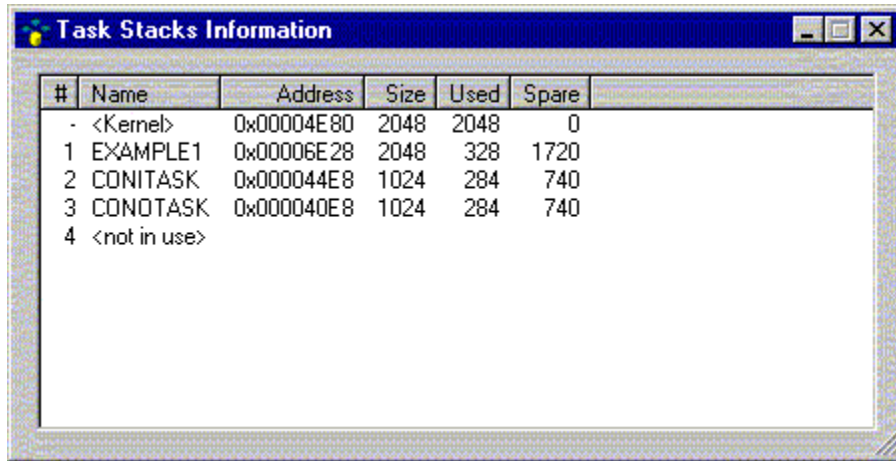
-----: Not Currently Valid *****: Never Valid

This display shows a list of the tasks currently in the application.

The information shown is as follows:

- The # (**pound sign**) column shows the object handle of the task.
- The **Name** column shows the task's name.
- The **Status** column shows the task's status. This can either be Current, Inactive, or if blank, indicates that the task is executing.
- The remaining columns show, where possible, the saved register context for the task. The number, names, and types of registers are processor-specific. The pattern "-----" indicates that that register's context is not valid at the current time. The pattern "*****" indicates that the register's context is never valid.

The Task Stacks Information Display



The image shows a window titled "Task Stacks Information" with a table containing the following data:

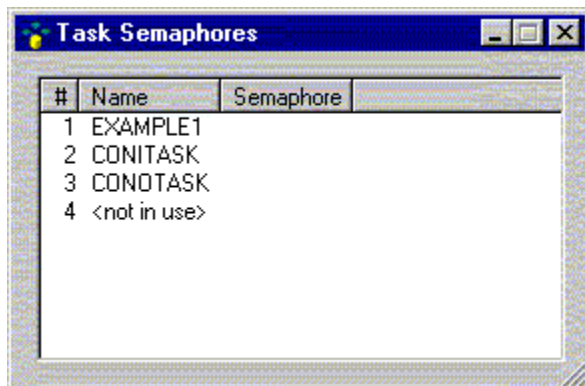
#	Name	Address	Size	Used	Spare
-	<Kernel>	0x00004E80	2048	2048	0
1	EXAMPLE1	0x00006E28	2048	328	1720
2	CONITASK	0x000044E8	1024	284	740
3	CONOTASK	0x000040E8	1024	284	740
4	<not in use>				

This display shows one line entry for every task in the application.

The information shown is as follows:

- The **# (pound sign)** column shows the object handle for the task.
- The **Name** column shows the name of the task.
- The **Address** column shows the base address of the task's stack (in hexadecimal).
- The **Size** column shows the amount of memory allocated for the stack.
- If *Task Statistics* are enabled for the application, the following two columns appear:
 - The **Used** column shows the high watermark of stack space used.
 - The **Spare** column shows the amount of stack space left over.

The Task Semaphores Display



The image shows a window titled "Task Semaphores" with a table containing the following data:

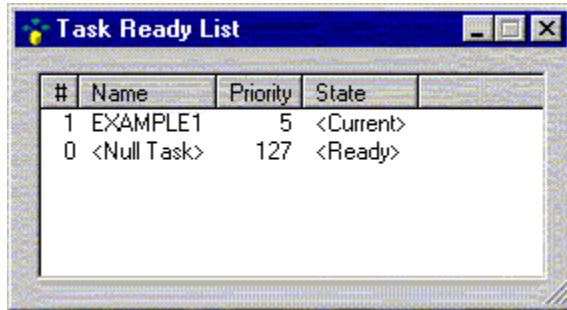
#	Name	Semaphore
1	EXAMPLE1	
2	CONITASK	
3	CONOTASK	
4	<not in use>	

This display shows one line entry for every task in the application.

The information shown is as follows:

- The # (**pound sign**) column shows the object handle for the task.
- The **Name** column shows the name of the task.
- The **Semaphore** column shows the name of the task's abort semaphore.

The Task Ready List Display



#	Name	Priority	State
1	EXAMPLE1	5	<Current>
0	<Null Task>	127	<Ready>

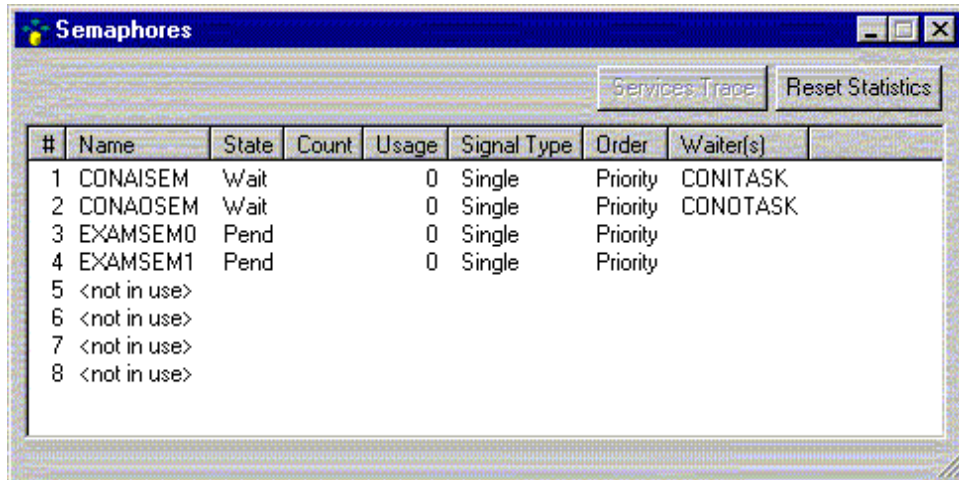
This display shows one line entry for every ready task in the application. A ready task is one that is either currently executing, or ready to execute.

The list is sorted in priority order, and represents the order in which the tasks will execute.

The information shown is as follows:

- The # (**pound sign**) column shows the object handle for the task.
- The **Name** column shows the name of the task.
- The **Priority** column shows the priority of the task, 1 being the highest priority and 127 the lowest.
- The **State** column shows the task's state. The first task listed will always be Current, and the others will always be Ready.

The Semaphores Display



The screenshot shows a window titled "Semaphores" with a table of semaphore statistics. The table has columns for #, Name, State, Count, Usage, Signal Type, Order, and Waiter(s). There are buttons for "Services Trace" and "Reset Statistics" at the top right.

#	Name	State	Count	Usage	Signal Type	Order	Waiter(s)
1	CONAISEM	Wait		0	Single	Priority	CONITASK
2	CONAQSEM	Wait		0	Single	Priority	CONOTASK
3	EXAMSEM0	Pend		0	Single	Priority	
4	EXAMSEM1	Pend		0	Single	Priority	
5	<not in use>						
6	<not in use>						
7	<not in use>						
8	<not in use>						

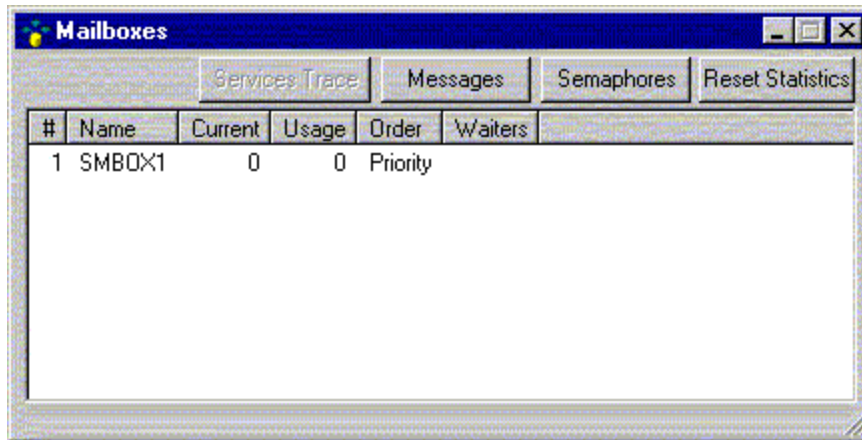
This display shows one line entry for every semaphore in the system.

The information shown is as follows:

- The **# (pound sign)** column shows the object handle for the semaphore.
- The **Name** column shows the semaphore's name.
- The **State** column shows the semaphore's state. This can either be Wait, Pend, or Done.
- The **Count** column shows either the semaphore's count if the state is not Pend.
- If *Semaphore Statistics* are enabled for the application, the **Usage** column shows the semaphore's usage count, otherwise this column does not appear.
- The **Signal Type** column shows whether it is a Single or Multiple signal type semaphore.
- The **Order** column shows the semaphore's waiting order type, which can be either Priority or FIFO.
- The **Waiter(s)** column shows the tasks that are waiting on the semaphore, if any.

If *Semaphore Statistics* are enabled for the application, the **Reset Statistics** button may be used to reset the contents of the **Usage** column, otherwise the button is disabled.

The **Mailboxes** Display



This display shows one line entry for each mailbox in the application.

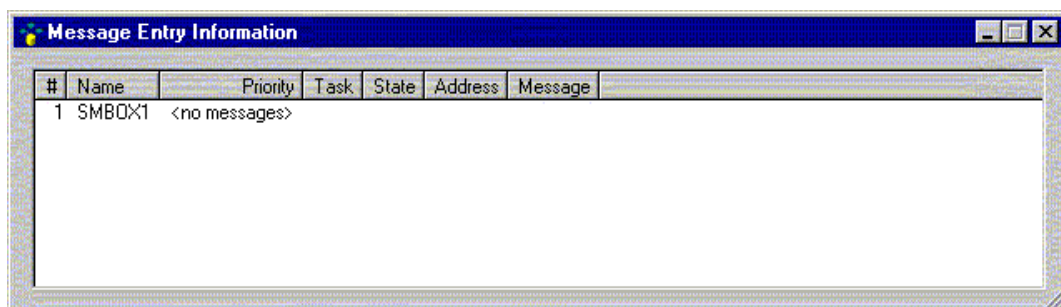
The information shown is as follows:

- The **# (pound sign)** column shows the mailbox's object handle.
- The **Name** column shows the mailbox's name.
- The **Current** column shows the current number of messages in the mailbox.
- If *Mailbox Statistics* are enabled for the application, the **Usage** column shows the total number of messages that have been placed in the mailbox, otherwise this column does not appear.
- The **Order** column shows the mailbox's waiting order type, which can be either Priority or FIFO.
- The **Waiter(s)** column shows the tasks that are waiting on the mailbox, if any.

A number of buttons are available to request additional actions. They are as follows:

- The **Messages** button launches the **Message Entry Information** display.
- If *Mailbox Semaphores* are enabled for the application, the **Semaphores** button launches the **Mailbox Semaphores** display, otherwise the button is disabled.
- If *Mailbox Statistics* are enabled for the application, the **Reset Statistics** button may be used to reset the contents of the **Usage** column, otherwise the button is disabled.

The Message Entry Information Display

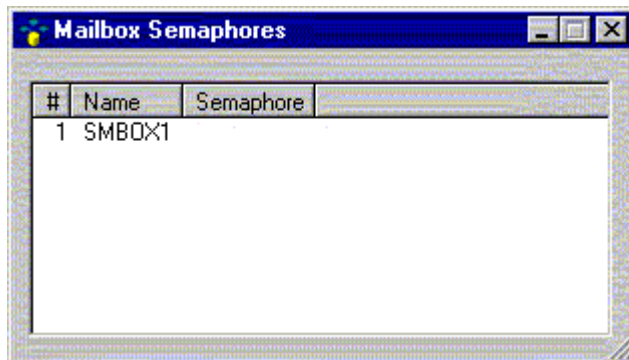


For each mailbox in the application, this display shows one line entry for each message in that mailbox.

The information shown is as follows:

- The **# (pound sign)** column shows the mailbox's object handle.
- The **Name** column shows the mailbox's name.
- The **Priority** column shows the priority of the message (Normal or Urgent).
- The **Task** column shows which task is to be sent an acknowledgement, if any.
- The **State** column shows the state of the message.
- The **Address** column shows the memory address of the message data buffer (in hexadecimal).
- The **Message** column shows the contents, in hexadecimal, of the first few memory locations in the message data buffer.

The Mailbox Semaphores Display

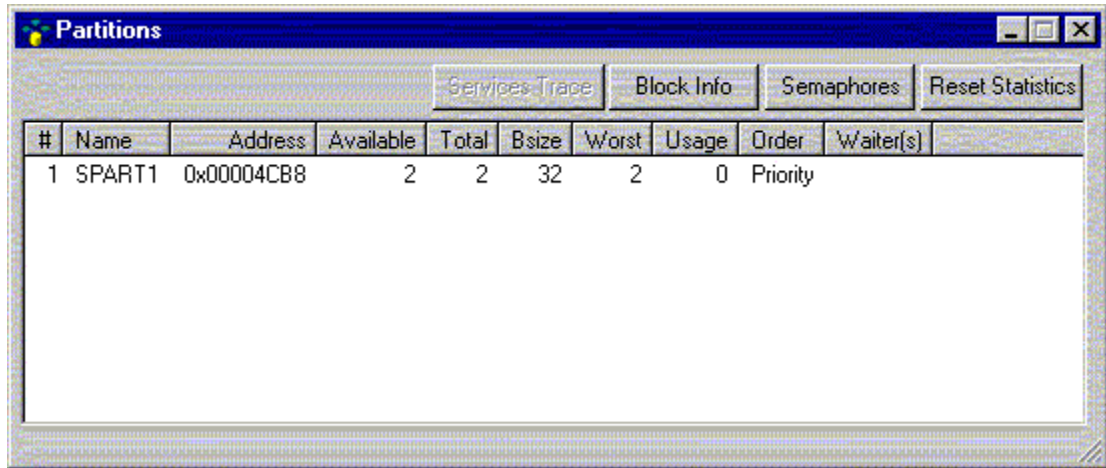


This mailbox shows one line entry for each mailbox in the application.

The information shown is as follows:

- The **# (pound sign)** column shows the object handle of the mailbox.
- The **Name** column shows the mailbox's name.
- The **Semaphore** column shows the name of the Mailbox-Not-Empty semaphore associated with the mailbox, if any.

The Partitions Display



The screenshot shows a window titled 'Partitions' with a table of partition information. The table has columns: #, Name, Address, Available, Total, Bsize, Worst, Usage, Order, and Waiter(s). There is one row of data for partition SPART1.

#	Name	Address	Available	Total	Bsize	Worst	Usage	Order	Waiter(s)
1	SPART1	0x00004CB8	2	2	32	2	0	Priority	

Buttons at the top: Services Trace, Block Info, Semaphores, Reset Statistics.

This display shows one line entry for each partition in the application.

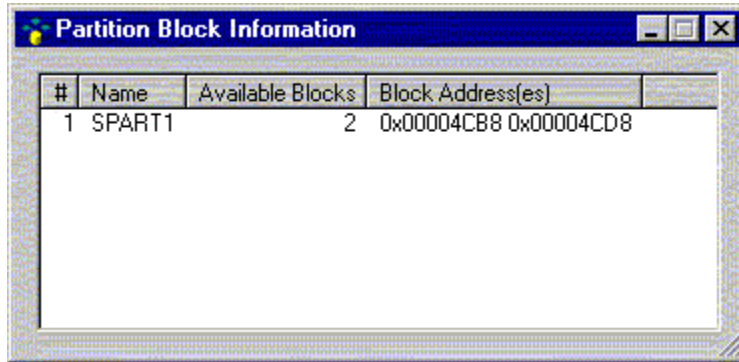
The information shown is as follows:

- The **# (pound sign)** column shows the object handle of the partition.
- The **Name** column shows the partition's name.
- The **Address** column shows the base memory address of the partition (in hexadecimal).
- The **Available** column shows the current number of available blocks in the partition.
- The **Total** column shows the total number of blocks in the partition.
- The **Bsize** column shows the size of each block in the partition.
- If *Partition Statistics* are enabled for the application, the following columns appear:
 - The **Worst** column shows the low watermark for available blocks.
 - The **Usage** column shows the usage count for the partition (i.e., the number of KS_FreeBlk() API calls to date).
- The **Order** column shows the waiting order for the partition. This is either Priority or FIFO.
- The **Waiter(s)** column shows the tasks that are waiting for the partition to become not-empty, or for the partition's not-empty semaphore, if any. If timeouts are involved, the ticks remaining are shown.

A number of buttons are available to request additional actions. They are as follows:

- The **Block Info** button launches the **Partition Block Information** display.
- If *Partition Semaphores* are enabled for the application, the **Semaphores** button can be used to launch the **Partition Semaphores** display, otherwise it is disabled.
- If *Partition Statistics* are enabled for the application, the **Reset Statistics** button can be used to reset the contents of the **Worst** and **Usage** columns, otherwise it is disabled.

The Partition Block Information Display



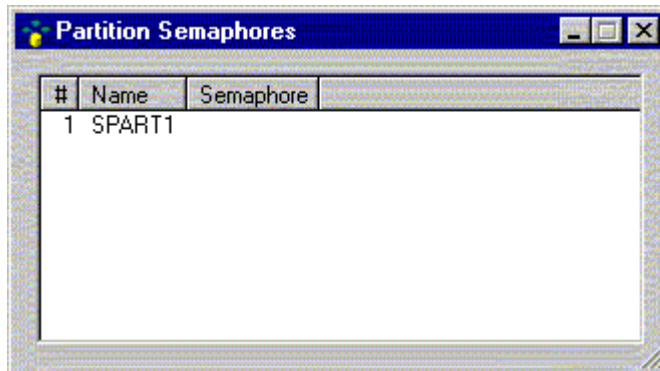
#	Name	Available Blocks	Block Address(es)
1	SPART1	2	0x00004CB8 0x00004CD8

This display shows one line for each partition in the application.

The information shown is as follows:

- The # (**pound sign**) column shows the object handle of the partition.
- The **Name** column shows the partition's name.
- The **Available Blocks** column shows the current number of available blocks in the partition.
- The **Block Address(es)** column shows the hexadecimal memory address of blocks in the partition. A maximum of 16 block addresses will be displayed.

The Partition Semaphores Display



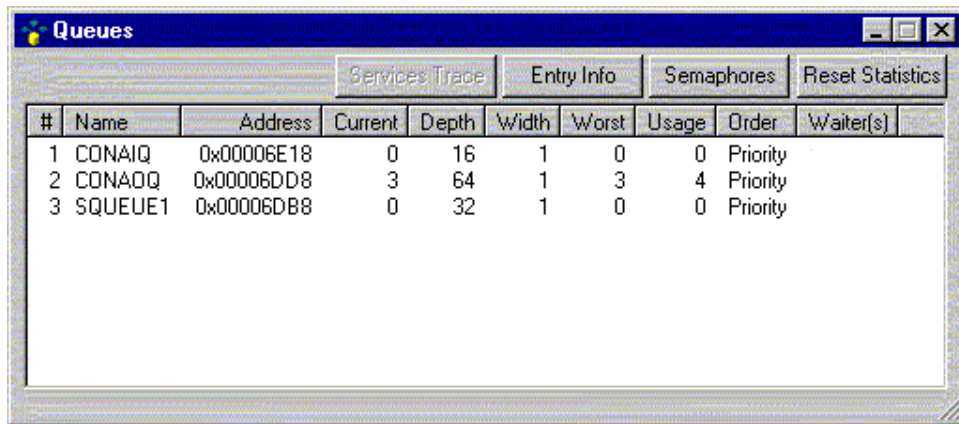
#	Name	Semaphore
1	SPART1	

This display shows one line for each partition in the application.

The information shown is as follows:

- The # (**pound sign**) column shows the object handle of the partition.
- The **Name** column shows the partition's name.
- The **Semaphore** column shows the name of the Partition-Not-Empty (<PNE>) semaphore associated with the partition, if any.

The Queues Display



The screenshot shows a window titled 'Queues' with a blue title bar. Below the title bar are four buttons: 'Services Trace', 'Entry Info', 'Semaphores', and 'Reset Statistics'. The main area contains a table with the following data:

#	Name	Address	Current	Depth	Width	Worst	Usage	Order	Waiter(s)
1	CONAIQ	0x00006E18	0	16	1	0	0	Priority	
2	CONAQ	0x00006DD8	3	64	1	3	4	Priority	
3	SQUEUE1	0x00006DB8	0	32	1	0	0	Priority	

This display shows one line entry for each queue in the application.

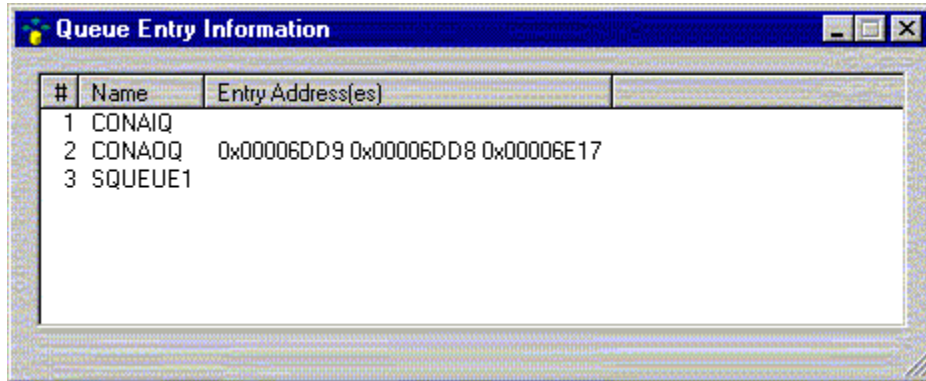
The information shown is as follows:

- The **# (pound sign)** column shows the object handle of the queue.
- The **Name** column shows the queue's name.
- The **Address** column shows the base memory address of the queue (in hexadecimal).
- The **Current** column shows the current number of entries in the queue.
- The **Depth** column shows the maximum number of entries in the queue.
- The **Width** column shows the size of each entry in the queue.
- If *Queue Statistics* are enabled for the application, the following two columns appear:
 - The **Worst** column shows the maximum number of entries that have been in the queue.
 - The **Usage** column shows total number of accesses to the queue to date.
- The **Order** column shows the waiting order for the queue. This is either Priority or FIFO.
- The **Waiters** column lists the tasks that are waiting on the queue, if any.

A number of buttons are available to request additional actions. They are as follows:

- The **Entry Info** button launches the **Queue Entry Information** display.
- If *Queue Semaphores* are enabled for the application, then the **Semaphores** button launches the **Queue Semaphores** display, otherwise it is disabled.
- If *Queue Statistics* are enabled for the application, then the **Reset Statistics** button can be used to reset the contents of the **Worst** and **Usage** columns, otherwise it is disabled.

The Queue Entry Information Display




#	Name	Entry Address(es)
1	CONAIQ	
2	CONAQ	0x00006DD9 0x00006DD8 0x00006E17
3	SQUEUE1	

This display shows one line for each queue in the application.

The information shown is as follows:

- The # (**pound sign**) column shows the object handle of the queue.
- The **Name** column shows the queue's name.
- The **Entry Address(es)** column shows the addresses (in hexadecimal) of entries currently in the queue. A maximum of 16 addresses will be displayed.

The Queue Semaphores Display



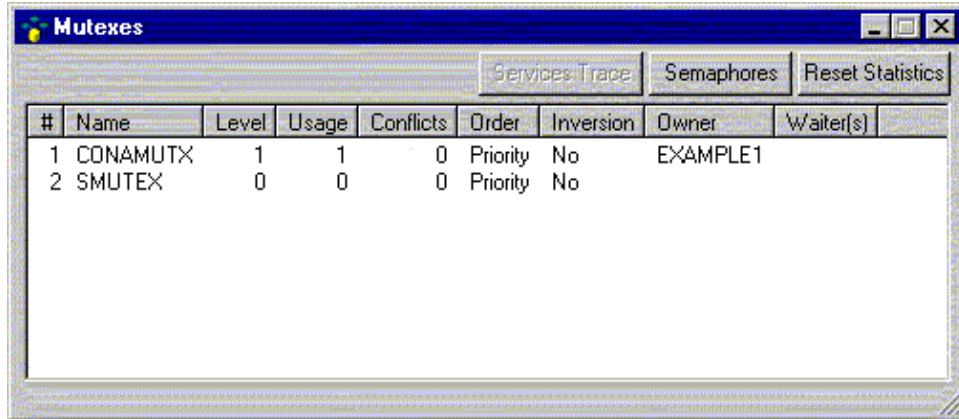
#	Name	Semaphore(s)
1	CONAIQ	
2	CONAQ	
3	SQUEUE1	

This display shows one line entry for each queue in the application.

The information show is as follows:

- The # (**pound sign**) column shows the object handle of the queue.
- The **Name** column shows the queue's name.
- The **Semaphore(s)** column shows names of the Queue-Not-Full (<QNF>) and Queue-Not-Empty (<QNE>) semaphores for the queue, if any.

The **Mutexes** Display



#	Name	Level	Usage	Conflicts	Order	Inversion	Owner	Waiter(s)
1	CONAMUTX	1	1	0	Priority	No	EXAMPLE1	
2	SMUTEX	0	0	0	Priority	No		

This display shows one line entry for each mutex in the application.

The information shown is as follows:

- The **# (pound sign)** column shows the object handle of the mutex.
- The **Name** column shows the mutex's name.
- If *Mutex Statistics* are enabled for the application, the following two columns appear:
 - The **Usage** column shows the number of releases performed.
 - The **Conflicts** column shows number of mutex contentions that have occurred.
- The **Order** column shows the waiting order for the mutex. This is either Priority or FIFO.
- The **Inversion** column shows whether or not priority inversion is enabled for the mutex.
- The **Waiters** column lists the tasks that are waiting on the mutex, if any.

A number of buttons are available to request additional actions. They are as follows:

- If *Mutex Semaphores* are enabled for the application, then the **Semaphores** button launches the **Mutex Semaphores** display, otherwise it is disabled.
- If *Mutex Statistics* are enabled for the application, then the **Reset Statistics** button resets the contents of the **Usage** and **Conflict** columns, otherwise it is disabled.

The **Mutex Semaphores** Display

#	Name	Semaphore
1	CONAMUTX	
2	SMUTEX	

This display shows one line entry for each mutex in the application.

The information show is as follows:

- The # (**pound sign**) column shows the object handle of the mutex.
- The **Name** column shows the mutex's name.
- The **Semaphore** column shows name of the mUtex-Not-Busy (<UNB>) semaphore, if any.

The Pipes Display

#	Name	Full	Empty	NBuFs	Bsize	Worst	Usage	PutFull Action	PutEmpty Action
1	SPIPE1	0	1	2	32	0	0	<no action>	<no action>
2	SPIPE2	0	2	2	64	0	0	<no action>	<no action>

This display shows one line entry for each pipe in the application.

The information shown is as follows:

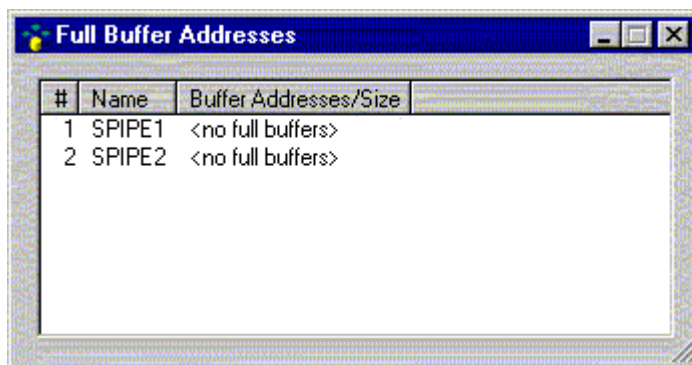
- The # (**pound sign**) column shows the object handle of the pipe.
- The **Name** column shows the pipe's name.
- The **Full** column shows the current number of full buffers.
- The **Empty** column shows the current number of empty buffers.
- The **NBuFs** column shows maximum number of buffers.
- The **Bsize** column shows the size of each buffer.
- If *Pipe Statistics* are enabled for the application, the following two columns appear:

- The **Worst** column shows maximum full buffer count.
- The **Usage** column shows the total number of accesses.
- The **PutFull Action** column shows the action to be taken (and the object to operate on) when a full buffer is put in the pipe.
- The **PutEmpty Action** column shows the action to be taken (and the object to operate on) when an empty buffer is put in the pipe.

A number of buttons are available to request additional actions. They are as follows:

- The **Full Buffers** button launches **Full Buffer Addresses** display.
- The **Empty Buffers** button launches **Empty Buffer Addresses** display.
- If *Pipe Statistics* are enabled for the application, then the **Reset Statistics** button can be used to reset the contents of the **Worst** and **Usage** columns, otherwise it is disabled.

The **Full Buffer Addresses** Display



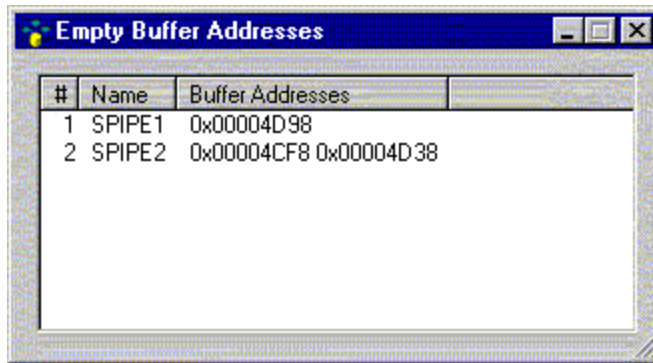
#	Name	Buffer Addresses/Size
1	SPIPE1	<no full buffers>
2	SPIPE2	<no full buffers>

This display shows one line entry for each pipe in the application.

The information shown is as follows:

- The **# (pound sign)** column shows the object handle of the pipe.
- The **Name** column shows the pipe's name.
- The **Buffer Addresses/Sizes** column shows the addresses and sizes of the pipe's full buffers. A maximum of 16 buffer addresses and sizes will be displayed.

The **Empty Buffer Addresses** Display

A screenshot of a Windows-style window titled "Empty Buffer Addresses". It contains a table with three columns: "#", "Name", and "Buffer Addresses". There are two rows of data.

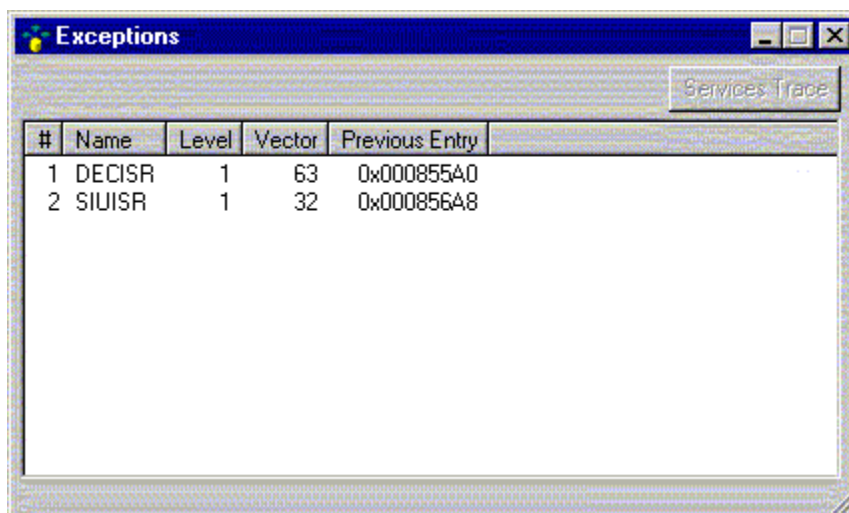
#	Name	Buffer Addresses
1	SPIPE1	0x00004D98
2	SPIPE2	0x00004CF8 0x00004D38

This display shows one line entry for each pipe in the application.

The information shown is as follows:

- The # (**pound sign**) column shows the object handle of the pipe.
- The **Name** column shows the pipe's name.
- The **Buffer Addresses** column shows the addresses of the pipe's empty buffers. A maximum of 16 buffer addresses will be displayed.

The Exceptions Display

A screenshot of a Windows-style window titled "Exceptions". It contains a table with five columns: "#", "Name", "Level", "Vector", and "Previous Entry". There are two rows of data. A "Services Trace" button is visible in the top right corner.

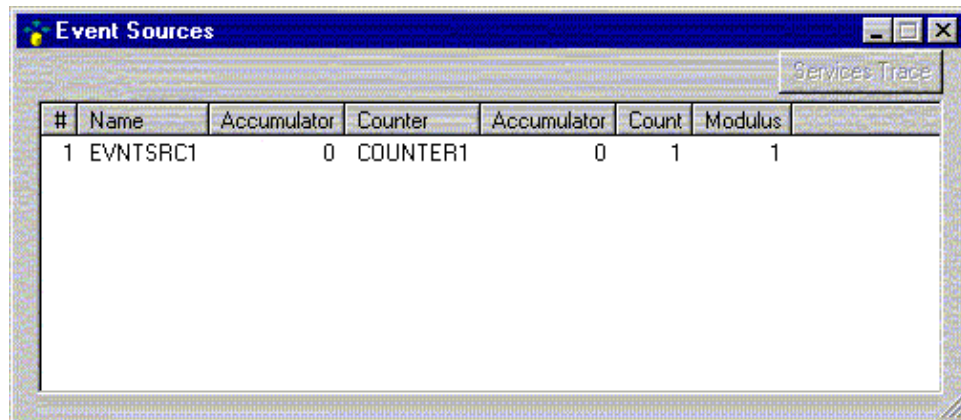
#	Name	Level	Vector	Previous Entry
1	DECISR	1	63	0x000855A0
2	SIUIR	1	32	0x000856A8

This display shows one line entry for each exception in the application.

The information shown is as follows:

- The # (**pound sign**) column shows the object handle of the exception.
- The **Name** column shows the exception's name.
- The **Level** column shows the interrupt level.
- The **Vector** column shows the vector number.
- The **Previous Entry** column shows the saved entry point address.

The Event Sources Display



The screenshot shows a window titled "Event Sources" with a "Services Trace" button in the top right. It contains a table with the following data:

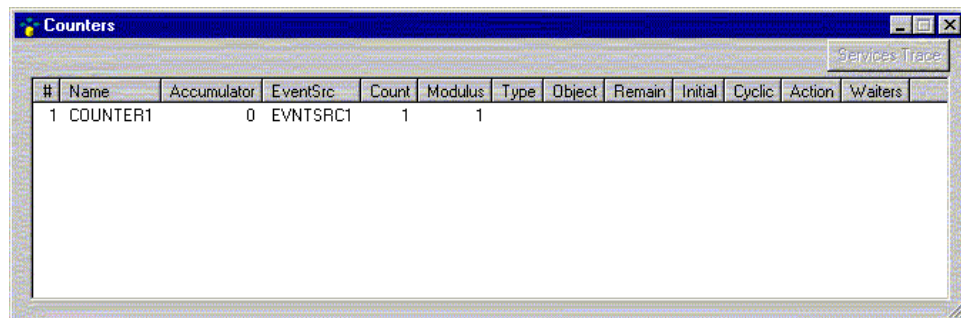
#	Name	Accumulator	Counter	Accumulator	Count	Modulus
1	EVNTSRC1	0	COUNTER1	0	1	1

This display shows one line entry for each counter in the application.

The information shown is as follows:

- The # (**pound sign**) column shows the object handle of the event source.
- The **Name** column shows the event source's name.
- The leftmost **Accumulator** column shows the event source's accumulator.
- The **Counter** column shows the name of the counter.
- The rightmost **Accumulator** column shows the counter's accumulator.
- The **Count** column shows the counter's count.
- The **Modulus** column shows the counter's modulus.

The Counters Display



The screenshot shows a window titled "Counters" with a "Services Trace" button in the top right. It contains a table with the following data:

#	Name	Accumulator	EventSrc	Count	Modulus	Type	Object	Remain	Initial	Cyclic	Action	Waters
1	COUNTER1	0	EVNTSRC1	1	1							

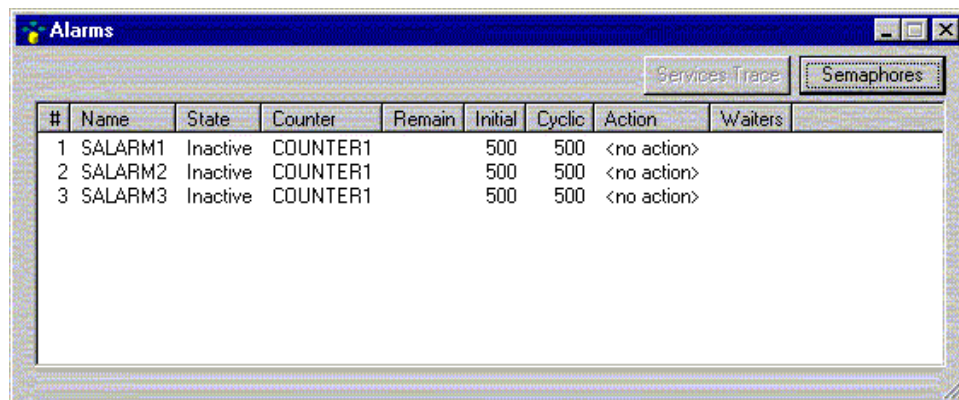
This display shows the counters in the application, and their associated event sources and alarms.

The information shown is as follows:

- The # (**pound sign**) column shows the object handle of the counter.

- The **Name** column shows the counter's name.
- The **Accumulator** column shows the counter's accumulator.
- The **EventSrc** column shows the event source with which the counter is associated.
- The **Count** column shows the counter's count value.
- The **Modulus** column shows the counter's modulus.
- The **Type** column shows the type of object associated with the alarm.
- The **Object** column shows the name of the object associated with the alarm.
- The **Remain** column shows the alarm expiration in ticks.
- The **Initial** column shows the initial period in ticks.
- The **Cyclic** column shows the cyclic period in ticks.
- The **Action** column shows the alarm action.
- The **Waiters** column shows the tasks that are waiting on the alarm, if any.

The Alarms Display



#	Name	State	Counter	Remain	Initial	Cyclic	Action	Waiters
1	SALARM1	Inactive	COUNTER1		500	500	<no action>	
2	SALARM2	Inactive	COUNTER1		500	500	<no action>	
3	SALARM3	Inactive	COUNTER1		500	500	<no action>	

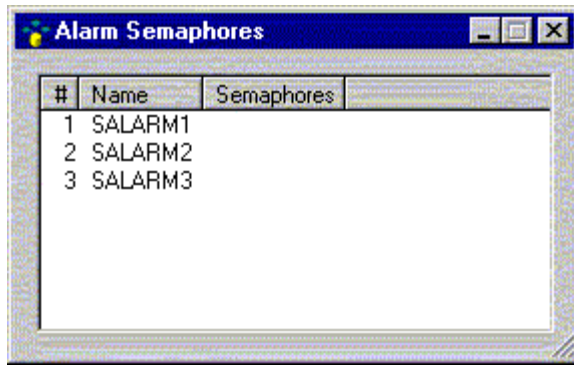
This display shows the alarms in the application.

The information shown is as follows:

- The **# (pound sign)** column shows the object handle of the alarm.
- The **Name** column shows the alarm's name.
- The **State** column shows the alarm's state (Active or Inactive).
- The **Counter** column shows the counter with which the alarm is associated.
- The **Remain** column shows the expiration in ticks.
- The **Initial** column shows the initial period in ticks.
- The **Cyclic** column shows the cyclic period in ticks.
- The **Action** column shows the alarm action upon expiration.
- The **Waiters** column shows the tasks that are waiting on the alarm, if any.

If *Alarm Semaphores* is enabled for the application, the **Semaphores** button launches the **Alarm Semaphores** display.

The Alarm Semaphores Display

A screenshot of a Windows-style window titled "Alarm Semaphores". It contains a table with three columns: "#", "Name", and "Semaphores". The table lists three entries: 1 SALARM1, 2 SALARM2, and 3 SALARM3.

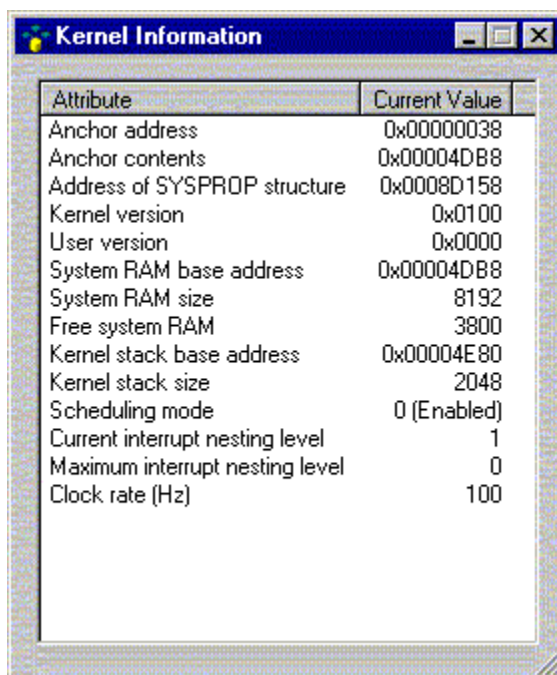
#	Name	Semaphores
1	SALARM1	
2	SALARM2	
3	SALARM3	

This display shows the alarms in the application.

The information shown is as follows:

- The # (**pound sign**) column shows the object handle of the alarm.
- The **Name** column shows the alarm's name.
- The **Semaphores** column shows the alarm's Alarm-Expired semaphore ("**<AE>**") and/or its Alarm-Aborted semaphore ("**<AA>**").

The **Kernel Information** Display

A screenshot of a Windows-style window titled "Kernel Information". It contains a table with two columns: "Attribute" and "Current Value". The table lists various system attributes and their current values.

Attribute	Current Value
Anchor address	0x00000038
Anchor contents	0x00004DB8
Address of SYSPROP structure	0x0008D158
Kernel version	0x0100
User version	0x0000
System RAM base address	0x00004DB8
System RAM size	8192
Free system RAM	3800
Kernel stack base address	0x00004E80
Kernel stack size	2048
Scheduling mode	0 (Enabled)
Current interrupt nesting level	1
Maximum interrupt nesting level	0
Clock rate (Hz)	100

This display shows general information about the kernel.

The information shown is as follows:

- **Anchor address** is the address which contains the address of the kernel workspace
- **Anchor contents** is the address of the kernel workspace
- **Address of SYSPROP structure** is the address of the system properties data structure
- **Kernel version** is a sixteen-bit quantity defining the version number of the RTXQ Quadros kernel. The high-order byte is the main version number. The next four bits specify the minor version number. The low-order four bits are the binding number. For example, a value of 0x0101 represents major version number 1, minor version number 0, binding number 1.
- **User version** is the user-defined version number, as specified in RTXQgen
- **System RAM base address** is the lowest address in system RAM. This should always match the **Anchor contents**.
- **Free system RAM** is the amount of unused system RAM
- If the kernel is not SS-only, the following information is displayed:
 - **Kernel stack base address** shows the lowest address in the kernel stack. This is regardless of whether the stacks on the target processor grow toward larger or smaller addresses.
 - **Kernel stack size** shows the size of the kernel stack
 - **Scheduling mode** shows whether the task scheduler is currently enabled or disabled
- **Current interrupt nesting level** shows the current number of nested interrupts/exceptions
- If *Exception Statistics* is enabled, **Maximum interrupt nesting level** is displayed, showing the deepest nesting level of interrupts/exceptions that has occurred thus far