RadiSys ARTIC960 Programmer's Reference

RadiSys Corporation
5445 NE Dawson Creek Drive
Hillsboro, OR   97124
(503) 615-1100
FAX: (503) 615-1150
www.radisys.com

November 2000

RadiSys Corporation
5445 NE Dawson Creek Drive
Hillsboro, OR 97124

November 2000

Contents

Figures

Tables

About This Book

This book describes specific aspects of programming in the RadiSys ARTIC960 coprocessor environments.

This book contains information about the ARTIC960 services available for writing adapter-resident programs. It also contains a brief description of the system unit utility programs and the steps required to compile and link both system unit and adapter programs.

This book does not include sample code.

Guide Contents

The following lists the contents of this Guide.

Chapter		Description
1	Loading and Configuring	Explains how to load and configure the kernel and related subsystems and RadiSys ARTIC960 Support for OS/2, AIX, and Windows NT.
2	ARTIC960 Kernel Services	Provides a summary of RadiSys ARTIC960 kernel services and ARTIC960 parameter types.
3	Base Kernel Services	Describes the base kernel services
4	Kernel Commands	Lists and describes the kernel commands.
5	Adapter Library Routines	Lists ANSI C library calls and describes the Miscellaneous Service, the System Bus Interface Services, and the PCI Services
6	System Unit Utilities	Describes the available system unit utilities.
7	System Unit APIs	Describes the system unit APIs.

The appendices provide additional information about ARTIC960.

Appendix		Description
A	Structure Definition	Lists the RIC_CONFIG, RIC_VERDATA, and RIC_EXCEPT structures.
B	Message File	Explains the error messages and the actions to be taken
C	Return, Error, and Exit Codes	Lists and explains the return codes.

Notational Conventions

This manual uses the following conventions:

- The term *ARTIC960* always refers to the RadiSys ARTIC960 products.
- The term *ARTIC960* can refer to programs that run on the ARTIC960, ARTIC960 PCI, ARTIC960Rx PCI, or ARTIC960Hx PCI adapters, or the adapters themselves.
- The term *ARTIC960 PCI* refers to functions supported only on the ARTIC960 PCI adapter; *ARTIC960 MCA* refers to functions supported only on the ARTIC960 Micro Channel adapter.
- The term *ARTIC960Rx PCI* refers to functions supported by the ARTIC960Rx PCI adapter; *ARTIC960Hx PCI* refers to functions supported by the ARTIC960Hx PCI adapter.
- The term *ARTIC960RxD PCI* refers to functions supported by the ARTIC960RxD PCI adapter.
- The term *OS/2* always refers to the IBM OS/2 operating system.
- The term *AIX* always refers to the IBM AIX operating system.
- The term *system bus* can refer to either the Micro Channel or PCI bus.
- All counts in this book are assumed to start at zero.
- All numeric parameters and command line options are assumed to be decimal values, unless stated otherwise.
- To pass a hexadecimal value for any numeric parameter, the parameter should be prefixed by **0x** or **0X**. Thus, the numeric parameters **16**, **0x10**, and **0X10** are all equivalent.
- Utilities all accept the **?** switch as a request for help with command syntax.
- All representations of bytes, words, and double words are in the little endian format.
- All bit numbering conforms to the industry standard of the most significant bit having the highest bit number. Bit 0 is the low-order bit.
- If a bit is set to 1, the associated description is true unless stated otherwise.
- `Screen text and syntax strings appear in this font.`

Notes indicate important information about the product.

Tips indicate alternate techniques or procedures that you can use to save time or better understand the product.

The globe indicates a World Wide Web address.

Cautions indicate situations that may result in damage to data or the hardware.

ESD cautions indicate situations that may cause damage to hardware via electrostatic discharge.

Warnings indicate situations that may result in physical harm to you or the hardware.

Where to Get More Information

You can find out more about RadiSys ARTIC960 from these sources:

- **World Wide Web**: RadiSys maintains an active site on the World Wide Web. The site contains current information about the company and locations of sales offices, new and existing products, contacts for sales, service, and technical support information. You can also send E-mail to RadiSys using the web site.

  When sending E-mail for technical support, please include information about both the hardware and software, plus a detailed description of the problem, including how to reproduce it.

  To access the RadiSys web site, enter this URL in your web browser:
  `http://www.radisys.com`

  Requests for sales, service, and technical support information receive prompt response.

- **Other**: If you purchased your RadiSys product from a third-party vendor, you can contact that vendor for service and support.

Reference Publications

You may need to use one or more of the following publications for reference:

- *RadiSys ARTIC960 Programmer's Guide*
- *RadiSys ARTIC960 STREAMS Environment Reference*
- Operating and Installation documentation provided with your computer system
- *Guide to Operations* books for one of the following coprocessor adapters:

  RadiSys ARTIC960 PCI adapter
  RadiSys ARTIC960Rx PCI adapter
  RadiSys ARTIC960Hx PCI adapter
  RadiSys ARTIC960RxD PCI adapter

  Each book contains a description of the coprocessor adapter, instructions for physically installing the adapter, parts listings, and warranty information.

- IBM Publications:
  - *IBM Operating System/2 (OS/2) Version 3.0, Advanced Interactive Executive (AIX) Version 4.1 and 4.2*
  - *IBM AIX Version 4.x Kernel Extensions and Device Support, Programming Concepts, (SC23-2207)*

    For information about writing a STREAMS module or driver, refer to the AIX Web site:

    `http://www.rs6000.ibm.com/doc_link/en_US/a_doc.lib/aixprogd/progcomc/str_prgref.htm`

    AIX supports a subset of SVR4.2 STREAMS calls, and the on-card STREAMS subsystem supports a subset of AIX STREAMS.
  - *IBM Personal System/2 Hardware Reference, S85F-1678)*
  - *IBM XL C Language Reference, (SC09-1260)*
- Intel Publications:
  - *i960 RP Microprocessor User's Manual*
  - *i960 Rx I/O Microprocessor Developer's Manual*
  - *i960 Hx Microprocessor User's Manual*
  - *i960 Cx Microprocessor User's Manual*
  - *80960CA User's Manual*

Developer's Assistance Program

Programming and hardware development assistance is provided by the RadiSys ARTIC Developer's Assistance Program (DAP). The DAP provides, via phone and electronic communications, on-going technical support—such as sample programs, debug assistance, and access to the latest microcode upgrades.

You can get more information or activate your *free* membership in the RadiSys ARTIC DAP by contacting us.

By telephone, call (561) 981-3200.

By E-mail, send to artic@radisys.com.

1 Loading and Configuring

This chapter contains information about loading and configuring:

- The kernel and related subsystems
- The ARTIC960 Support for OS/2
- The ARTIC960 Support for AIX
- The ARTIC960 Support for Windows NT

Supported Adapters

Table 1-1 shows which adapters are supported by each operating system.

Table 1-1. Adapters Supported by Each Operating System

Adapter	OS/2 Version 1.2.2	AIX Version 1.4.1	Windows NT Version 1.2.0
ARTIC960 Micro Channel	√	√	
ARTIC960 PCI	√	√	√
ARTIC960Rx PCI	√	√	√
ARTIC960Hx PCI	√	√	√
ARTIC960RxD PCI		√	
ARTIC960Rx Frame Relay PCI	√		√

Kernel and Subsystems

The kernel and related subsystems (collectively called *system executables*) must be loaded onto the adapter before any application processes are loaded. The list of system executables and associated file names are:

ric_kern.rel Runtime kernel. This module provides all of the services described in *Chapter 3: Base Kernel Services* on page 21.

ric_kdev.rel This module can be used instead of ric_kern.rel during the debug phase of application development.

ric_base.rel Base device driver. This module provides memory protection services.

If ric_base.rel is loaded when memory protection is not active, it is unloaded automatically by the kernel.

ric_mcio.rel System Bus I/O subsystem. This module provides basic support for moving data between adapters and the system unit.

ric_scb.rel This module provides peer-to-peer transport services using the Subsystem Control Block (SCB) architecture.

ric_oss.rel On-card STREAMS subsystem (OSS). This module provides a STREAMS environment on the adapter.

ric_ess.rel On-card STREAMS Cross Bus Subsystem. This module transmits STREAMS data across the system unit bus between STREAMS Access Library (SAL) and the On-card STREAMS Subsystem (OSS).

ric_pci.rel PCI bus configuration driver. This module provides basic services for configuring devices attached to the adapter's local PCI bus.

Specific applications may not require all modules.

The system executables must be loaded in the preceding order using the Application Loader utility. For information, see *Application Loader (ricload) Utility* on page 196.

Kernel Performance Considerations

Kernel performance can be affected by the way the adapter is loaded and configured.

Instruction Cache

The following support provides options that enable the kernel to pin critical kernel code in instruction cache:

- ARTIC960 Support for IBM OS/2, Version 1.2.1
- ARTIC960 Support for IBM AIX, Version 1.2 or higher
- ARTIC960 Support for Microsoft Windows NT, Version 1.0

There are two types of critical kernel code.

- Code critical for process-intensive applications (dispatcher, request/release semaphore, and so forth)
- Code critical for interrupt intensive applications (such as, first level interrupt handlers and enter/exit critical section)

The amount of kernel code that can be pinned depends on the size of the instruction cache which varies by processor type:

- The Cx processor is used on ARTIC960 and ARTIC960 PCI cards. On a Cx or Rx processor, the kernel pins 2 KB of the 4 KB instruction cache. On a Cx or Rx processor, only one type of critical code can be pinned.
- On an Hx processor, the kernel pins up to 8 KB of the 16 KB instruction cache. On an Hx processor, the cache is big enough to allow both process-intensive and interrupt-intensive critical code to be pinned.

The type of critical code to be pinned is controlled by the `PIN_KERN_PROC_CODE` and `PIN_KERN_INT_CODE` kernel configuration parameters. See page 5 for information about these parameters.

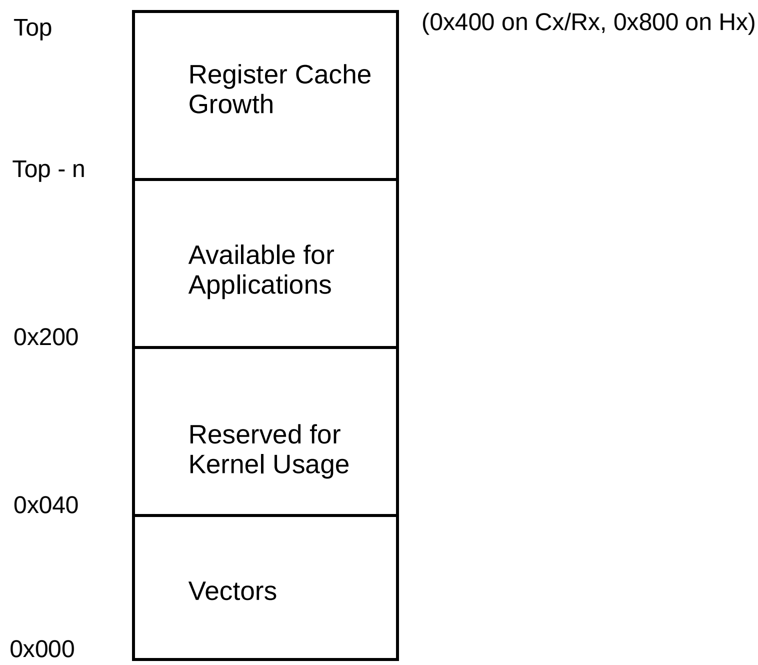
Internal Data RAM

The following provide for use of i960 internal data RAM.

- ARTIC960 Support for OS/2 (Version 1.2.1)
- ARTIC960 Support for AIX (Version 1.2)
- ARTIC960 Support for Windows NT (Version 1.0)

Internal data RAM is used for key kernel data and is also available for application use. The size of the internal data RAM is 1 KB for a Cx/Rx processor and 2 KB for an Hx processor.

Internal data RAM is used in the following manner:



The value of n is determined by the number of cached register sets. This is controlled by the `REG_CACHE` kernel parameter. The default for this parameter is 7. Values of 5 or less require no additional internal data RAM (n = 0). Values from 6 to 15 for `REG_CACHE` cause 64 bytes of internal data RAM to be used for each stack frame (*n*=(`REG_CACHE`–5)*64). On the ARTIC960Rx PCI card, internal data RAM is not used for register cache growth (*n* = 0).

Applications can use the range of internal data RAM from 200 to the top–*n*. However, the kernel does not manage this data area. To avoid potential conflicts, only applications that take over the card (that is, do not share the card with other applications) can make use of the application internal data RAM area.

It is not guaranteed that the compatibility of this function will be maintained across future releases.

Run Time Versus Development Kernel

There are two versions of the kernel:

- ric_kern.rel (runtime)
- ric_kdev.rel (development)

These versions are supported by the following ARTIC960 programs.

- ARTIC960 Support for OS/2, Version 1.2.1
- ARTIC960 Support for AIX, Version 1.2 or higher
- ARTIC960 Support for Windows NT, Version 1.0

Either version of the kernel can be loaded onto the adapter by way of the ricload utility.

The runtime version has limited error checking and no memory protection support. Validity checking of most input parameters has been eliminated from kernel service calls. Once an application has been debugged, this version can be used to give better performance.

The development version contains full support. The additional functions it provides are normally needed only during application debug.

Configuration Parameters

Configuration for the kernel and related subsystems is done through load-time parameters that can be passed on the command line or through a configuration file when using RICLOAD. These parameters take the form of keywords (representing specific parameters) followed by an equal sign (=) and their value. The individual parameters are separated by spaces, tabs, or new lines. Parameters not specified at load time take on default values. The configuration parameters for the kernel, the SCB subsystem, and the System Bus I/O subsystem follow. There are no parameters for the base device driver.

Kernel Parameters

The following are the kernel parameters that can be set. The default value for the parameter is underlined.

Table 1-2 (Sheet 1 of 2). Kernel Parameters

Parameter	Description
MEMORY_PROTECTION=<u>YES</u>|NO	Global memory protection enable. When YES, all normal processes run with memory protection on. This parameter is ignored when an application is running on an adapter that does not support memory protection.
DEFAULT_PRIORITY=40	Default process priority. Unless otherwise specified, when a process is loaded its priority is this value. It must be at least 32.
MAX_DD_SS=<u>16</u>	Maximum number of device drivers and subsystems.
MAX_REMOTE_MAILBOX=<u>16</u>	Maximum number of remote mailboxes.
MAX_PEER_ADAPTERS=<u>0</u>	Maximum number of peer units, not including this adapter or the system unit.

Table 1-2 (Sheet 2 of 2). Kernel Parameters

Parameter	Description
MAX_SYSTEM_MC_REQ=8	Maximum number of system bus read/write requests from the system unit outstanding.
DEFAULT_STACK_SIZE=4096	Default process stack size.
TIME_SLICE_INTERVAL=10	Time slice interval/disable. 0 means disable. Interval value is in milliseconds.
WATCHDOG_INTERVAL=2000	Watchdog interval/disable. 0 means disable. Interval value is in milliseconds. The watchdog timer is not supported on ARTIC960Rx PCI and ARTIC960Hx PCI cards. It will be ignored.
TIME_OF_DAY=YES|NO	Time-of-day clock enable.
PERFORMANCE_TIMER=YES|NO	Performance timer enable. If the performance timer is not enabled, the StartPerfTimer, StopPerfTimer, and ReadPerfTimer services return `RC_PERF_TIMER_NOT_ENABLED`. You can request the kernel to leave the time slice timer, watchdog timer, time-of-day timer, or performance timer available for a user process. See *Timer Notes* on page 5 for more information.
DATA_CACHE=YES|NO	Data cache enable. This parameter is ignored if data cache hardware is not present on the adapter or if `MEMORY_PROTECTION=YES`.
REG_CACHE=7	Number of register sets that are cached. Valid values depend on the type of processor in use.
INST_CACHE=YES|NO	Instruction cache enable.
PIN_KERN_PROC_CODE=YES|NO	When YES, kernel code that is critical for process-intensive applications is pinned in instruction cache, if instruction cache is enabled.
PIN_KERN_INT_CODE=YES|NO	When YES, kernel code that is critical for interrupt-intensive applications is pinned in instruction cache if instruction cache is enabled.
PEER_TIMEOUT=5	Timeout value used by the kernel mailbox subsystem when communicating with peer processes. Valid values are 0 to 60 seconds. A value of 0 means that the timeout will be disabled.

Timer Notes

For ARTIC960 and ARTIC960 PCI adapters, you can request the kernel to leave the timeslice timer, watchdog timer, time-of-day timer, or performance timer available for a user process. If `TIME_SLICE_INTERVAL`=0, `WATCHDOG_INTERVAL`=0, `TIME_OF_DAY`=`NO`, or `PERFORMANCE_TIMER`=`NO`, the kernel does not allocate the indicated timer. The timer can be allocated by a user process.

For ARTIC960Rx PCI and ARTIC960Hx PCI adapters, you can request the kernel to leave TIMER0 available for a user process. If `TIME_SLICE_INTERVAL`=0 and `PERFORMANCE_TIMER`=`NO`, the kernel will not allocate TIMER0. The timer can be allocated by a user process.

Subsystems Configuration

- Base Device Driver—There are no configuration parameters defined for the base subsystem.
- SCB Subsystem—The SCB Subsystem parameters that can be set are as follows. The default parameters are underlined.

Parameter	Description
MEMPROT = YES|<u>NO</u>	Subsystem memory protection enable. Protection is enabled only if kernel memory protection has been enabled.
SIGHANDPROT = YES|<u>NO</u>	Signal interrupt handler memory protection enable. Protection is enabled only if kernel memory protection and subsystem memory protection have been enabled.

- System Bus I/O Subsystem—The System Bus I/O Subsystem parameters that can be set are as follows. The default parameters are underlined:

Parameter	Description
THRESHOLD = <u>128</u>	Maximum number of bytes to be transferred using channel 1. Requests above this threshold value are sent on channel 2.
MEMPROT = YES|<u>NO</u>	Subsystem memory protection enable. Protection is enabled only if kernel memory protection has been enabled. If you are running the ARTIC960 Support for OS/2, Version 1.1.0 or higher, or the ARTIC960 Support for AIX, Version 1.1.3.0 or higher, this parameter is ignored. The System Bus I/O Subsystem always runs with its memory protection disabled.
TCINTPROT = YES|<u>NO</u>	Terminal count interrupt handler memory protection enable. Protection is enabled only if kernel memory protection and subsystem memory protection have been enabled. If you are running the ARTIC960 Support for OS/2, Version 1.1.0 or higher, or the ARTIC960 Support for AIX, Version 1.1.3.0 or higher, this parameter is ignored. The System Bus I/O Subsystem always runs with its memory protection disabled.
USERCHANNUM = 1|2	Channel number of the channel to be reserved for the user. It can be set to 1 or 2. The default is no channel is reserved for the user.

ARTIC960 Support for OS/2

The following sections describe the ARTIC960 Support for OS/2.

Supported ARTIC960 Configurations

The ARTIC960 adapter supports a wide variety of configurations such as interrupt levels, I/O addresses, and system bus memory configurations.

ARTIC960 32-bit Support for OS/2 supports all configurable adapter options with the following restrictions:

Interrupt level
All configurable interrupt levels are supported.

I/O address
All configurable base I/O addresses are supported.

8/16 KB memory window (ARTIC960 Micro Channel only)
This memory window is not used by the 32-bit OS/2 support. Its presence and location do not affect operation.

8 KB memory mapped (ARTIC960 PCI and ARTIC960Hx)
This memory window is not used by the 32-bit OS/2 support. Its presence and location do not affect operation.

Full memory window
Under OS/2, the system unit driver does not require or use direct access to the full memory window to communicate with an ARTIC960 adapter (except for ARTIC960Rx). However, the full memory window must be mapped onto the system bus to support peer-to-peer adapter operations. If the window is not visible on the system bus (either not physically mapped or not addressable due to slot constraints), peer-to-peer adapter operations are not supported.

Multiple Virtual DOS Machines (MVDM) DOS applications are not supported in ARTIC960 OS/2 Support.

Device Driver Installation

Two pieces of code provide OS/2 device driver support: the device driver and a detached process.

The ARTIC960 OS/2 device driver (*RICIO16.SYS*) is installed through *CONFIG.SYS*. It is a symmetric multiprocessing safe (SMP safe) device driver.

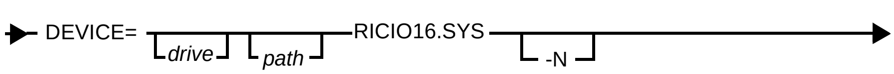


Figure 1-1. OS/2 Device Driver Syntax

This entry must be placed in the CONFIG.SYS file to call the ARTIC960 OS/2 device driver.

-N Disable interrupt nesting

Driver Messages

The content of the message file is listed in *Appendix B: Message File* on page 295. The following are the messages in that file used by the OS/2 driver.

Table 1-3. OS/2 Driver Messages

Message Number	Notes
RIC0001	(Invalid option)
RIC0002	(Invalid parameter)
RIC0009	Warning message (POST error)
RIC0010	Warning message (adapter failure)
RIC0016	(System error)
RIC0020	Information-only message (installing)
RIC0021	(Installed)
RIC0039	(No adapters)
RIC0049	(Unable to install interrupt handler)
RIC0064	Card ROM error (warning)
RIC0066	Interrupt nesting disabled (information)
RIC0071	Card ROM downlevel (warning)
RIC0081	Calibrating ARTIC960Rx timers (information)

Mailbox Process (RICMBX32.EXE)

The mailbox process, RICMBX32.EXE, is a detached process that works with the physical device driver to handle remote mailbox processing.

Mailbox Process Call

The mailbox process is called using the following syntax. It expects configuration parameters to be supplied to it through the command line or through a configuration file. The mailbox process must be loaded prior to any application process calls to mailbox.

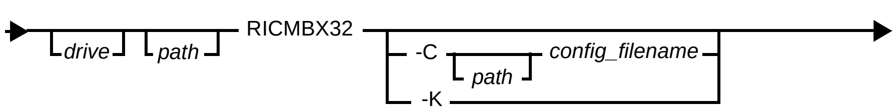


Figure 1-2. OS/2 Mailbox Process Syntax

-C *config_filename*
Specifies that the contents of the file *config_filename* should be used as input to the mailbox process for configuration parameters.

-K Specifies to stop the active mailbox process.

If the mailbox process is stopped, it may not be restarted without resetting and reloading the adapters.

The mailbox process requires certain initialization parameters. If you do not specify these parameters, they are assigned default values. The parameters take the form of keywords followed by an "=" sign and the value. Spaces, tabs, or new lines should separate individual parameters.

The following parameters can be set:

`MAX_GLOBAL_MAILBOX`
The maximum number of global mailboxes created in the system unit. The default is 16.

`MAX_REMOTE_MAILBOX`
The maximum number of remote mailboxes opened from the system unit. The default is 16.

`MAX_REMOTE_MAILBOX_OPEN`
The maximum number of remote mailbox open requests outstanding at any one time. The default is 16.

`MAX_REMOTE_MAILBOX_SEND`
The maximum number of remote mailbox send requests outstanding at any one time. The default is 32.

`MAX_REMOTE_MAILBOX_RCV`
The maximum number of remote mailbox receive requests outstanding at any one time. The default is 64.

`MAX_NUM_OF_UNITS`
The maximum number of SCB units. The default is 16.

`MBX_PROCESS_PRI_CLASS`
The priority class of the mailbox process, as listed below. The default is 4.

1 Idle
2 Regular
3 Time critical
4 Fixed-high

`MBX_PROCESS_PRI_DELTA`
The priority delta of the mailbox process. The priority delta parameter is a decimal value in the range –31 to +31. The default is 0.

`PEER_TIMEOUT`
Timeout value in seconds when communicating with peer processes. Valid values are 1 to 60. The default is 5.

For remote mailbox processing to occur, the Configuration utility must be used to establish communication between the system unit and adapters. For information on this utility, see *Configuration Utility* on page 207.

Mailbox Process Messages and Return Codes

The content of the message file is listed in *Appendix B: Message File* on page 295. The following table correlates the return code of the driver with the driver messages used by the OS/2 mailbox process.

Table 1-4. OS/2 Mailbox Process Messages

Message Number	Return Code	Notes
RIC0001	RC_UTIL_INVALID_CMDLINE_OPTION	
RIC0002	RC_UTIL_INVALID_CMDLINE_PARM	

Table 1-4. OS/2 Mailbox Process Messages

Message Number	Return Code	Notes
RIC0003	RC_UTIL_FILE_NOT_FOUND	
RIC0004	RC_UTIL_FILE_ACCESS	
RIC0006	RC_UTIL_NO_MORE_MEM	
RIC0016	RC_UTIL_SYSTEM_ERROR	
RIC0019	RC_UTIL_NOT_INSTALLED	
RIC0021	RC_UTIL_SUCCESS	Process successfully started
RIC0048	RC_UTIL_WRNHELP_GIVEN	
RIC0050	RC_UTIL_RESOURCE_BUSY	
RIC0051	RC_UTIL_ALREADY_STARTED	
RIC0062	RC_UTIL_SUCCESS	Process terminated successfully
RIC0063	RC_UTIL_NOT_RUNNING	Not found

ARTIC960 Support for AIX

The following sections describe the ARTIC960 Support for AIX.

Supported ARTIC960 Configurations

The ARTIC960 adapter supports a wide variety of configurations, such as interrupt levels, I/O addresses, and system bus memory configurations. ARTIC960 Support for AIX Version 1.3 added support for 14 ARTIC960 adapters (0 through 13).

The ARTIC960 Support for AIX supports all configurable hardware options with the following restrictions.

Interrupt Level
All configurable interrupt levels are supported.

I/O Address
All configurable base I/O addresses are supported. For the RadiSys ARTIC960 PCI and ARTIC960Hx, this window is used for peer-to-peer I/O memory operations only.

8/16-KB Memory Window (ARTIC960 Micro Channel only)
This window is used only during device driver configuration, and then it is disabled.

8-KB Memory Mapped I/O (ARTIC960 PCI and ARTIC960Hx only)
This window is used for system-unit-to-card I/O memory operations.

Full Memory Window
Under AIX, the system unit driver uses this window for small accesses to ARTIC960 memory.

DMA (Direct Memory Access) Peer-to-Peer Support
ARTIC960 Support for AIX Version 1.1.6 supports DMA between two peer adapters. In versions after 1.1.6, DMA between two peer adapters is supported only for Micro Channel adapters.

Micro Channel Only

Arbitration Levels

All configurable arbitration levels are supported.

The ARTIC960 adapters can have two arbitration levels defined. The ARTIC960 AIX support uses the first arbitration level for system-unit-to-adapter DMA transfers and the second arbitration level for peer-to-peer DMA transfers. The adapter attribute that controls the second arbitration level is DMA2Enable, and it can be changed using SMIT. When DMA2Enable is set to YES, a second arbitration level is reserved for peer-to-peer transfers.

Streaming Data Enable

Use SMIT to change this attribute.

Selected Feedback Return Enable

Use SMIT to change this attribute.

Parity Enable

Use SMIT to change this attribute.

Channel Check Enable

Use SMIT to change this attribute.

Device Driver Installation

Two pieces of code provide the AIX support: the device driver and a daemon process.

The ARTIC960 AIX device driver (ricio) is installed through the AIX Configuration Manager at system boot time. It is a multiprocessing safe (MP Safe) device driver.

Mailbox Process (ricmbx)

The mailbox process, ricmbx, is a daemon process that works in conjunction with the device driver to handle remote mailbox processing.

Version 1.3 of ricmbx added the support for the first ten ARTIC960 adapters, numbers 0 through 9. Mailboxes can be used locally on the adapters 10 and above, but the system unit mailboxes will not be able to communicate remotely.

Mailbox Process Call

Configuration parameters must be supplied on the command line or through a configuration file. The mailbox process must be loaded prior to any application process calls to mailbox services.

You can start the mailbox process at system boot time by adding a line to the /etc/inittab file.

The mailbox process is called using the following syntax. The superuser authority is required to start the mailbox process.

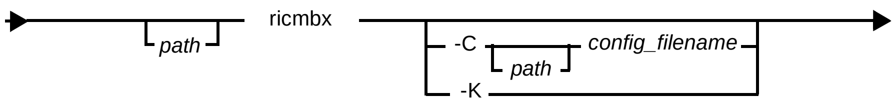


Figure 1-3. AIX Mailbox Process Syntax

-C *config_filename*
Specifies that the contents of the file *config_filename* should be used as input to the mailbox process for configuration parameters.

-K Kill the active mailbox process (superuser authority required).

The mailbox process requires certain initialization parameters. If you do not specify these parameters, they take default values. The parameters take the form of keywords followed by an = sign and their value. Spaces, tabs, or new lines should separate individual parameters.

The following parameters can be set.

`MAX_GLOBAL_MAILBOX`
The maximum number of global mailboxes created in the system unit. The default is 16.

`MAX_REMOTE_MAILBOX`
The maximum number of remote mailboxes opened from system unit. The default is 16.

`MAX_REMOTE_MAILBOX_OPEN`
The maximum number of remote mailbox open requests outstanding at any one time. The default is 16.

`MAX_REMOTE_MAILBOX_SEND`
The maximum number of remote mailbox send requests outstanding at any one time. The default is 32.

`MAX_REMOTE_MAILBOX_RCV`
The maximum number of remote mailbox receive requests outstanding at any one time. The default is 64.

`MAX_NUM_OF_UNITS`
The maximum number of SCB units. The default is 16.

`AIX_MBX_PROCESS_PRIORITY`
The mailbox process priority for the mailbox. Application processes wanting to use the mailbox services need to have their process priority a lesser priority than the mailbox process (1 is the highest priority level within AIX). The default is 16.

For remote mailbox processing to occur, the Configuration utility must be used to establish communication between the system unit and adapters. For information on this utility, see *Configuration Utility* on page 207.

Mailbox Process Messages and Return Codes

The content of the message file is listed in *Appendix B: Message File* on page 295. The following table correlates the return code of the driver with the driver messages used by the AIX mailbox process.

Table 1-5. AIX Mailbox Process Messages

Message Number	Return Code	Notes
RIC0001	RC_UTIL_INVALID_CMDLINE_OPTION	Invalid configuration file, invalid parameter names or values
RIC0002	RC_UTIL_INVALID_CMDLINE_PARM	
RIC0003	RC_UTIL_FILE_NOT_FOUND	Configuration file not found
RIC0004	RC_UTIL_FILE_ACCESS	Cannot access configuration file
RIC0006	RC_UTIL_NO_MORE_MEM	Parameter value exceeds system memory limit
RIC0016	RC_UTIL_SYSTEM_ERROR	OS or device driver error
RIC0019	RC_UTIL_NOT_INSTALLED	Device driver not installed
RIC0021	RC_UTIL_SUCCESS	Process successfully started
RIC0048	RC_UTIL_WRNHELP_GIVEN	
RIC0050	RC_UTIL_RESOURCE_BUSY	shmid,semid, used by mailbox has been allocated in the system
RIC0051	RC_UTIL_ALREADY_STARTED	
RIC0076	RC_UTIL_FILE_ACCESS	No root authority

Error Logging

The *error log* is a tool designed to help isolate hardware problems. The AIX Support Device Driver provides error logging.

The following ARTIC errors are logged to the system error log:

I/O Error
Problems reading or writing to the system bus address space.

ROM Error
The read only memory (ROM) boot strap microcode not responding in reasonable time during initialization or ROM finds a hardware error during its boot strap initialize or reset.

Watchdog Timer Interrupt
Hard exceptions reported by the ARTIC960 kernel or the adapter (ARTIC960 Watchdog Timeouts).

Adapter Kernel Exception
Software exceptions by the ROM or the kernel.

Trace Facility

The AIX Support device driver provides trace hooks to monitor the entry and exit of the driver routines and the interrupt routine. The trace event is `29F`.

ARTIC960 Support for Windows NT

The following sections describe the ARTIC960 Support for Windows NT.

Supported ARTIC960 Configurations

The ARTIC960 Support for Windows NT uses the hardware-abstraction layer (HAL) to configure the configurable hardware options such as interrupt level, I/O addresses, and system bus memory configurations.

Device Driver Installation

The Windows NT Version 4.0 device driver is installed when the ARTIC960 Support for Windows NT is installed. The driver is started at boot time. It is a symmetric multiprocessing (SMP) safe device driver.

Mailbox Process

The ARTIC960 Support for Windows NT supports card-to-card mailbox activity. However, the System Unit mailbox process is not supported.

Event Logging

Four types of events are logged to the Windows NT Event Log for any particular ARTIC960 device.

Configuration Errors
These errors are issued when the device driver has encountered errors with interfacing to the hardware-abstraction layer (HAL).

ROM Errors
The read only memory (ROM) bootstrap microcode is not responding in reasonable time during initialization, or ROM finds a hardware error during its bootstrap initialize or reset.

Watchdog Timer Interrupt
Hard exceptions reported by the ARTIC960 kernel or adapter.

Informational
Various messages indicating starting and stopping of a device or ARTIC960 kernel exceptions.

2 ARTIC960 Kernel Services

This chapter summarizes the ARTIC960 kernel services and parameter types.

Summary of Services

Table 2-1 lists the modes in which each kernel service can be called. The first column lists all the services in the same sequence as they appear in this book. The remaining columns define whether the service can be called from an interrupt handler, a signal handler, an asynchronous event notification handler, a process exit routine, and a critical section. Normal process time is one of the modes that is not in the table because all services can be called at normal process time. The other mode that is not in the table is device driver or subsystem call handlers. The rules that determine which services can be called are the same as the mode from which the device driver or subsystem was called. Each service is described in *Chapter 3: Base Kernel Services* on page 21.)

Table 2-1 (Sheet 1 of 4). ARTIC960 Kernel Services

Function	Interrupt Handler	Signal Handler	Async Handler	Process Exit	Critical Section
Process Management Services					
CompleteInit	No	No	No	No	No
QueryProcessStatus	Yes	Yes	Yes	Yes	Yes
QueryCardInfo	Yes	Yes	Yes	Yes	Yes
QueryConfigParams	Yes	Yes	Yes	Yes	Yes
CreateProcess	No	No	No	Yes	Yes
StartProcess	No	No	No	Yes[3]	Yes
StopProcess	No	No	No	Yes[3]	Yes
UnloadProcess	No	No	No	Yes[3]	Yes
SuspendProcess	Yes[6]	Yes[6]	Yes[6]	Yes	Yes[1]
ResumeProcess	Yes	Yes	Yes	Yes	Yes
SetExitRoutine	No	No	No	No	Yes
SetPriority	Yes[7]	Yes[7]	Yes[7]	Yes	Yes
QueryPriority	Yes[7]	Yes[7]	Yes[7]	Yes	Yes
QueryProcessInExec	Yes	Yes	Yes	Yes	Yes
SetProcessData	No	Yes[7]	Yes[7]	Yes	Yes
GetProcessData	Yes	Yes	Yes	Yes	Yes
EnterCritSec	Yes[5]	Yes	Yes	Yes	Yes
ExitCritSec	Yes[5]	Yes	Yes	Yes	Yes
Dispatch	No	No	No	Yes	Yes[1]

Table 2-1 (Sheet 2 of 4). ARTIC960 Kernel Services

Function	Interrupt Handler	Signal Handler	Async Handler	Process Exit	Critical Section
Process Synchronization Services					
CreateSem	No	No	No	Yes	Yes
OpenSem	No	No	No	Yes	Yes
CloseSem	No	No	No	Yes	Yes
ReleaseSem	Yes	Yes	Yes	Yes	Yes
RequestSem	No	No	No	Yes	Yes[1]
QuerySemCount	Yes	Yes	Yes	Yes	Yes
SetSemCount	Yes	Yes	Yes	Yes	Yes
CreateEvent	No	No	No	Yes	Yes
OpenEvent	No	No	No	Yes	Yes
CloseEvent	No	No	No	Yes	Yes
WaitEvent	No	No	No	Yes	Yes[1]
Memory Management Services					
CreateMem	No	No	No	Yes	Yes
OpenMem	No	No	No	Yes	Yes
CloseMem	No	No	No	Yes	Yes
ResizeMem	No	No	No	Yes	Yes
SetMemProt	No	No	No	Yes	Yes
SetProcMemProt	Yes	Yes	Yes	Yes	Yes
QueryMemProt	No	No	No	Yes	Yes
QueryProcMemProt	Yes	Yes	Yes	Yes	Yes
QueryFreeMem	Yes	Yes	Yes	Yes	Yes
InitSuballoc	No	No	No	Yes	Yes
GetSuballoc	Yes	Yes	Yes	Yes	Yes[1]
FreeSuballoc	Yes	Yes	Yes	Yes	Yes
GetSuballocSize	Yes	Yes	Yes	Yes	Yes
MallocMem	Yes	Yes	Yes	Yes	Yes
FreeMem	Yes	Yes	Yes	Yes	Yes
CollectMem	No	Yes	Yes	Yes	Yes
Timer Services					
CreateSwTimer	No	No	No	Yes	Yes
CloseSwTimer	No	No	No	Yes	Yes
StartSwTimer	Yes	Yes	Yes	Yes	Yes
StopSwTimer	Yes	Yes	Yes	Yes	Yes
SetSystemTime	Yes	Yes	Yes	Yes	Yes
QuerySystemTime	Yes	Yes	Yes	Yes	Yes
StartPerfTimer	Yes	Yes	Yes	Yes	Yes
StopPerfTimer	Yes	Yes	Yes	Yes	Yes
ReadPerfTimer	Yes	Yes	Yes	Yes	Yes

Table 2-1 (Sheet 3 of 4). ARTIC960 Kernel Services

Function	Interrupt Handler	Signal Handler	Async Handler	Process Exit	Critical Section
Process Communication Services					
CreateQueue	No	No	No	Yes	Yes
OpenQueue	No	No	No	Yes	Yes
CloseQueue	No	No	No	Yes	Yes
PutQueue	Yes	Yes	Yes	Yes	Yes
GetQueue	Yes[2]	Yes[2]	Yes[2]	Yes	Yes[1]
SearchQueue	Yes	Yes	Yes	Yes	Yes
CreateMbx	No	No	No	Yes	Yes
OpenMbx	No	No	No	Yes	Yes[4]
GetMbxBuffer	Yes	Yes	Yes	Yes	Yes
FreeMbxBuffer	Yes	Yes	Yes	Yes	Yes
SendMbx	Yes[8]	Yes[8]	Yes[8]	Yes	Yes[4]
ReceiveMbx	Yes[2]	Yes[2]	Yes[2]	Yes	Yes[1]
CloseMbx	No	No	No	Yes	Yes
CreateSig	No	No	No	Yes	Yes
OpenSig	No	No	No	Yes	Yes
CloseSig	No	No	No	Yes	Yes
InvokeSig	Yes	Yes	Yes	Yes	Yes
Device Driver/Subsystem Services					
CreateDev	No	No	No	Yes	Yes
OpenDev	No	No	No	Yes	Yes
CloseDev	No	No	No	Yes	Yes
InvokeDev	Yes	Yes	Yes	Yes	Yes
AllocVector	No	No	No	Yes	Yes
ReturnVector	No	No	No	Yes	Yes
SetVector	No	No	No	Yes	Yes
AllocHW	No	No	No	Yes	Yes
ReturnHW	No	No	No	Yes	Yes
QueryHW	No	No	No	Yes	Yes
AllocVectorMux	No	No	No	Yes	Yes
SetVectorMux	No	No	No	Yes	Yes
Asynchronous Event Notification Services					
RegisterAsyncHandler	No	No	No	Yes	Yes
DeregisterAsyncHandler	No	No	No	Yes	Yes
Hooks					
RegisterHook	No	No	No	Yes	Yes
DeregisterHook	No	No	No	Yes	Yes

Table 2-1 (Sheet 4 of 4). ARTIC960 Kernel Services

Function	Interrupt Handler	Signal Handler	Async Handler	Process Exit	Critical Section
Kernel Trace Services					
InitTrace	No	No	No	Yes	Yes
EnableTrace	Yes	Yes	Yes	Yes	Yes
DisableTrace	Yes	Yes	Yes	Yes	Yes
LogTrace	Yes	Yes	Yes	Yes	Yes

1 When the service is called with Preemption or Interrupts disabled, if the process blocks, interrupts and preemption are enabled.

2 May be called with timeout equal to 0.

3 When in an exit handler, a process cannot start, stop, or unload itself.

4 If the service is called for a remote mailbox, interrupts and preemption are enabled.

5 Preemption cannot be enabled/disabled.

6 A process may not suspend itself from an interrupt handler, a signal handler, or an asynchronous handler.

7 When in a handler, a process ID must be provided, that is, not the process currently in execution.

8 May be called to send a message to a local mailbox only.

Parameter Types

The description of each service includes the type of each parameter. The following types are defined:

Table 2-2 (Sheet 1 of 2). Parameter Types

Service	Description
RIC_DEVHANDLE	Device driver or subsystem resource handle
RIC_PROCESSID	Process ID
RIC_EVNHANDLE	Event resource handle
RIC_MBXHANDLE	Mailbox resource handle
RIC_QUEHANDLE	Queue resource handle
RIC_SEMHANDLE	Semaphore resource handle
RIC_SIGHANDLE	Signal resource handle
RIC_TMRHANDLE	Software timer resource handle
RIC_ASYNCHANDLER	Entry point code address for an asynchronous event handler
RIC_SIGHANDLER	Entry point code address for a signal
RIC_VECTOR	Code address
RIC_SLONG	Signed number
RIC_TIMEOUT	Signed number
RIC_ULONG	Unsigned number
RIC_USHORT	Unsigned number
RIC_RESPMBX	Unsigned number
RIC_INVOKENUM	Subsystem call function number
RIC_CARDNUM	Logical card number

Table 2-2 (Sheet 2 of 2). Parameter Types

Service	Description
RIC_DOHANDLER	Entry point of code address for an *OpenDev* entry point
RIC_DCHANDLER	Entry point of code address for a *CloseDev* entry point
RIC_DIHANDLER	Entry point of code address for an *InvokeDev* entry point
RIC_VECTOR_MUX	Code address

3 Base Kernel Services

The realtime multi-tasking kernel (which is downloaded to the adapter) provides the following base services.

Process Management Services

The following are the process management services.

Refer to the *ARTIC960 Programmer's Guide* for additional information.

CompleteInit—Mark Process as Completely Initialized

This service notifies the kernel that the calling process has completed initialization. This service can also indicate initialization errors.

Functional Prototype

```
RIC_ULONG CompleteInit (RIC_ULONG  ErrorCode,
                        RIC_ULONG  ProcessRev,
                        RIC_ULONG  OptionWord,
                        RIC_ULONG  Reserved);
```

Parameters

ErrorCode Input. Contains process-specific information stored in the process control block. If this field is 0, the process initialized successfully. If this field is greater than 0, the process found an error during initialization.

ProcessRev Input. Contains process-specific information stored in the process control block. Although no specific format is defined, the following format is recommended: ProcessRev is a 32-bit application-version number:

- 8-bit major version number (most significant byte)
- 8-bit minor version number
- 16-bit revision number (least significant two bytes)

OptionWord

Input. A bit field specifying options for the CompleteInit service.

`PERMANENT_PROCESS`
The process is defined as permanent and cannot be stopped or unloaded.

`TRANSIENT_PROCESS`
Specifies that the process can be stopped or unloaded.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                  RC_INVALID_CALL
RC_INVALID_RESERVED_PARM    RC_INVALID_OPTION
RC_ALREADY_INITIALIZED
```

Remarks

This service is used by all processes, device drivers, and subsystems. Device drivers and subsystems will not receive OPEN requests until this service has been called for successful initialization. This service is optional for normal processes. However, to use the –W option of the Application Loader utility, the process must use this service.

If the caller passes a non-zero value in ErrorCode, the process is stopped and does not regain control after the call. The ErrorCode is intended as a safety net for reporting status when no other method is available (for example, the process was not able to open a mailbox). If a process wants to report non-error initialization status, another communications mechanism should be used.

Although the ProcessRev format is not required, it is recommended that application programmers implement it because the diagnostic utility (RICSTAT) uses this field.

QueryProcessStatus—Get the Process Status

This service gets the status and other process-related information, accepting either a process name or process ID for input. When a process name is specified, this service resolves it to a process ID.

Functional Prototype

```
RIC_ULONG QueryProcessStatus (char                          *ProcessName,
                     RIC_PROCESSID                          ProcessID,
                     RIC_ULONG                              OptionWord,
                     struct RIC_ProcessStatusBlock         *PSBBufferPtr,
                     RIC_ULONG                              BufferSize,
                     RIC_ULONG                              Reserved);
```

Parameters

ProcessName

Input. Process name whose status is required.

ProcessID Input. Process ID whose status is required.

OptionWord

Input. Possible values are:

`PROCESS_NAME_OPTION`
Specifies the ProcessName parameter is used.

`PROCESS_ID_OPTION`
Specifies the ProcessID parameter is used.

PSBBufferPtr

Input. Process status and other process-related information is returned to the requesting process in this buffer.

BufferSize Input. Size of buffer pointed to by PSBBufferPtr. If the buffer is not large enough, the service copies BufferSize bytes into the user's buffer and returns an error code.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                     RC_INVALID_MEM_ACCESS
RC_INVALID_NAME                RC_INVALID_OPTION
RC_INVALID_RESERVED_PARM       RC_INVALID_PROCESSID
RC_NAME_NOT_FOUND              RC_BUFFER_TOO_SMALL
```

Remarks

The kernel returns the information to the calling process using the following structure.

```
struct RIC_ProcessStatusBlock
{
  RIC_PROCESSID  ProcessID;
  RIC_ULONG      ProcessState;
  RIC_ULONG      ProcessInfo;
  RIC_ULONG      ProcessType;
  RIC_ULONG      Priority;
  RIC_ULONG      MemProtState;
};
```

ProcessID The process ID.

ProcessState

Defines the current state of the process, using two sets of bits (see *Primary Process State Bits* and *Secondary Process State Bits* on page 27).

ProcessInfo Process-related information, which is passed to the kernel in the ProcessRev field on the CompleteInit system call.

ProcessType Returned by the kernel. It can be one of the following types:

```
PROCESS_TYPE_NORMAL
PROCESS_TYPE_DEVDRV
PROCESS_TYPE_SUBSYS
```

Priority Indicates the current execution priority for this process.

MemProtState

Defines the state of memory protection.

MEMPROT_ENABLE	Memory protection enabled
MEMPROT_DISABLE	Memory protection disabled

Primary Process State Bits

The primary process state bits are shown in the following table.

State bit	Description
LOADED	The `LOADED` bit is set while a process is being loaded and is reset when the loading operation is complete.
PROC_STOPPED	The `PROC_STOPPED` bit is set when a process has been loaded and is reset when it is unloaded by the system unit or another process.
STARTED	The `STARTED` bit is set when a process is started and is reset when it is stopped by the system unit or another process.
STOPPING	The `STOPPING` bit is set when the exit handler of a process is running.

Secondary Process State Bits

Processes that are in the *started* or *stopping* states have a valid secondary state, as defined in the following table.

State	Description
SUSPENDED	The `SUSPENDED` bit is set when the process has been suspended. The process is taken off the dispatch queue.
BLOCKED	The `BLOCKED` bit is set when the process has been blocked using a RequestSem, WaitEvent, GetQueue, or ReceiveMbx call. The process is taken off the dispatch queue.
DEVICE_DRIVER	The `DEVICE_DRIVER` bit is set if a process declares itself as a device driver.
QUEUED	The `QUEUED` bit is set when a process is ready to run.
WAITING_ON_PMREQ	The `WAITING_ON_PMREQ` bit is set when a process is blocked because it has issued a StopProcess or UnloadProcess call that is being serviced.

Process Information Bits

Active processes may have valid information bits:

INITIALIZED	The `INITIALIZED` bit is set when the process issues the CompleteInit system call.
PERMANENT	The `PERMANENT` bit is set when a process, subsystem, or device driver sets this field with the CompleteInit system call. The process, subsystem, or device driver cannot be unloaded by the UnloadProcess call when this bit is set. It may be unloaded from the system unit at any time.

QueryCardInfo—Get the Card Configuration Information

This service gets information from the read-only memory (ROM) data structure.

Functional Prototype

```
RIC_ULONG QueryCardInfo (struct RIC_CardInfo *ParmPtr,
                         RIC_ULONG            BufferSize,
                         RIC_ULONG            Reserved);
```

Parameters

ParmPtr Input. Pointer to the user's buffer. The card information is copied into this memory.

BufferSize Input. Size of the buffer is pointed to by ParmPtr. If the buffer is not large enough, the service copies BufferSize bytes into the user's buffer and returns an error code.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_MEM_ACCESS
RC_BUFFER_TOO_SMALL
RC_INVALID_RESERVED_PARM
```

Remarks

This is the card information structure. These values are taken from the ROS structure:

```
struct  RIC_CardInfo
{
  RIC_ULONG             PageSize;
  RIC_ULONG             KernelVersion;
  RIC_ULONG             BaseSubVersion;
  RIC_ULONG             MCIOSubVersion;
  RIC_ULONG             SCBSubVersion;
  RIC_CARDNUM           CardNum;
  RIC_ULONG             NumCards;
  RIC_ULONG             CardType;
  RIC_ULONG             Master1Version;
  RIC_ULONG             Master2Version;
  RIC_ULONG             Master3Version;
  RIC_ULONG             Reserved;
  RIC_ULONG             BaseCardVersion;
  RIC_ULONG             ROSVersion;
  RIC_ULONG             MemRegions
struct RIC_MemRegion    MemInfo[MAX_MEM_REGIONS];
}
```

```
struct  RIC_MemRegion
{
  RIC_ULONG       MemBase;
  RIC_ULONG       MemTotal;
  RIC_ULONG       MemType;
}
```

Parameters

PageSize Size of memory protection page

KernelVersion
Kernel version number

BaseSubVersion
Base subsystem version

MCIOVersion
System Bus I/O subsystem version

SCBVersion SCB subsystem version

CardNum Card number

NumCards Number of ARTIC960 cards in the configuration

CardType Type of adapter card. Provides information about the type of bus, the presence of data cache, and the type of interface chip. The following masks can be used to determine CardType information.

`RIC_CARD_TYPE`
Indicates the type of bus. Possible values are:

`RIC_MCA`	Micro Channel
`RIC_PCI`	PCI (Peripheral Component Interconnect)

`RIC_DCACHE`
Indicates the presence of a data cache. Possible values are:

0	Data cache hardware is not present.
1	Data cache hardware is present.

`RIC_IF_CHIP`
Indicates the type of interface chip. Possible values are:

`RIC_MIAMI`	Miami
`RIC_MP2P`	Miami PCI to PCI
`RIC_RP`	i960RP
`RIC_RxD`	i960RxD

Master1Version
Version of ARTIC 32-bit Memory Controller Chip

Master2Version
Version of system bus Interface Chip

Master3Version
Version of CFE Local Bus/AIB Interface Chip

Reserved Reserved field

BaseCardVersion
Base card version

ROSVersion
ROS version

MemRegions
Number of memory regions

MemBase Base address of memory region

MemTotal Size, in bytes, of memory region

MemType Type of memory region. Possible values are:

```
MEM_TYPE_INSTRUCTION
MEM_TYPE_PACKET
```

QueryConfigParams—Get the Configuration Parameters

This service gets the kernel parameters.

Functional Prototype

```
RIC_ULONG QueryConfigParams (struct RIC_ConfigParms *ParmPtr,
                             RIC_ULONG               BufferSize,
                             RIC_ULONG               Reserved);
```

Parameters

ParmPtr Input. Pointer to user's structure. The kernel parameters are copied into this memory.

BufferSize Input. Number of bytes to copy to the user's buffer.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_BUFFER_TOO_SMALL
RC_INVALID_MEM_ACCESS
RC_INVALID_RESERVED_PARM
```

Remarks

This is the configuration parameter structure. These values are set at load time. The user can set these with a configuration file. A default value is used for each parameter not set by the user.

```
struct  RIC_ConfigParms
{
  RIC_ULONG   MemoryProtection;
  RIC_ULONG   DefaultPriority;
  RIC_ULONG   MaxProcess;
  RIC_ULONG   MaxTimer;
  RIC_ULONG   MaxSemaphore;
  RIC_ULONG   MaxMemAlloc;
  RIC_ULONG   MaxQueue;
  RIC_ULONG   MaxEvent;
  RIC_ULONG   MaxDDSS;
  RIC_ULONG   MaxSignal;
  RIC_ULONG   MaxLocalMailbox;
  RIC_ULONG   MaxGlobalMailbox;
  RIC_ULONG   MaxRemoteMailbox;
  RIC_ULONG   MaxRemoteMailboxOpen;
  RIC_ULONG   MaxRemoteMailboxSend;
  RIC_ULONG   MaxRemoteMailboxRcv;
  RIC_ULONG   MaxPeerAdapters;
  RIC_ULONG   MaxSystemMCReq;
  RIC_ULONG   MaxAdapterMCReq;
  RIC_ULONG   DefaultStackSize;
  RIC_ULONG   TimeSlice;
  RIC_ULONG   WatchDog;
```

```
  RIC_ULONG   TimeOfDay;
  RIC_ULONG   PerfTimer;
  RIC_ULONG   DataCache;
  RIC_ULONG   InstCache;
  RIC_ULONG   RegCache;
  RIC_ULONG   PinKernProcCode;
  RIC_ULONG   PinKernIntcode;
  RIC_ULONG   PeerTimeout;
}
```

Parameter

MemoryProtection
Memory protection enable flag
0 Disabled
1 Enabled)

DefaultPriority
Default process priority

MaxProcess Maximum number of processes; includes device drivers and subsystems

MaxTimer Maximum number of timers

MaxSemaphore
Maximum number of semaphores

MaxMemAlloc
Maximum number of memory allocations

MaxQueue Maximum number of queues

MaxEvent Maximum number of events

MaxDDSS Maximum number of device drivers and subsystems; does not include kernel device drivers and kernel subsystems

MaxSignal Maximum number of signals

MaxLocalMailbox
Maximum number of local mailboxes

MaxGlobalMailbox
Maximum number of global mailboxes

MaxRemoteMailbox
Maximum number of remote mailboxes

MaxRemoteMailboxOpen
Maximum number of remote mailboxes open

MaxRemoteMailboxSend
Maximum number of remote mailbox sends outstanding

MaxRemoteMailboxRcv
Maximum number of remote mailbox receives outstanding

MaxPeerAdapters
Maximum number of peer adapters

MaxSystemMCReq
: Maximum number of system bus read/write requests from the system unit outstanding.

MaxAdapterMCReq
: Maximum number of system bus move requests outstanding

DefaultStackSize
: Default process stack size

TimeSlice Time slice interval/disable (interval value in milliseconds; 0 means disabled)

Watchdog Watchdog interval/disable (interval value in milliseconds; 0 means disabled)

TimeOfDay Time of day enable flag
- 0 Disabled
- 1 Enabled

PerfTimer Performance timer enable flag
- 0 Disabled
- 1 Enabled

DataCache Data cache enable flag
- 0 Disabled
- 1 Enabled

InstCache Instruction cache enable flag
- 0 Disabled
- 1 Enabled

RegCache Number of register sets that are cached. Valid values are 5 through 15.

PinKernProcCode
: Option to pin kernel code critical for process intensive applications
- 0 Disabled
- 1 Enabled

PinKernIntCode
: Option to pin kernel code critical for interrupt intensive applications
- 0 Disabled
- 1 Enabled

PeerTimeout
: Timeout value used by kernel mailbox subsystem when communicating with peer processes.

CreateProcess—Create a Process

This service creates a peer process.

Functional Prototype

```
RIC_ULONG CreateProcess (char           *ProcessName,
                         RIC_USERENTRY   EntryPoint,
                         RIC_ULONG       StackSize,
                         void           *ParamPtr,
                         RIC_ULONG       ParamSize,
                         RIC_ULONG       Priority,
                         RIC_ULONG       OptionWord,
                         RIC_PROCESSID  *ProcessID,
                         RIC_ULONG       Reserved);
```

Parameters

ProcessName
: Input. A process name to assign to the created process. The kernel's subsystems have process names beginning with "RIC_". User process names should not begin with this prefix.

EntryPoint Input. Address of the entry point of the created process.

StackSize Input. Size of stack to be allocated for the created process. If this parameter is 0, the kernel allocates the default size stack.

ParamPtr Input. Pointer to a parameter area passed to created process.

ParamSize Input. Size of parameter area.

Priority Input. The priority of the created process set by creating process. A 0 means use the default priority as specified in the kernel configuration parameter `DEFAULT_PRIORITY`.

OptionWord Input. A bit field of options for creating a process. The constants for the following options should be ORed together to build the appropriate set of options.

- Process start option

  `CREATE_AND_NO_START`
  Creates a peer process without issuing a start.

  `CREATE_AND_START`
  Starts the process after it is created. This is the default.

- Stack cache option

  `CREATE_CACHE_STACK`
  By default, the stack is not cached. To designate the stack as cacheable, can be ORed into the option word. This option is ignored if a data cache is not present on the adapter, or if a data cache has not been enabled through the kernel configuration `DATA_CACHE` parameter.

- Data cache option

  `CREATE_CACHE_DATA`

  By default, the data section is not cached. To designate the data section cacheable, can be ORed into the option word. This option is ignored if data cache hardware is not present on the adapter, or if data cache has not been enabled through the kernel configuration `DATA_CACHE` parameter.

  CreateProcess ignores the `CREATE_CACHE_DATA` option if the load module that contains the process issuing the CreateProcess was not itself loaded with the data section cacheable. This is because the spawned process shares the data section of the load module.

ProcessID Output. Process ID of the created process.

Reserved Input. Reserved parameter (must be 0).

Returns

`RC_SUCCESS`	`RC_DUP_RES_NAME`
`RC_INVALID_RESERVED_PARM`	`RC_INVALID_MEM_ACCESS`
`RC_INVALID_NAME`	`RC_INVALID_CALL`
`RC_NO_MORE_PROC`	`RC_INVALID_PRIORITY`
`RC_NO_MORE_MEM`	`RC_INVALID_OPTION`
`RC_NO_MORE_RES`	

Remarks

The kernel allocates the stack for the newly created process. The size of stack depends on the StackSize parameter passed to the service. The newly created process shares the code and data area of the calling process. It runs at the priority level set by the creator. The newly created process does not inherit the creator’s resources, exit routine, or floating point usage. Even if the creator is a subsystem, the new process starts as a normal process if the start option is used. The kernel gives control to the newly created process at its entry point, with ParamPtr and ParamSize as parameters.

The new process gets control at *main()* with the arguments parsed into *argc* and *argv* if:

- The passed parameters are built up in the creator’s data area
- The passed parameters are in the format of null-terminated strings with the last string double-null terminated
- The label *ricstart* is passed for the entry point

StartProcess—Start a Process

This service starts a stopped process.

Functional Prototype

```
RIC_ULONG StartProcess (RIC_PROCESSID  ProcessID,
                        RIC_ULONG      Reserved);
```

Parameters

ProcessID Input. Process ID of the process that is to be started.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_PROCESSID
RC_PROCESS_ALREADY_STARTED
RC_INVALID_CALL
```

Remarks

The kernel starts a previously loaded process. The entry point of the process is defined when the process is loaded from the system unit or by the CreateProcess service of the kernel.

StopProcess—Stop a Process

This service stops a previously started process.

Functional Prototype

```
RIC_ULONG StopProcess (RIC_PROCESSID  ProcessID,
                       RIC_ULONG      Reserved);
```

Parameters

ProcessID Input. Process ID of the process that is to be stopped. A value of 0 means that the calling process is stopping itself.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                      RC_INVALID_CALL
RC_INVALID_RESERVED_PARM        RC_PERMANENT_PROCESS
RC_PROCESS_NOT_STARTED          RC_DEVICE_DRIVER
RC_INVALID_PROCESSID
```

Remarks

The kernel calls the exit routine of the process before stopping the process. All the resources acquired by the process are released. This process can be restarted at a later time.

When a process is stopping another process, the requesting process will not run until the stopping process has completely stopped (including execution of its exit handler).

Locally, only a device driver/subsystem can stop a device driver/subsystem. The system unit can stop and unload a device driver/subsystem through the `-U` parameter of the Application Loader utility (see *Application Loader (ricload) Utility* on page 196 for information on this utility). The system unit can stop a device driver/subsystem through a global mailbox command to a kernel mailbox from any unit (see *Chapter 4: Kernel Commands* on page 163 for details on the mailbox commands).

UnloadProcess—Unload a Process

This service unloads a previously loaded process.

Functional Prototype

```
RIC_ULONG UnloadProcess (RIC_PROCESSID  ProcessID,
                         RIC_ULONG      Reserved);
```

Parameters

ProcessID Input. Process ID of the process that is to be unloaded. A value of 0 means that the calling process is unloading itself.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_PROCESSID
RC_INVALID_CALL
RC_PERMANENT_PROCESS
RC_DEVICE_DRIVER
```

Remarks

The kernel calls the exit routine of the process before unloading the process, if the process had been started. All the resources acquired by the process are released. The kernel releases the code, data, parameter, and stack memory areas of the process. The process cannot be restarted without being reloaded.

When a process is stopping another process, the requesting process will not run until the stopping process has completely stopped—including execution of its exit handler.

Locally, only a device driver/subsystem can unload a device driver/subsystem. The system unit can unload a device driver/subsystem through the –U parameter of the Application Loader utility (see *Application Loader (ricload) Utility* on page 196 for details about the utility) or through a global mailbox command to a kernel mailbox from any unit (see *Chapter 4: Kernel Commands* on page 163 for details about mailbox commands).

SuspendProcess—Suspend a Process

This service suspends a process. It is taken off the dispatch queue and its process state is set to `SUSPENDED`.

Functional Prototype

```
RIC_ULONG SuspendProcess (RIC_PROCESSID  ProcessID,
                          RIC_ULONG      Reserved);
```

Parameters

ProcessID Input. Process ID of the process that is to be suspended. A value of 0 means the calling process is suspending itself.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_PROCESS_NOT_STARTED
RC_INVALID_PROCESSID
RC_DEVICE_DRIVER
RC_INVALID_CALL
```

Remarks

None

ResumeProcess—Resume a Process

This service resumes a process.

Functional Prototype

```
RIC_ULONG ResumeProcess (RIC_PROCESSID  ProcessID,
                         RIC_ULONG      Reserved);
```

Parameters

ProcessID Input. Process ID of the process that is to be resumed.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_PROCESS_NOT_STARTED
RC_INVALID_PROCESSID
```

Remarks

When the process is resumed, it is put back on the dispatch queue. If the process is already on the dispatch queue, no action is taken.

If the process was suspended by another process, after it blocked for a semaphore or an event, ResumeProcess will not make it ready to run immediately unless the semaphore or event is also available at the time.

SetExitRoutine—Set the Exit Routine for the Process

This service sets the exit routine for the process.

Functional Prototype

```
RIC_ULONG SetExitRoutine (RIC_VECTOR ExitRoutine,
                          RIC_ULONG  Reserved);
```

Parameters

ExitRoutine Input. Address of the routine the kernel calls when this process is stopped normally or abnormally.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_CALL
RC_INVALID_MEM_ACCESS
```

Remarks

The kernel calls the ExitRoutine of the process when the process is stopped, whether it was normal or abnormal because of asynchronous errors.

This service is mapped to the C function **atexit**, which allows the registration of multiple exit handlers. No kernel trace information is provided for this service.

SetPriority—Set the Priority of the Process

This service changes the priority of the current process.

Functional Prototype

```
RIC_ULONG SetPriority (RIC_PROCESSID  ProcessID,
                       RIC_ULONG      Priority,
                       RIC_ULONG      Reserved);
```

Parameters

ProcessID Input. Sets this process ID to the new priority. A value of 0 means the calling process.

Priority Input. New priority of the process. A value of 0 means the default priority.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_CALL
RC_INVALID_PROCESSID
RC_INVALID_PRIORITY
```

Remarks

The kernel changes the priority of the process to Priority. If the priority of the currently executing process is lowered, a dispatch cycle may occur.

Refer to the *ARTIC960 Programmer's Guide* for the priority recommendations.

QueryPriority—Query the Priority of the Process

This service queries the priority of the process.

Functional Prototype

```
RIC_ULONG QueryPriority (RIC_PROCESSID  ProcessID,
                         RIC_ULONG     *Priority,
                         RIC_ULONG      Reserved);
```

Parameters

ProcessID Input. Queries the priority of this process. A value of 0 means the calling process.

Priority Output. Priority of the process.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_PROCESSID
RC_INVALID_MEM_ACCESS
```

Remarks

None

QueryProcessInExec—Get ID of Process in Execution

This service returns the process ID of the process that currently is executing.

Functional Prototype

```
RIC_ULONG QueryProcessInExec (RIC_PROCESSID  *ProcessID,
                              RIC_ULONG       Reserved);
```

Parameters

ProcessID Output. The process ID of the currently executing process.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_MEM_ACCESS
```

Remarks

At process time, this call returns the caller's process ID. When called in interrupt handlers, this call returns the process that was executing at the time of the interrupt. If no process was executing at the time of the interrupt, ProcessID is set to `INVALID_PROCESSID`.

SetProcessData—Set Process Data

This service sets process instance data for the indicated application environment and process.

Functional Prototype

```
RIC_ULONG SetProcessData (void           *ProcessDataPtr,
                          unsigned char   ApplID,
                          RIC_PROCESSID   ProcessID);
```

Parameters

ProcessDataPtr
Input. Pointer to process instance data.

ApplID Input. Unique ID to indicate which application environment the process instance data is associated with. IDs 0 through 63 are reserved for ARTIC960 use.

ProcessID Input. Indicates which process the instance data is for. A value of 0 indicates the process in execution.

Returns

```
RC_SUCCESS
RC_NO_MORE_RES
RC_INVALID_PROCESSID
RC_INVALID_CALL
```

Remarks

This service maintains process instance data pointers for up to 15 application IDs. If more than 15 application IDs are specified, `RC_NO_MORE_RES` is returned.

This service cannot be called from an interrupt handler. It can be called from a call handler. However, doing so with a ProcessID value of 0 can give unexpected results and should be used with caution. While in a call handler, the process in execution is considered to be the process that called the handler. If call processes are nested, it is the process that called the first handler.

To set process data for a process that is started by CreateProcess, services should be called in the following order:

1. CreateProcess
2. EnterCritSec to disable preemption
3. StartProcess
4. SetProcessData
5. ExitCritSec to enable preemption

GetProcessData—Get Process Data

This service returns the process instance data associated with the indicated application environment and process.

Functional Prototype

```
RIC_ULONG GetProcessData (void           *ProcessDataPtr,
                          unsigned char   ApplID,
                          RIC_PROCESSID   ProcessID);
```

Parameters

ProcessDataPtr
Input. Pointer to location where the kernel returns the pointer to the process instance data. If no process instance data is found, a NULL pointer is returned.

ApplID Input. Unique ID to indicate with which application environment the process instance data is associated.

ProcessID Input. Process ID of the instance data to be retrieved. A value of 0 indicates the process in execution.

Returns

```
RC_SUCCESS
RC_INVALID_PROCESSID
```

Remarks

This service can be called from an interrupt handler or a call handler. However, doing so with a ProcessID value of 0 may give unexpected results and should be used with caution. While in an interrupt handler, the process in execution is considered to be the kernel. While in a call handler, the process in execution is considered to be the process that called the handler. If call processes are nested, it is the process that called the first handler.

EnterCritSec—Enter Critical Section

This service disables interrupts and/or preemptions.

Functional Prototype

```
RIC_ULONG EnterCritSec (RIC_ULONG  OptionWord,
                        RIC_ULONG  Reserved);
```

Parameters

OptionWord

Input.

`DISABLE_INTERRUPTS`
: If ORed into the option word, interrupts are disabled; the default is not to change the interrupt state.

`DISABLE_PREEMPTION`
: If ORed into the option word, preemption is disabled; the default is not to change the preemption state.

Failure to select either option causes an `RC_INVALID_OPTION` to be returned.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_CALL
RC_INVALID_OPTION
```

Remarks

The number of calls to enable interrupts must match the number of calls to disable interrupts, similar to pushes and pops of a stack. The same is true for preemption.

An interrupt handler cannot disable preemption.

The following situation forces a critical section to end. If (1) interrupts or preemption is disabled and (2) a process calls a kernel service that causes the process to block, interrupts and preemption are automatically enabled. This allows the block to proceed. In other words, a blocking call ends a critical section.

ExitCritSec—Exit Critical Section

This service enables interrupts and/or preemption.

Functional Prototype

```
RIC_ULONG ExitCritSec (RIC_ULONG  OptionWord,
                       RIC_ULONG  Reserved);
```

Parameters

OptionWord

Input.

`ENABLE_INTERRUPTS`
: If ORed into the *OptionWord*, interrupts are enabled; the default is not to change the interrupt state.

`ENABLE_PREEMPTION`
: If ORed into the *OptionWord*, preemption is enabled; the default is not to change the preemption state.

Failure to select either option causes an `RC_INVALID_OPTION` to be returned.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_CALL
RC_INVALID_OPTION
```

Remarks

The number of calls to enable interrupts must match the number of calls to disable interrupts, similar to pushes and pops of a stack. The same is true of preemption.

An interrupt handler cannot enable preemption.

Dispatch—Cause a Dispatch Cycle

This service causes a dispatch cycle.

Functional Prototype

```
RIC_ULONG Dispatch (void);
```

Returns

```
RC_SUCCESS
RC_INVALID_CALL
```

Remarks

This service cannot be called from an interrupt handler.

Process Synchronization Services

Process synchronization is accomplished through semaphores and events.

Semaphores are the post/wait mechanism for all processes and come in two types: *mutual exclusion* and *counting* semaphores.

- Mutual exclusion (*mutex*) semaphores are used for serializing access to code or data structures.
- Counting semaphores are used for synchronizing processes, such as synchronizing a producer-consumer pair of processes.

Semaphores can be *explicit* or *implicit*.

- Explicit semaphores are decremented before control returns to the process.
- Implicit semaphores are decremented when the process calls the appropriate resource services, such as removing a queue element or mailbox message.

Processes can allocate and manipulate semaphores using the following services.

Service Name	**Page**
CreateSem	51
OpenSem	52
CloseSem	53
ReleaseSem	54
RequestSem	55
QuerySemCount	56
SetSemCount	57
CreateEvent	58
OpenEvent	59
CloseEvent	60
WaitEvent	61

Refer to the ARTIC960 *Programmer’s Guide* for additional information.

CreateSem—Create a Semaphore

This service creates a semaphore and gives access to the requesting process.

Functional Prototype

```
RIC_ULONG CreateSem (char          *SemName,
                     RIC_ULONG      SemCount,
                     RIC_ULONG      OptionWord,
                     RIC_SEMHANDLE *SemHandle,
                     RIC_ULONG      Reserved);
```

Parameters

SemName Input. A name to assign to the semaphore so other processes can get access to the same semaphore by name. This name can be NULL; however, the semaphore cannot be shared when SemName is NULL. The kernel's subsystems allocate all resources, with the first four characters as "RIC_" for the resource name. User semaphore names should not start with this prefix.

SemCount Input. New count of semaphore. Values greater than `0x80000000` are not permitted. In addition, mutual exclusion semaphores cannot be assigned a count greater than one.

OptionWord Input.

`SEMTYPE_COUNTING`	Specifies the semaphore as a counting type
`SEMTYPE_MUTEX`	Indicates a mutual exclusion type semaphore

SemHandle Output. Semaphore handle returned to requesting process. This handle is passed to all other semaphore services when referring to this semaphore.

Reserved Input. Reserved parameter (must be 0).

Returns

`RC_SUCCESS`	`RC_INVALID_MEM_ACCESS`
`RC_NO_MORE_RES`	`RC_INVALID_CALL`
`RC_INVALID_RESERVED_PARM`	`RC_INVALID_OPTION`
`RC_INVALID_NAME`	`RC_INVALID_SEM_COUNT`
`RC_DUP_RES_NAME`	

Remarks

This service creates a new semaphore and assigns to it the specified name. The usual initial count for counting semaphores is 0; the initial count for mutual exclusion semaphores is 1. To use another starting semaphore count, see *SetSemCount—Set a Semaphore Count* on page 57. Other processes can get access to the same semaphore using the OpenSem service (see *OpenSem—Open a Semaphore* on page 52). If a mutex semaphore is created with a count of 0, the creator owns it also, Otherwise, the first requester owns it.

OpenSem—Open a Semaphore

This service opens a semaphore previously created by another process.

Functional Prototype

```
RIC_ULONG OpenSem (char          *SemName,
                   RIC_SEMHANDLE *SemHandle,
                   RIC_ULONG      Reserved);
```

Parameters

SemName	Input. The semaphore name used to create the semaphore.
SemHandle	Output. Semaphore handle returned to requesting process. This handle is passed to all other semaphore services when referring to this semaphore.
Reserved	Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_NAME_NOT_FOUND
RC_NO_MORE_RES
RC_INVALID_RESERVED_PARM
RC_INVALID_NAME
RC_INVALID_MEM_ACCESS
RC_INVALID_CALL
```

Remarks

This service gets access to a semaphore already created by another process with the CreateSem service.

CloseSem—Close a Semaphore

This service releases access to a semaphore and deletes the semaphore if no other processes have access.

Functional Prototype

```
RIC_ULONG CloseSem (RIC_SEMHANDLE SemHandle,
                    RIC_ULONG     Reserved);
```

Parameters

SemHandle Input. Handle of semaphore to release.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_HANDLE
RC_INVALID_RESERVED_PARM
RC_DEPENDENT_EVENTS
RC_INVALID_CALL
```

Remarks

If the close is issued by a process while other processes still have access to the semaphore, the service removes access rights for the issuing process. When the last process with access rights calls this service, the semaphore ceases to exist. See *CreateSem—Create a Semaphore* on page 51 and *OpenSem—Open a Semaphore* on page 52 for more information.

If a process is stopped or unloaded, the kernel closes all of its resources. It notifies, through asynchronous notification, all other processes that shared those resources that the process has gone away. If a process closes a mutual exclusion semaphore that it owns (that is, it requested the semaphore last but has not released it), all processes waiting for the semaphore are awakened with an error of `RC_OWNER_CLOSED_SEM`. This is done because the code and data protected by the mutual exclusion semaphore may have been left in an indeterminable state. When this happens, the semaphore count is reset to one, so the semaphore can be re-requested if the application process knows that the protected code and data is in a valid state.

ReleaseSem—Release a Semaphore

This service makes a semaphore available to the next process waiting for it.

Functional Prototype

```
RIC_ULONG ReleaseSem (RIC_SEMHANDLE SemHandle,
                      RIC_ULONG     Reserved);
```

Parameters

SemHandle Input. Handle of semaphore to increment.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_HANDLE
RC_INVALID_RESERVED_PARM
RC_SEM_NOT_OWNED
```

Remarks

The next process waiting for the semaphore is posted if this is the only semaphore for which it is waiting. If no processes are waiting, the semaphore count is incremented.

A mutual exclusion semaphore cannot be released with this service twice by the same process, unless it does a RequestSem in between. In addition, a mutual exclusion semaphore cannot be released by a process other than the one that last requested it.

RequestSem—Request a Semaphore

This service waits for a semaphore.

Functional Prototype

```
RIC_ULONG RequestSem (RIC_SEMHANDLE SemHandle,
                      RIC_TIMEOUT   Timeout,
                      RIC_ULONG     Reserved);
```

Parameters

SemHandle Input. Handle of semaphore to decrement.

Timeout Input. Optional timeout for waiting for a semaphore.

–1 Wait indefinitely

0 Return immediately if the semaphores are unavailable.

Any other value from 1 to 65535

Wait time in milliseconds. The granularity of the timer is five milliseconds. The timeout value is rounded up to the next multiple of five, if it is not already a multiple of five.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                    RC_OWNER_CLOSED_SEM
RC_INVALID_HANDLE             RC_SEM_ALREADY_OWNED
RC_TIMEOUT                    RC_INVALID_CALL
RC_INVALID_RESERVED_PARM      RC_INVALID_TIMEOUT
RC_NEW_SEM_COUNT
```

Remarks

If the semaphore count is positive, control returns immediately to the caller and the count is decremented. If the count is zero, the calling process is made to wait. Only processes that have created or opened the semaphore can wait for the semaphore.

Processes are made to wait in a first-in, first-out (FIFO) order, rather than by priority.

If a mutual exclusion semaphore is owned by a process that is stopped, all waiting processes are awakened with an `RC_OWNER_CLOSED_SEM`, indicating the owner was stopped. The error is returned because the code and data protected by the mutual exclusion semaphore may have been left in an indeterminable state. If the semaphore's count is modified using SetSemCount, any process waiting for the semaphore is awakened with `RC_NEW_SEM_COUNT`.

Processes cannot wait for implicit semaphores with this service. Instead, processes should use the services related to the semaphore, such as GetQueue or ReceiveMbx. In addition, implicit semaphores can be part of an event wait.

QuerySemCount—Get a Semaphore Count

This service returns the count of a semaphore.

Functional Prototype

```
RIC_ULONG QuerySemCount (RIC_SEMHANDLE  SemHandle,
                         RIC_ULONG     *SemCount,
                         RIC_ULONG      Reserved);
```

Parameters

SemHandle Input. Handle of semaphore.

SemCount Output. Count of semaphore.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_MEM_ACCESS
RC_INVALID_HANDLE
```

Remarks

If the count is zero, the semaphore is not available. Other processes may be waiting for the semaphore. A positive count indicates the number of times that processes can request the semaphore before they are blocked.

This is the only semaphore service that can be used on implicit semaphores.

SetSemCount—Set a Semaphore Count

This service sets the count of a semaphore.

Functional Prototype

```
RIC_ULONG SetSemCount (RIC_SEMHANDLE SemHandle,
                       RIC_ULONG     SemCount,
                       RIC_ULONG     Reserved);
```

Parameters

SemHandle Input. Semaphore handle.

SemCount Input. New count of semaphore. Values less than zero are not permitted. In addition, mutual exclusion semaphores cannot be assigned a count greater than 1.

Reserved Input. Reserved parameter (must be 0).

Returns

RC_SUCCESS	RC_INVALID_SEM_COUNT
RC_INVALID_HANDLE	RC_PROCESSES_WAITING_ON_SEM
RC_INVALID_RESERVED_PARM	RC_NEW_SEM_COUNT

Remarks

This service should be used immediately after the semaphore is created to configure the semaphore to the desired type. If any processes are waiting for the semaphore when the count is set, they are released and returned with `RC_NEW_SEM_COUNT`. This includes processes waiting for events that include the semaphore.

CreateEvent—Create an Event Word

This service creates an event word based on a semaphore list and mask.

Functional Prototype

```
RIC_ULONG CreateEvent (char          *EvnName,
                       RIC_SEMHANDLE *SemHandles,
                       RIC_ULONG      SemCount,
                       RIC_EVNHANDLE *EvnHandle,
                       RIC_ULONG      Reserved);
```

Parameters

EvnName Input. A name to assign to the event word so that other processes can get access to it.

SemHandles Input. Pointer to an array of up to 32 semaphore handles to associate with the event word. These semaphore handles can be implicit or explicit.

SemCount Input. Number of semaphores in semaphore handle array (no more than 32 semaphores).

EvnHandle Output. Event handle to be used with other event services when referring to this event.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                    RC_NO_MORE_EVNS
RC_NO_MORE_RES                RC_INVALID_HANDLE
RC_INVALID_RESERVED_PARM      RC_INVALID_MEM_ACCESS
RC_INVALID_NAME               RC_INVALID_CALL
RC_DUP_RES_NAME               RC_INVALID_COUNT
RC_DUP_RES_HANDLES
```

Remarks

The semaphore handle list can be any combination of explicit (returned by CreateSem or OpenSem) or implicit (returned by other services, such as queues and mailboxes) semaphores. A process, therefore, can wait for synchronization with other processes as well as resources at the same time. Explicit semaphores are decremented before control returns to the process. Implicit semaphores are decremented when the process calls the appropriate resource services, such as removing a queue element or mailbox message.

OpenEvent—Open Access to an Event Word

This service provides access to a previously created event word.

Functional Prototype

```
RIC_ULONG OpenEvent (char          *EvnName,
                     RIC_EVNHANDLE *EvnHandle,
                     RIC_ULONG      Reserved);
```

Parameters

EvnName Input. Event name to be accessed.

EvnHandle Output. Event handle to be used with other event services when referring to this event.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                  RC_INVALID_NAME
RC_NAME_NOT_FOUND           RC_INVALID_HANDLE
RC_NO_MORE_RES              RC_INVALID_MEM_ACCESS
RC_INVALID_RESERVED_PARM    RC_INVALID_CALL
```

Remarks

The calling process must have already opened the semaphores that make up the event.

CloseEvent—Release Access to an Event Word

This service releases access to an event word and deletes the event, if no other processes have access.

Functional Prototype

```
RIC_ULONG CloseEvent (RIC_EVNHANDLE EvnHandle,
                      RIC_ULONG     Reserved);
```

Parameters

EvnHandle Input. Event handle returned by CreateEvent service.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_HANDLE
RC_INVALID_RESERVED_PARM
RC_INVALID_CALL
```

Remarks

If a process closes an event that is shared with other processes, this service removes access rights for the caller only. Only when the last process closes the event does the event cease to exist.

WaitEvent—Wait on an Event

This service waits for the requesting process until the event occurs.

Functional Prototype

```
RIC_ULONG WaitEvent (RIC_EVNHANDLE   EvnHandle,
                     RIC_ULONG       Mask,
                     RIC_ULONG       OptionWord,
                     RIC_TIMEOUT     Timeout,
                     RIC_ULONG      *Status,
                     RIC_ULONG       Reserved);
```

Parameters

EvnHandle Input. Event handle returned by CreateEvent and OpenEvent services.

Mask Input. Mask telling which semaphores to include in the event wait. If bit *n* is set in the mask, the *nth* semaphore in the semaphore handle array passed to CreateEvent is included in the event wait.

OptionWord Input.

EVENT_WAIT_ALL
: Indicates that the process is awakened only when all the semaphores are available.

EVENT_WAIT_ANY
: Indicates that the process is awakened when the first semaphore becomes available.

Timeout Input. Optional timeout value for waits for events.

–1 Wait indefinitely

0 Return immediately if the semaphores are unavailable.

Any other value from 1 to 65535
: Wait time in milliseconds. The granularity of the timer is five milliseconds. The timeout value is rounded up to the next multiple of five, if it is not already a multiple of five.

Status Output. Bit field that returns which semaphores (that were part of the event wait) were positive/available.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                    RC_INVALID_MEM_ACCESS
RC_INVALID_HANDLE             RC_INVALID_CALL
RC_INVALID_RESERVED_PARM      RC_INVALID_OPTION
RC_TIMEOUT                    RC_INVALID_TIMEOUT
RC_NEW_SEM_COUNT              RC_OWNER_CLOSED_SEM
RC_INVALID_EVN_MASK
```

Remarks

If the OptionWord parameter is set to `EVENT_WAIT_ALL`, the service tests each semaphore count for a positive value. If all are positive, the parameter decrements the explicit semaphores that are positive and control returns to the caller. If all the semaphores do not have a positive value, the requester is waited. When one or more semaphores in the list become available, all other semaphores are tested to determine if they are positive values. Any explicit semaphores that are positive are decremented and control returns to the caller. The performance of this option can be optimized by specifying the semaphore handles least likely to be available first in the list of semaphore handles supplied on the CreateEvent service.

If the OptionWord parameter is set to `EVENT_WAIT_ANY`, the service tests to see if any one of the semaphores is positive. If one is positive, the service decrements the explicit semaphores that are positive and returns to the caller. If no semaphores are positive, the caller is waited. When one or more semaphores in the list become available, the service decrements the explicit semaphores that are positive and control returns to the caller.

If the timeout value is exceeded, the process is awakened, regardless of the state of the event.

If a semaphore included in a wait event gets a new semaphore count, any processes waiting for events that include that semaphore are awakened with the error code `RC_NEW_SEM_COUNT`.

If a process closes a mutex semaphore while owning it, the WaitEvent is canceled with the error code `RC_OWNER_CLOSED_SEM`.

Memory Management Services

The following are the memory management services.

Service Name	Page
CreateMem	64
OpenMem	67
CloseMem	68
ResizeMem	69
SetMemProt	70
SetProcMemProt	71
QueryMemProt	72
QueryProcMemProt	73
QueryFreeMem	74
InitSuballoc	75
GetSuballoc	77
FreeSuballoc	78
GetSuballocSize	79
MallocMem	80
FreeMem	81
CollectMem	82

Refer to the *ARTIC960 Programmer's Guide* for additional information.

CreateMem—Allocate Memory

This service allocates memory from the free storage pool to a requesting process.

Functional Prototype

```
RIC_ULONG CreateMem (char        *MemName,
                     RIC_ULONG    Size,
                     RIC_ULONG    Alignment,
                     RIC_ULONG    Access,
                     RIC_ULONG    MemType,
                     void       **Baseptr,
                     RIC_ULONG    Reserved);
```

Parameters

MemName Input. An optional storage area name to assign to the memory block so that other processes can get access to the same block by name. This name also can be NULL. Memory cannot be shared when MemName is NULL. The kernel's subsystems allocate all resources, with the first four characters as "RIC_" for the resource name. User memory names should not start with this prefix.

Size Input. Size of allocated block in bytes. If the size is 0, `RC_INVALID_SIZE` is returned.

Alignment Input. Boundary alignment for the start of the allocated block. *`Alignment`* values are the log of the boundary number. For example, a 4 KB boundary translates to an *`Alignment`* value of log (4096) = 12. *`Alignment`* values less than 4 KB are rounded up to 4 KB.

Access Input. Bit field specifying the access rights to the memory block. See *Remarks on page 65* for more information.

MemType Input. Flag indicating the type of memory to be allocated: `MEM_TYPE_INSTR` or `MEM_TYPE_PACKET`. By hardware design, the processor is more efficient using instruction memory. Packet memory is more efficient for access from the daughter card or system bus. On adapters that have only packet memory, packet memory is allocated even if instruction memory is requested.

`MEM_TYPE_PACKET`	Allocate packet memory. Return with an error if no packet memory is available.
`MEM_TYPE_INSTR`	Allocate instruction memory. Return with an error if no instruction memory is available.

Baseptr Output. Pointer to allocated memory block.

Reserved Input. Reserved parameter (must be 0).

Returns

RC_SUCCESS	RC_INVALID_MEM_ACCESS
RC_INVALID_RESERVED_PARM	RC_INVALID_CALL
RC_NO_MORE_RES	RC_INVALID_SIZE
RC_INVALID_NAME	RC_INVALID_OPTION
RC_DUP_RES_NAME	RC_INVALID_ALIGNMENT
RC_NO_MORE_MEM	

Remarks

This service is intended for large memory allocations, such as buffer pools. For smaller, more dynamic allocations, see *GetSuballoc—Suballocate Memory* on page 77. The minimum amount of memory that can be allocated is one page (4 KB).

The following constants are defined and can be ORed to create the appropriate access rights for the allocated memory.

`MEM_SHARE`
: Memory can be shared with other processes. The default is memory that cannot be shared.

`MEM_READABLE`
: Memory can be read by the 80960. The default is memory cannot be read or written by the 80960.

`MEM_WRITABLE`
: Memory can be written by the 80960. The default is memory cannot be read or written by the 80960.

`MEM_OVERRIDE_MC_ACCESS`
: The current system bus access to the created memory is overridden. The default is system bus access is not changed.

`MEM_MC_READABLE`
: Memory can be read from the system bus. In addition, the on-card DMA channel can read the memory. The default is memory cannot be read or written from either.

`MEM_MC_WRITABLE`
: Memory can be written from the system bus. In addition, the on-card DMA channel can write to memory. The default is memory cannot be read or written from either.

`MEM_OVERRIDE_AIB_ACCESS`
: The current daughter board access to the created memory is overridden. The default is daughter board access is not changed.

`MEM_AIB_READABLE`
: The daughter board DMA can read from memory. The default is memory cannot be read or written by the daughter board DMA.

`MEM_AIB_WRITABLE`
: The daughter board DMA can write to memory. The default is memory cannot be read or written by the daughter board DMA.

MEM_DCACHE

Memory is cachable. The default is that memory is not cachable. This option should not be used for memory that is accessed by other masters. This option is ignored if data cache hardware is not present on the adapter or if data cache has not been enabled through the kernel configuration DATA_CACHE parameter.

MEM_BIG_ENDIAN

The big-endian address of the allocated memory is returned. The byte order of the allocated memory is big endian. By default, all memory is treated as little endian.

If the kernel does not support big-endian memory regions, RC_INVALID_OPTION is returned. The kernel supports only big-endian memory regions on the ARTIC960Hx adapter.

OpenMem—Get Addressability to Allocated Memory

This service gets addressability to memory allocated by another process.

Functional Prototype

```
RIC_ULONG OpenMem (char       *MemName,
                   RIC_ULONG   Access,
                   void      **Baseptr,
                   RIC_ULONG   Reserved);
```

Parameters

MemName Input. Name of allocated memory. This should be the same as the name used to allocate the memory block.

Access Input. Bit field specifying the access rights to the memory block. These flags are sharable, read/write, and read only. The `MEM_DCACHE` and `MEM_BIG_ENDIAN` flags are ignored by this service. The access rights do not have to be the same as the process that created the memory. The memory must be sharable to be able to open it. See the *Remarks* section under *CreateMem—Allocate Memory* on page 64 for more information.

Baseptr Output. Pointer to memory block.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                     RC_NAME_NOT_FOUND
RC_INVALID_RESERVED_PARM       RC_MEM_SHARING_ERROR
RC_NO_MORE_RES                 RC_INVALID_MEM_ACCESS
RC_INVALID_NAME                RC_INVALID_CALL
RC_INVALID_OPTION
```

Remarks

This service gets access to a memory block allocated with the CreateMem service, provided that the memory was allocated as shareable.

CloseMem—Remove Addressability to Memory

This service releases access to previously allocated memory.

Functional Prototype

```
RIC_ULONG CloseMem (void      *Baseptr,
                    RIC_ULONG  Reserved);
```

Parameters

Baseptr Input. Pointer to allocated memory block.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
C_INVALID_RESERVED_PARM
RC_INVALID_BASEPTR
RC_NO_RES_ACCESS
RC_INVALID_CALL
```

Remarks

This service complements the function of CreateMem and OpenMem. When the last process releases access to a block of memory, the memory is returned to the free storage pool and all access rights are revoked.

ResizeMem—Reallocate Memory

This service resizes allocated memory.

Functional Prototype

```
RIC_ULONG ResizeMem (void      *Baseptr,
                     RIC_ULONG  NewSize,
                     RIC_ULONG  Reserved);
```

Parameters

Baseptr	Input. Pointer to allocated memory block.
NewSize	Input. New size of the memory block in bytes. If the size is 0, `RC_INVALID_SIZE` is returned.
Reserved	Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_BASEPTR
RC_INVALID_SIZE
RC_INVALID_CALL
```

Remarks

The size of the block can be increased only if it does not increase the number of memory pages. The block can always be reduced in size.

SetMemProt—Change Memory Protection

This service changes the access of a process to a block of memory.

Functional Prototype

```
RIC_ULONG SetMemProt (void      *BlockPtr,
                      RIC_ULONG  Size,
                      RIC_ULONG  Access,
                      RIC_ULONG  Reserved);
```

Parameters

BlockPtr Input. Pointer to block of memory. The calling process must have created or opened the memory that contains this block.

Size Input. Size of block of memory in bytes.

Access Input. New access rights to memory. The MEM_DCACHE and MEM_BIG_ENDIAN options are ignored by this service. See the *Remarks* section under *CreateMem—Allocate Memory* on page 64 for more information.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                    RC_INVALID_SIZE
RC_INVALID_RESERVED_PARM      RC_CANT_STOP_SHARING
RC_INVALID_BASEPTR            RC_INVALID_CALL
```

Remarks

If the kernel has been loaded with memory protection enabled, the access rights to the referenced memory block change for the calling process. Only a single set of daughter card and system bus access flags are kept. They are not stored on a per process basis. Therefore, setting these two sets of access flags affects all processes.

To use this service, the process had to have created or opened the memory. This service differs from SetProcMemProt, which does not verify that the caller created or opened the memory. However, SetProcMemProt is available only to device drivers and subsystems.

SetProcMemProt—Change a Process' Memory Protection

This service changes the access of a given process to a block of memory. It is available only to device drivers and subsystems.

Functional Prototype

```
RIC_ULONG SetProcMemProt (RIC_PROCESSID  ProcessID,
                          void          *BlockPtr,
                          RIC_ULONG      Size,
                          RIC_ULONG      Access,
                          RIC_ULONG      Reserved);
```

Parameters

ProcessID	Input. Process ID of process whose access is to be set.
BlockPtr	Input. Pointer to a block of memory.
Size	Input. Size of block of memory in bytes.
Access	Input. New access rights to memory. The `MEM_DCACHE` and `MEM_BIG_ENDIAN` options are ignored by this service. See the *Remarks* section of *CreateMem—Allocate Memory* on page 64 for more information.
Reserved	Input. Reserved parameter (must be 0).

Returns

RC_SUCCESS	RC_INVALID_ADDRESS
RC_INVALID_RESERVED_PARM	RC_NOT_DD_OR_SS
RC_INVALID_PROCESSID	RC_INVALID_OPTION

Remarks

If the kernel has been loaded with memory protection enabled, the access rights to the referenced memory block change for the given process. Only a single set of daughter card and system bus access flags are kept. They are not stored on a per process basis. Therefore, setting these two sets of access flags affect all processes.

This service is available only to device drivers and subsystems so they can gain access to client memory areas.

QueryMemProt—Query Memory Protection

This service queries the memory protection of a block of memory.

Functional Prototype

```
RIC_ULONG QueryMemProt (void       *BlockPtr,
                        RIC_ULONG   Size,
                        RIC_ULONG  *Access,
                        RIC_ULONG   Reserved);
```

Parameters

BlockPtr Input. Pointer to block of memory. The caller must have created or opened the memory that contains this block.

Size Input. Size of block to query.

Access Output. Access rights to memory. This can include the MEM_DCACHE and MEM_BIG_ENDIAN options. See the *Remarks* section under *CreateMem—Allocate Memory* on page 64 for more information.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_BASEPTR
RC_INVALID_RESERVED_PARM
RC_INVALID_CALL
RC_INVALID_MEM_ACCESS
RC_INVALID_SIZE
```

Remarks

This service returns the access rights to the memory block for the calling process. Only a single set of daughter card and system bus access flags are saved by the memory protection services. Therefore, this service returns the same value for these two sets of flags, regardless of the caller’s process ID.

QueryProcMemProt—Query a Process' Memory Protection

This service queries the memory protection of a block of memory for a given process. It is available only to device drivers and subsystems.

Functional Prototype

```
RIC_ULONG QueryProcMemProt (RIC_PROCESSID  ProcessID,
                            void          *BlockPtr,
                            RIC_ULONG      Size,
                            RIC_ULONG     *Access,
                            RIC_ULONG      Reserved);
```

Parameters

ProcessID	Input. Process ID of process whose memory protection is to be queried.
BlockPtr	Input. Pointer to a block of memory.
Size	Input. Size of block to query.
Access	Output. Access rights to memory. This can include the `MEM_DCACHE` and `MEM_BIG_ENDIAN` options. See the *Remarks* section under *CreateMem—Allocate Memory* on page 64 for more information.
Reserved	Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_PROCESSID
RC_INVALID_ADDRESS
RC_INVALID_MEM_ACCESS
RC_NOT_DD_OR_SS
```

Remarks

This service returns the access rights to the memory block for the given process. This service is made available for device drivers and subsystems so that they can check memory access for their clients.

QueryFreeMem—Query Free Memory

This service returns the total amount of free memory and the size of the largest unallocated block of memory.

Functional Prototype

```
RIC_ULONG QueryFreeMem (RIC_ULONG   OptionWord,
                        RIC_ULONG  *Largest,
                        RIC_ULONG  *Total,
                        RIC_ULONG   Reserved);
```

Parameters

OptionWord

Input.

MEM_TYPE_PACKET	Free packet memory
MEM_TYPE_INSTR	Free instruction memory

Largest Output. Size of largest block of free memory in bytes.

Total Output. Total amount of free memory in bytes.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_MEM_ACCESS
RC_INVALID_OPTION
```

Remarks

None

InitSuballoc—Prepare a Block of Memory for Suballocation

This service prepares a block of allocated memory area for suballocation.

Functional Prototype

```
RIC_ULONG InitSuballoc (void       *BlockPtr,
                        RIC_ULONG   Size,
                        RIC_ULONG   Alignment,
                        RIC_ULONG   SuballocUnit,
                        RIC_ULONG   Reserved);
```

Parameters

BlockPtr Input. Pointer to block of memory. On cards that support big-endian memory regions, the memory must have been created as little endian. If a big-endian pointer is given, `RC_INVALID_BASEPTR` is returned.

Size Input. Size of block in bytes.

Alignment Input. Boundary alignment of suballocated memory. Alignment values are the log of the boundary number. For example:

- An 8-byte boundary would translate to an Alignment value of $\log_2(8)=3$.
- A 4 KB boundary would translate to an Alignment value of $\log_2(4096)=12$.

Alignment defaults to 1 byte if a value of 0 is passed.

SuballocUnit

Input. Size of smallest block of memory that can be suballocated. Larger suballocated memory blocks are suballocated as multiples of this unit. The unit size is rounded up to the next power of 2.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                  RC_SUBALLOCATED_MEM
RC_INVALID_RESERVED_PARM    RC_INVALID_CALL
RC_INVALID_BASEPTR          RC_INVALID_ALIGNMENT
RC_INVALID_SIZE
```

Remarks

The block must be contained within memory that is accessible to the calling process. The suballocation unit size helps tune the suballocation services for higher-performance and lower-memory utilization. The suballocation unit size should be as large as possible, while still mapping well to the size of the expected suballocations. Bit map allocation is used to implement suballocation—the larger the unit size, the fewer the bits required to represent the pool. This results in smaller bit map size and quicker searches of the bit map.

When calculating the alignment of suballocation chunks, this service rounds the unit size up to the next power of two. The actual alignment is the larger of this rounded value and the alignment represented by the Alignment parameter. For example:

- A unit size of 4 bytes and Alignment value of 0 (1-byte boundary) result in suballocation on 4-byte boundaries.
- A unit size of 4 bytes and an Alignment value of 4 (16-byte boundary) result in suballocation on 16-byte boundaries.
- A unit size of 3 and an Alignment value of 1 (2-byte boundaries) result in suballocation on 4-byte boundaries.

Use GetSuballocSize to determine the proper size of the block to accommodate the requested number of suballocation units and the bit map.

GetSuballoc—Suballocate Memory

This service suballocates memory from previously allocated memory.

Functional Prototype

```
RIC_ULONG GetSuballoc (void        *Blockptr,
                       RIC_ULONG    Size,
                       void       **Suballocptr,
                       RIC_ULONG    Reserved);
```

Parameters

Blockptr Input. Pointer to beginning of suballocation pool. On cards that support big-endian memory regions, the memory must have been created as little endian. If a big-endian pointer is given, RC_INVALID_BASEPTR is returned.

Size Input. Amount of memory in bytes to suballocate. The size is rounded up to a multiple of the suballocation unit size set with InitSuballoc. If the size is 0, RC_INVALID_SIZE is returned.

Suballocptr Output. Pointer to suballocated memory.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                     RC_NO_MORE_MEM
RC_INVALID_RESERVED_PARM       RC_INVALID_MEM_ACCESS
RC_INVALID_BASEPTR             RC_INVALID_SIZE
```

Remarks

No more than 65535 (64K–1) times the suballocation unit size in bytes can be suballocated with a single call to GetSuballoc. This restriction lowers the memory overhead of the suballocation services.

Application writers should be aware that the kernel’s suballocation control information is stored in the user’s memory, unlike all the other kernel services whose control information is in protected memory. This decision was made to improve suballocation performance, but it potentially allows corruption of kernel suballocation data structures.

FreeSuballoc—Free Suballocated Memory

This service frees suballocated memory.

Functional Prototype

```
RIC_ULONG FreeSuballoc (void       *Blockptr,
                        void       *Suballocptr,
                        RIC_ULONG   Reserved);
```

Parameters

Blockptr Input. Pointer to beginning of suballocation pool.

Suballocptr Input. Pointer to suballocated memory.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_BASEPTR
RC_INVALID_SUBALLOC_ADDR
```

Remarks

None

GetSuballocSize—Return Size of Suballocation Pool

This service returns the amount of memory required for a suballocation pool.

Functional Prototype

```
RIC_ULONG GetSuballocSize (RIC_ULONG  UnitCount,
                           RIC_ULONG  UnitSize,
                           RIC_ULONG  Alignment,
                           RIC_ULONG *SuballocSize,
                           RIC_ULONG  Reserved);
```

Parameters

UnitCount Input. Number of suballocation blocks in the pool.

UnitSize Input. Size of the smallest block of memory that can be suballocated. Larger suballocated memory blocks are suballocated as multiples of this unit. The unit size is rounded up to the next power of 2. If the size is 0, `RC_INVALID_SIZE` is returned.

Alignment Input. Boundary alignment of suballocated memory. Alignment values are the log of the boundary number. For example, a 16-byte boundary would translate to an Alignment value of $\log_2(16)=4$.

SuballocSize
Output. Number of bytes of memory required to make a suballocation pool with the given suballocation unit size and number of units. This size can then be used to calculate the amount of memory to allocate with CreateMem.

Reserved Input. Reserved parameter (must be 0).

Returns

RC_SUCCESS	RC_INVALID_COUNT
RC_INVALID_RESERVED_PARM	RC_INVALID_SIZE
RC_INVALID_MEM_ACCESS	RC_INVALID_ALIGNMENT

Remarks

This service should be used by applications using the suballocation services so that they know how much memory to allocate for a suballocation pool. This service returns only a byte count. It does not actually allocate or initialize any memory.

When calculating the alignment of suballocation chunks, this service rounds the unit size up to the next power of two. The actual alignment is the larger of this rounded value and the alignment represented by the Alignment parameter. For example:

- A unit size of 4 bytes and Alignment value of 0 (1-byte boundary) results in suballocation on 4-byte boundaries.
- A unit size of 4 bytes and an Alignment value of 4 (16-byte boundary) results in suballocation on 16-byte boundaries.
- A unit size of 3 and an Alignment value of 1 (2-byte boundaries) results in suballocation on 4-byte boundaries.

MallocMem—Allocate Memory

This service allocates a block of memory from the dynamic memory pool.

Functional Prototype

```
void *     MallocMem (RIC_ULONG  Size,
                      RIC_ULONG  OptionWord);
```

Parameters

Size Input. Size in bytes of memory block to be allocated.

OptionWord

Input. Bit field to describe the options to be used to allocate the memory. The following constants should be ORed together to build the appropriate set of options.

- Type of memory to create

  By default, memory is allocated without regard to memory type. If the option word is set to `OPTION_PACKET_MEMORY`, memory is allocated from packet memory. If memory protection is active, the packet memory is given system bus read/write access. The default option is `OPTION_ANY_MEMORY`.

- Data cache option for created memory

  By default, memory is not created as cachable. To create cachable memory, `MEM_DCACHE` can be ORed into the option word. This option should not be used for memory that is accessed by other masters. This option is ignored if data cache hardware is not present on the adapter or if data cache has not been enabled through the kernel configuration `DATA_CACHE` parameter.

- Big-endian option for created memory

  By default, memory is created little endian. To create memory for big-endian access, `MEM_BIG_ENDIAN` can be ORed into the option word. This option is valid only on the ARTIC960Hx PCI adapter. An invalid option causes a value of NULL to be returned.

Returns

Pointer to the allocated memory. A NULL pointer means that no memory is available or that an invalid size or option was specified.

Remarks

This service can be called from an interrupt handler.

The C library **malloc** function is mapped into this service using the default option.

FreeMem—Free Memory

This service returns a block of memory that was allocated using the service MallocMem to the dynamic memory pool.

Functional Prototype

```
RIC_ULONG FreeMem (void    *Blockptr);
```

Parameters

Blockptr   Input. Pointer to the memory block to be freed.

Returns

```
RC_SUCCESS
RC_INVALID_BASEPTR
```

Remarks

This service can be called from an interrupt handler.

The C library **free** function is mapped into this service.

CollectMem—Collect Memory

This service returns pages of memory that are in dynamic memory pools and are not being used. The pages are returned to the memory page pool. It also provides information about the amount of memory available in dynamic memory pools after collection is done.

Functional Prototype

```
RIC_ULONG CollectMem (RIC_ULONG  OptionWord,
                      RIC_ULONG *FreeUnits,
                      RIC_ULONG *FreedPages);
```

Parameters

OptionWord

Input. A bit field specifying options for the CollectMem service.

`OPTION_COLLECT_PROCESS`

Unused pages belonging to the dynamic memory pool of the process in execution are returned. `OPTION_COLLECT_PROCESS` is meaningful only if memory protection is active.

`OPTION_COLLECT_ALL`

All unused pages in both the general and the process-specific dynamic memory pools are returned. This is the default.

FreeUnits

Output. The number of free units that exist in the dynamic memory pools after the collection is done. If `OPTION_COLLECT_PROCESS` was used, it reflects the number of free units in the dynamic memory pools of the process. A unit is 32 bytes.

FreedPages

Output. The number of pages in dynamic memory pools that were returned to the Memory Page Pool after the collection is done.

Returns

```
RC_SUCCESS
RC_INVALID_OPTION
RC_INVALID_CALL
```

Remarks

This service cannot be called from an interrupt handler.

If `RC_NO_MORE_xxx` is returned by a service, CollectMem can be issued to make unused pages available. Then the service can be retried to determine if enough memory is available.

Timer Services

The following are the timer services.

Refer to the *ARTIC960 Programmer's Guide* for additional information.

CreateSwTimer—Allocate a Software Timer

This service creates a software timer and gives access to the requesting process.

Functional Prototype

```
RIC_ULONG CreateSwTimer (char          *TimerName,
                         RIC_TMRHANDLE *TimerHandle,
                         RIC_ULONG      Reserved);
```

Parameters

TimerName Input. A name to assign to the timer. This parameter also can be NULL. The kernel subsystems allocate all resources, with the first four characters as “RIC_” for the resource name. User timer names should not start with this prefix.

TimerHandle

Input. Timer handle returned to requesting process. This handle is passed to all other timer services when this timer is referenced.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_DUP_RES_NAME
RC_NO_MORE_RES
RC_INVALID_NAME
RC_INVALID_MEM_ACCESS
RC_INVALID_CALL
```

Remarks

This service creates a new software timer and assigns it the specified name. Because software timers cannot be shared, there is not an equivalent *open* service.

The granularity of the software timer is five milliseconds. The TimeCount value is rounded up to the next multiple of five, if it is not already a multiple of five.

CloseSwTimer—Return a Software Timer

This service returns a previously created software timer.

Functional Prototype

```
RIC_ULONG CloseSwTimer (RIC_TMRHANDLE TimerHandle,
                        RIC_ULONG     Reserved);
```

Parameters

TimerHandle
Input. Timer handle of the timer to be returned. This handle is passed to the process by the service CreateSwTimer.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_HANDLE
RC_INVALID_CALL
```

Remarks

This service returns a previously-created software timer. The process cannot access this software timer any more. This call stops a timer that is started.

StartSwTimer—Start a Software Timer

This service starts a software timer.

Functional Prototype

```
RIC_ULONG StartSwTimer (RIC_TMRHANDLE   TimerHandle,
                        RIC_ULONG       TimeCount,
                        RIC_TMRHANDLER  TimerHandler,
                        RIC_ULONG       OptionWord,
                        RIC_ULONG       TimerMemo,
                        RIC_ULONG       Reserved);
```

Parameters

TimerHandle
Input. Timer handle of the timer to be started. This handle is passed to the process by the service CreateSwTimer.

TimeCount Input. Timeout count. This parameter is specified in terms of milliseconds and can range from 1 to 65535. The granularity of the timer is five milliseconds. The timeout value is rounded up to the next multiple of five, if it is not already a multiple of five. A value of 0 is not valid.

TimerHandler
Input. Address of timer handler.

OptionWord
Input. A set of options for starting the software timer.

`TIMER_REPEAT`
If the constant is ORed with OptionWord, the timer is restarted after expiration. This occurs until the user stops the timer using StopSwTimer, or restarts it with another StartSwTimer.

`TIMER_ONE_SHOT`
The timer is not restarted.

`OPTION_PROT_ON`
If the constant is ORed with OptionWord and global protection is on, memory protection is enabled for the timer handler.

`OPTION_PROT_OFF`
The timer handler runs without memory protection.

TimerMemo Input. Optional user-defined input.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_HANDLE
RC_INVALID_MEM_ACCESS
RC_INVALID_OPTION
RC_INVALID_TIMEOUT
RC_NO_BASE_DEVICE_DRIVER
```

Remarks

This service starts a software timer for the requested timer interval unconditionally. When the timer expires, the kernel gives control to the timer handler TimerMemo as the parameter. The timer handler runs as an extension of the kernel interrupt handler.

Because a timer handler is an interrupt routine, care should be taken not to remain in the timer handler for very long.

If `TIMER_REPEAT` is ORed with OptionWord, the timer is restarted when the kernel gets control back from the timer handler of the process.

The process can stop the timer at any time with the StopSwTimer service.

StopSwTimer—Stop a Software Timer

This service stops a previously started software timer.

Functional Prototype

```
RIC_ULONG StopSwTimer(RIC_TMRHANDLE TimerHandle,
                      RIC_ULONG     Reserved);
```

Parameters

TimerHandle
: Input. Handle of the timer to be stopped. This handle is passed to the process by the service CreateSwTimer.

Reserved
: Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_HANDLE
```

Remarks

This service stops a previously started software timer. When called from an interrupt handler, this service can potentially stop a recently expired software timer (that is, the software timer expired but the timer handler of the process was not called yet).

SetSystemTime—Set the Time-of-Day Clock

This service sets the time-of-day clock.

Functional Prototype

```
RIC_ULONG SetSystemTime (struct TimeInfo *SysTimeInfo,
                         RIC_ULONG        Reserved);
```

Parameters

SysTimeInfo
Input. Pointer to a user's structure that contains time information (see the *Remarks* section for this service).

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_NO_BASE_DEVICE_DRIVER
```

Remarks

The TimeInfo is defined as follows:

```
struct TimeInfo
{
   RIC_ULONG Time;
   RIC_SLONG TimeZone;
   RIC_LONG  DayLight;
   char      TimeZoneStr[4];
   char      DayLightStr[4];
}
```

Time Is the time in seconds in GMT since 1970.

TimeZone Is the difference in hours between the local time zone and GMT.

DayLight Is true if daylight savings time is to be applied.

TimeZoneStr
Is a time zone character string, for example, EST and CST.

DayLightStr
Is a daylight savings time zone, for example, EDT and PDT.

QuerySystemTime—Get the Time of Day

This service gets the time of day.

Functional Prototype

```
RIC_ULONG QuerySystemTime (struct TimeInfo *Time,
                           RIC_ULONG        Reserved);
```

Parameters

Time Output. Pointer to a user's structure that contains time information (see the *Remarks* section for this service).

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_MEM_ACCESS
RC_INVALID_RESERVED_PARM
RC_NO_BASE_DEVICE_DRIVER
RC_TOD_NOT_ENABLED
```

Remarks

Users typically use the standard C calls for getting and setting the time. The underlying services call this kernel service.

StartPerfTimer—Start the Performance Timer

This service starts the performance timer.

Functional Prototype

```
RIC_ULONG StartPerfTimer (RIC_ULONG Reserved);
```

Parameters

Reserved	Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_TIMER_IS_ACTIVE
RC_INVALID_RESERVED_PARM
RC_NO_BASE_DEVICE_DRIVER
RC_PERF_TIMER_NOT_ENABLED
```

Remarks

The range of the performance timer is from 1 microsecond to 6 seconds.

The performance timer cannot be restarted once it is active. To restart the performance timer, it must first be stopped with StopPerfTimer. As long as users check the return code from this service, it effectively serializes use of the performance timer.

StopPerfTimer—Stop the Performance Timer

This service stops the performance timer and returns the final time.

Functional Prototype

```
RIC_ULONG StopPerfTimer (RIC_ULONG *TimeCount,
                         RIC_ULONG  Reserved);
```

Parameters

TimeCount Output. Final count of performance timer in microseconds.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_MEM_ACCESS
RC_INVALID_RESERVED_PARM
RC_TIMER_OVERFLOWED
RC_PERF_TIMER_NOT_ENABLED
```

Remarks

None

ReadPerfTimer—Read Current Time of the Performance Timer

This service reads the performance timer count without stopping it.

Functional Prototype

```
RIC_ULONG ReadPerfTimer (RIC_ULONG *TimeCount,
                         RIC_ULONG  Reserved);
```

Parameters

TimeCount	Output. Current count of performance timer in microseconds.
Reserved	Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_TIMER_IS_INACTIVE
RC_INVALID_MEM_ACCESS
RC_INVALID_RESERVED_PARM
RC_TIMER_OVERFLOWED
RC_PERF_TIMER_NOT_ENABLED
```

Remarks

None

Process Communication Services

Using the following services, process communication can be accomplished through queues, mailboxes, and signals.

Refer to the *ARTIC960 Programmer's Guide* for additional information.

CreateQueue—Create a Queue

This service creates a queue and gives access to the requesting process.

Functional Prototype

```
RIC_ULONG CreateQueue (char          *QueueName,
                       RIC_QUEHANDLE *QueueHandle,
                       RIC_SEMHANDLE *SemHandle,
                       RIC_ULONG      Reserved);
```

Parameters

QueueName
: Input. A queue name to assign to the queue so that other processes can access the same queue by name. This name also can be NULL. The queue cannot be shared when QueueName is NULL. The kernel's subsystems allocate all resources with the first four characters being "RIC_" for the resource name. User queue names should not start with this prefix.

QueueHandle
: Output. Queue handle returned to requesting process. This handle is passed to all other queue services when this queue is referenced.

SemHandle
: Output. Handle of the semaphore used to wait on queue elements. The handle is returned so that it can be part of a multiple event wait.

Reserved
: Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_NO_MORE_RES
RC_INVALID_RESERVED_PARM
RC_INVALID_NAME
RC_DUP_RES_NAME
RC_INVALID_MEM_ACCESS
RC_INVALID_CALL
```

Remarks

This service creates a new queue and assigns it the specified name. Other processes can access the same queue with the OpenQueue service. The initial semaphore count is set to 0. It is up to the process to ensure that it has read/write memory access to all queue elements.

Multiple processes can read and receive from a single queue.

OpenQueue—Open a Queue

This service opens a queue previously created by another process.

Functional Prototype

```
RIC_ULONG OpenQueue (char          *QueueName,
                     RIC_QUEHANDLE *QueueHandle,
                     RIC_SEMHANDLE *SemHandle,
                     RIC_ULONG      Reserved);
```

Parameters

QueueName
Input. The queue name used to create the queue.

QueueHandle
Output. Queue handle returned to the requesting process. This handle is passed to all other queue services when this queue is referenced.

SemHandle Output. Handle of the semaphore used to wait on queue elements. This is returned so it can be part of a multiple event wait.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                    RC_INVALID_NAME
RC_NAME_NOT_FOUND             RC_NO_MORE_TIMERS
RC_NO_MORE_RES                RC_INVALID_CALL
RC_INVALID_RESERVED_PARM      RC_INVALID_MEM_ACCESS
```

Remarks

This service gets access to a queue already created by another process with the CreateQueue service. It is up to the process to ensure that it has read/write memory access to the queue elements.

Multiple processes can read and receive from a single queue.

CloseQueue—Close a Queue

This service releases access to a queue and deletes the queue if no other processes have access to it.

Functional Prototype

```
RIC_ULONG CloseQueue (RIC_QUEHANDLE QueueHandle,
                      RIC_ULONG     Reserved);
```

Parameters

QueueHandle
Input. Queue handle of queue to release.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_HANDLE
RC_INVALID_RESERVED_PARM
RC_DEPENDENT_EVENTS
RC_INVALID_CALL
```

Remarks

If the close is issued by a process while other processes still have access to the queue, the service removes access rights for the issuing process. When the last process with access rights calls this service, the queue ceases to exist. See the services *CreateQueue—Create a Queue* on page 95 and *OpenQueue—Open a Queue* on page 96 for more information.

If the close is issued by the kernel on behalf of the process, such as when the kernel cleans up resources for a process that is stopped or unloaded, all other processes are notified through their asynchronous handlers that the process has gone away.

PutQueue—Put an Element into a Queue

This service puts a queue element on a queue and increments the semaphore associated with the queue.

Functional Prototype

```
RIC_ULONG PutQueue (RIC_QUEHANDLE  QueueHandle,
                    void          *Element,
                    RIC_ULONG      OptionWord,
                    RIC_ULONG     *QueueStatus,
                    RIC_ULONG      Reserved);
```

Parameters

QueueHandle
Input. Handle of queue to add element to.

Element Input. Pointer to element to add to queue.

OptionWord
Input.

`QUE_PUT_LIFO`	The queue element is added to the head of the queue.
`QUE_PUT_FIFO`	A queue element is added to the end of the queue.

QueueStatus
Output. Returns the status of the queue.

`QUE_EMPTY`	The queue went from empty to not-empty.
`QUE_NOT_EMPTY`	The queue already had at least one element on the queue.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_HANDLE
RC_INVALID_RESERVED_PARM
RC_INVALID_MEM_ACCESS
RC_INVALID_OPTION
```

Remarks

Eight bytes must be reserved at the top of the queue element for queueing service pointers.

- If all elements are queued with the `QUE_PUT_LIFO` flag on, the queue becomes a virtual stack.
- If all elements are queued with the `QUE_PUT_LIFO` flag off, the queue manages the elements in FIFO order as expected.

- If the elements are queued alternating between `QUE_PUT_LIFO` on and off, a two-priority queue is built.
  - Elements added with the `QUE_PUT_LIFO` flag on have a higher priority because they are put at the front of the queue.
  - Elements added with the `QUE_PUT_LIFO` flag off have a lower priority because they are put at the back of the queue.

GetQueue—Get or Peek at an Element on a Queue

This service gets or peeks at the top element of a queue. If the element is removed from the queue, the semaphore associated with the queue is decremented.

Functional Prototype

```
RIC_ULONG GetQueue (RIC_QUEHANDLE   QueueHandle,
                    void          **Element,
                    RIC_TIMEOUT     Timeout,
                    RIC_ULONG       OptionWord,
                    RIC_ULONG      *QueueStatus,
                    RIC_ULONG       Reserved);
```

Parameters

QueueHandle
Input. Handle of queue to get element from.

Element Output. Pointer to element removed from the queue.

Timeout Input. Size of time to wait for queue element.

–1 Wait indefinitely
0 Return immediately if there are no queue elements.
Any other value from 1 to 65535
Wait time in milliseconds. The granularity of the timer is five milliseconds. The timeout value is rounded up to the next multiple of five, if it is not already a multiple of five.

OptionWord
Input. Bit field that gives receive options.

`QUE_READ`
This bit should be set if the process wants only to peek at the top element of the queue without removing it from the queue.
`QUE_GET`
The queue element is removed from the queue.

QueueStatus
Output. Returns the status of the queue.

`QUE_EMPTY`
If returned, the queue went from not-empty to empty.
`QUE_NOT_EMPTY`
If returned, the queue still has at least one element in the queue.

Reserved Input. Reserved parameter (must be 0).

Returns

RC_SUCCESS	RC_INVALID_MEM_ACCESS
RC_INVALID_HANDLE	RC_INVALID_CALL
RC_INVALID_RESERVED_PARM	RC_INVALID_OPTION
RC_QUEUE_EMPTY	RC_INVALID_TIMEOUT

Remarks

If the `QUE_READ` bit of OptionWord is set, and if more than one process is reading the queue, each may get a pointer to the same queue element.

SearchQueue—Search a Queue for an Element

This service searches a queue for a queue element and optionally removes it from the queue.

Functional Prototype

```
RIC_ULONG SearchQueue (RIC_QUEHANDLE   QueueHandle,
                       void          **Element,
                       RIC_ULONG       OptionWord,
                       RIC_ULONG       KeyValue,
                       RIC_ULONG       KeyOffset,
                       RIC_ULONG       KeyMask,
                       RIC_ULONG       Reserved);
```

Parameters

QueueHandle
: Input. Handle of queue to search for element.

Element
: Output. Pointer to queue element.

OptionWord
: Input. Option word indicating how to do search.

`QUE_SEARCH_ADDRS`
: If ORed with OptionWord, the KeyValue parameter is an element address to search for.

`QUE_SEARCH_KEY`
: If ORed with OptionWord, the KeyValue is a key value within the queue elements to search for.

If the queue element is found:

`QUE_GET`
: If specified, the element is removed from the queue and the queue's semaphore is decremented.

`QUE_READ`
: If specified, the pointer to the element is returned and the queue element is not removed.

KeyValue
: Input. Either the address of the element to search for or the value with the queue element to search for.

KeyOffset
: Input. If the `QUE_SEARCH_KEY` is set, this parameter indicates the offset within the queue element where the key value is found. The key word must be located on a word (4-byte) boundary.

KeyMask
: Input. If the `QUE_SEARCH_KEY` is set, this parameter indicates the mask to be ANDed with the key word before comparing it with the KeyValue.

Reserved
: Input. Reserved parameter (must be 0).

Returns

RC_SUCCESS	RC_ELEMENT_NOT_FOUND
RC_INVALID_HANDLE	RC_INVALID_MEM_ACCESS
RC_INVALID_RESERVED_PARM	RC_INVALID_OPTION

Remarks

If the queue element is not found, control returns to the calling process. This service will not wait until a queue element arrives that satisfies the search criteria.

CreateMbx—Create a Mailbox

This service creates a mailbox and gives access to the requesting process.

Functional Prototype

```
RIC_ULONG CreateMbx (char          *MbxName,
                     char          *MbxRxMemName,
                     RIC_ULONG      MsgUnitSize,
                     RIC_ULONG      MsgUnitCount,
                     RIC_ULONG      OptionWord,
                     RIC_MBXHANDLE *MbxHandle,
                     RIC_SEMHANDLE *SemHandle,
                     RIC_ULONG      Reserved);
```

Parameters

MbxName
Input. A mailbox name to assign to the mailbox so other processes can access the same mailbox by name. This name also can be NULL. The mailbox cannot be shared when MbxName is NULL.

The kernel's subsystems allocate all resources, with the first four characters as "RIC_" for the resource name. User mailbox names should not start with this prefix.

MbxRxMemName
Input. Optional storage-area name associated with this mailbox for receiving messages. A value of NULL means there is no name associated with the memory and memory cannot be shared.

MsgUnitSize
Input. The smallest allocatable message size. All messages are allocated in units of this size. If the size is 0, `RC_INVALID_SIZE` is returned.

MsgUnitCounpmt
Input. The maximum number of message units that can be allocated from this mailbox.

OptionWord
Input. Bit field to describe the options to be used to create the mailbox. The following constants should be ORed together to build the appropriate set of options.

- Type of mailbox to create. The caller can create either type.

  `MBX_CREATE_GLOBAL`
  Mailbox accepts messages from other peer units

  `MBX_CREATE_LOCAL`
  Mailbox does not accept messages from other units

- Type of memory access for storage area. The caller can OR the following constants together to specify both types of access to the memory. The default is that neither access type is given for a local mailbox and that system bus access is given for a global mailbox.

  `MBX_MEM_MC_ACCESS`
  System-bus access rights to the memory

  `MBX_MEM_AIB_ACCESS`
  Daughter card access rights to the memory

MbxHandle
Output. Mailbox handle returned to requesting process. This handle is passed to all other mailbox services when this mailbox is referred to.

SemHandle Output. Semaphore handle associated with the mailbox. This handle is passed to event services when this mailbox-associated semaphore is referred to.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_NAME
RC_DUP_RES_NAME
RC_INVALID_OPTION
RC_NO_MORE_RES
RC_NO_MORE_MEM
RC_INVALID_MEM_ACCESS
RC_INVALID_CALL
RC_NO_MORE_SEM
RC_NO_MORE_TIMERS
RC_INVALID_COUNT
RC_INVALID_SIZE
```

Remarks

This service creates a semaphore associated with this mailbox. The initial semaphore count is set to 0. Users can wait on this semaphore through WaitEvent to receive messages.

This service also allocates the memory requested by the user. This memory is used to keep the messages in the mailbox. Optionally, a name can be assigned to this memory and the sending process can access this memory by passing the same name to OpenMbx. Sharing the memory between sending and receiving processes avoids a copy operation by SendMbx. Refer to the *ARTIC960 Programmer's Guide* for more information about mailbox memory options.

Optionally, mailboxes can accept messages from other peer units. The processes on other units can access this mailbox using OpenMbx. Only the process that created the mailbox with the CreateMbx service can receive messages from the mailbox.

If the process is sharing the receive memory of the mailbox with a previously created mailbox, the MsgUnitSize and MsgUnitCount parameters must be the same value on both create calls. If the mailbox receives messages from other units, the kernel ensures system bus access has been enabled for the mailbox's pool.

Mailbox memory areas are allocated from packet memory. If there is not enough packet memory to allocate the buffer, the `RC_NO_MORE_MEM` error is returned.

If a mailbox is not going to be accessed from off-card, it should be created with the `MBX_CREATE_LOCAL` option.

OpenMbx—Open a Mailbox

This service opens a mailbox previously created by another process.

Functional Prototype

```
RIC_ULONG OpenMbx (char           *MbxName,
                   char           *SendMbxMemName,
                   RIC_ULONG       MsgUnitSize,
                   RIC_ULONG       MsgUnitCount,
                   RIC_ULONG       OptionWord,
                   RIC_MBXHANDLE  *MbxHandle,
                   RIC_ULONG      *MbxType,
                   RIC_ULONG       Reserved);
```

Parameters

MbxName Input. A mailbox name used to create the mailbox.

SendMbxMemName
: Input. For local mailboxes, an optional storage-area name associated with the mailbox for sending messages by this process. A value of NULL means that there is no name associated with the memory and the memory cannot be shared. Refer to the *ARTIC960 Programmer's Guide* for more information about mailbox memory options.

MsgUnitSize
: Input. The smallest allocatable message size. All messages are allocated in units of this size. If the size is 0, `RC_INVALID_SIZE` is returned.

MsgUnitCount
: Input. The maximum number of messages that can be allocated from this mailbox.

OptionWord
: Input. A bit field to describe the options to be used to open the mailbox. The following constants should be ORed together to build the appropriate set of options:

- Search options for finding a mailbox

  `MBX_OPEN_SEARCH_GLOBAL`
  Other peer units are searched if the mailbox does not exist on this unit.

  `MBX_OPEN_SEARCH_LOCAL`
  Search only this unit.

- Type of memory access for storage area. The caller can OR the following constants together to specify both types of access to the memory. The default is that neither access type is given for a local mailbox and that system bus access is given for a global mailbox.

  `MBX_MEM_MC_ACCESS`
  For system bus access rights to the memory.

  `MBX_MEM_AIB_ACCESS`
  For daughter card access rights to the memory.

MbxHandle
Output. Mailbox handle returned to requesting process. This handle is passed to all other mailbox services when this mailbox is referred to.

MbxType Output. Type of mailbox that was opened. The MbxStatus field can return the following values:

`MBX_TYPE_LOCAL`
The mailbox is on this unit and does not accept messages from other units.

`MBX_TYPE_GLOBAL`
The mailbox is on this unit and accepts messages from other units.

`MBX_TYPE_REMOTE`
The mailbox was created on another unit.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                    RC_NO_MORE_RES_ON_REMOTE
RC_INVALID_RESERVED_PARM      RC_NO_MORE_REM_MBX
RC_INVALID_NAME               RC_NO_MORE_QUEUES
RC_NAME_NOT_FOUND             RC_REMOTE_CFG_NOT_EST
RC_INVALID_OPTION             RC_INVALID_MEM_ACCESS
RC_NO_MORE_RES                RC_INVALID_CALL
RC_DUP_RES_NAME               RC_INVALID_SIZE
RC_NO_MORE_MEM                RC_INVALID_COUNT
```

Remarks

If the memory name provided by the process is the same as that passed to CreateMbx, the service does not create a new memory pool; it gives the process access to the memory pool already created. If the memory name is not the same, this service allocates the memory requested by the process. This memory is used to send messages by this process and a copy operation is performed by SendMbx.

If the process is sharing the memory, the MsgUnitSize and MsgUnitCount parameters must be less than or equal to the values specified when the memory was created.

If messages are being sent to other units, the kernel ensures that system bus access has been enabled on the mailbox pool.

Mailbox memory areas are allocated from packet memory. If there is not enough packet memory to allocate the buffer, the service returns a `RC_NO_MORE_MEM` error.

GetMbxBuffer—Get a Free Mailbox Buffer

This service allocates a free mailbox buffer to the requesting process.

Functional Prototype

```
RIC_ULONG GetMbxBuffer (RIC_MBXHANDLE   MbxHandle,
                        RIC_ULONG       Size,
                        void          **MsgPtr,
                        RIC_ULONG       Reserved);
```

Parameters

MbxHandle Input. Handle of mailbox from which the process wants to get a message buffer.

Size Input. Message size specified in bytes. The size is rounded up to a multiple of the message unit size set by CreateMbx or OpenMbx. A value of 0 is invalid.

The maximum size allowed with a single call is 65535 times the size of the message unit.

MsgPtr Output. Pointer to allocated mailbox buffer.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_HANDLE
RC_NO_MBX_BUFFER
RC_NO_MBX_RECEIVER
RC_INVALID_SIZE
RC_INVALID_MEM_ACCESS
```

Remarks

No more than 65535 times the message size in bytes can be allocated with a single call to GetMbxBuffer.

FreeMbxBuffer—Free Mailbox Buffer

This service frees a previously allocated mailbox buffer.

Functional Prototype

```
RIC_ULONG FreeMbxBuffer (RIC_MBXHANDLE  MbxHandle,
                         void          *MsgPtr,
                         RIC_ULONG      Reserved);
```

Parameters

MbxHandle Input. Handle of mailbox where the process wants to free a message buffer.

MsgPtr Input. Pointer to allocated mailbox buffer.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_HANDLE
RC_INVALID_MBX_BUFFER_ADDR
RC_INVALID_MEM_ACCESS
RC_MBX_BUFFER_IN_QUEUE
```

Remarks

None

SendMbx—Send a Message

This service puts a message into a mailbox.

Functional Prototype

```
RIC_ULONG SendMbx (RIC_MBXHANDLE  MbxHandle,
                   void          *MsgPtr,
                   RIC_ULONG      Size,
                   RIC_ULONG      OptionWord,
                   RIC_ULONG      Reserved);
```

Parameters

MbxHandle Input. Handle of the mailbox where the process sends the message.

MsgPtr Input. Pointer to the message to be sent. When the `MBX_SEND_FREE_BUFFER` option is specified, MsgPtr must point to the start of the message buffer. Otherwise, it may point to any location contained in the message buffer.

Size Input. Size of the message buffer. A message size of 0 is invalid.

OptionWord

Input. Bit field to describe how to send the message. To build the appropriate set of options, OR the following constants.

`MBX_SEND_COPY`
: Forces a copy of the message in the mailbox memory. This option applies only when sender and receiver are sharing memory. The default is `MBX_SEND_NO_COPY`.

`MBX_SEND_FREE_BUFFER`
: Returns the buffer to the free pool. The default is `MBX_SEND_KEEP_BUFFER`, which means the buffer must be freed explicitly with the FreeMbxBuffer service.

`MBX_SEND_LIFO`
: Puts a message in the front of the message queue. The default is `MBX_SEND_FIFO`, which means that the message is put at the end of the message queue.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                     RC_NO_RCV_BUFFER
RC_INVALID_RESERVED_PARM       RC_INVALID_CALL
RC_INVALID_HANDLE              RC_INVALID_OPTION
RC_INVALID_SIZE                RC_INVALID_MEM_ACCESS
RC_NO_MBX_RECEIVER             RC_UNABLE_TO_ACCESS_UNIT
RC_MSG_BUFFER_NOT_FREED        RC_PIPES_NOT_CONFIGURED
RC_INVALID_MBX_BUFFER_ADDR     RC_MBX_BUFFER_IN_QUEUE
```

Remarks

The semaphore associated with this mailbox is incremented by 1.

The MBX_SEND_COPY option is valid only if the sender and the receiver are sharing memory. It can be used with shared memory to keep the message around for further processing. If the sender and the receiver are not sharing memory, the value of the MBX_SEND_COPY bit is ignored and the message is copied automatically to the receive memory.

The MBX_SEND_FREE_BUFFER option is ignored if the sender and receiver are sharing memory and the MBX_SEND_COPY option was not requested. The call returns the RC_MSG_BUFFER_NOT_FREED return code after sending the message.

If MBX_SEND_FREE_BUFFER is specified and the SendMbx service fails, the buffer is not freed. It must be explicitly freed by the sender using FreeMbxBuffer.

If messages are being sent to other units, the kernel ensures system bus access has been enabled on the mailbox pool.

ReceiveMbx—Receive a Message

This service reads or receives a message from a mailbox.

Functional Prototype

```
RIC_ULONG ReceiveMbx (RIC_MBXHANDLE    MbxHandle,
                      RIC_ULONG        OptionWord,
                      RIC_TIMEOUT      Timeout,
                      void           **MsgPtr,
                      RIC_ULONG       *Size,
                      RIC_ULONG        Reserved);
```

Parameters

MbxHandle Input. Handle of the mailbox from which the process wants to receive a message.

OptionWord

Input. Option word for specifying receive options. The following constant can be used:

MBX_RECEIVE_READ_MESSAGE
: Return a pointer to the message but do not remove the message from the mailbox.

MBX_RECEIVE_GET_MESSAGE
: Returns a pointer to the message and removes the message from the mailbox. This is the default.

Timeout Input. Optional timeout for waiting on semaphore associated with this mailbox.

–1 There is no timeout.
0 Return immediately if there are no mailbox elements.
Any other value from 1 to 65535
: Wait time in milliseconds. The granularity of the timer is five milliseconds. The timeout value is rounded up to the next multiple of five, if it is not already a multiple of five.

MsgPtr Output. Pointer to the received message buffer.

Size Output. Size of the received message buffer.

Reserved Input. Reserved parameter (must be 0).

Returns

RC_SUCCESS	RC_INVALID_OPTION
RC_INVALID_RESERVED_PARM	RC_INVALID_MEM_ACCESS
RC_INVALID_HANDLE	RC_INVALID_CALL
RC_INVALID_RECEIVER	RC_INVALID_TIMEOUT
RC_MBX_EMPTY	

Remarks

If the `MBX_RECEIVE_READ_MESSAGE` option is set in OptionWord, the message is not dequeued from the message queue.

If the `MBX_RECEIVE_READ_MESSAGE` option is not set in OptionWord, this service removes the first message from the queue, and the semaphore associated with the mailbox is decremented.

CloseMbx—Close a Mailbox

This service releases the mailbox and deletes it if no other process has access to it.

Functional Prototype

```
RIC_ULONG CloseMbx (RIC_MBXHANDLE MbxHandle,
                    RIC_ULONG     Reserved);
```

Parameters

MbxHandle Input. Handle of mailbox to close.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_HANDLE
RC_DEPENDENT_EVENTS
RC_INVALID_CALL
```

Remarks

If the close is issued by a process while other processes still have access to the mailbox, the service removes access rights for the calling process.

Any memory pool associated with the mailbox for sending by this process is released.

When the last process closes the mailbox, the mailbox is deleted. When the creator closes the mailbox, the semaphore associated with the mailbox is closed, and the memory used by the mailbox for receiving data is closed.

CreateSig—Create a Signal

This service creates a signal and optionally registers a signal handler.

Functional Prototype

```
RIC_ULONG CreateSig (char            *SigName,
                     RIC_SIGHANDLER  EntryPoint,
                     RIC_ULONG       OptionWord,
                     RIC_ULONG       SigHanID,
                     RIC_SIGHANDLE  *SigHandle,
                     RIC_ULONG       Reserved);
```

Parameters

SigName Input. Name to assign to signal so that other processes can access it. This parameter also can be NULL. However, if it is NULL, only the creating process can use the signal.

EntryPoint Input. Entry address on which user gets control on calling of the signal. If this parameter is NULL, the calling process does not get control through this signal. It gets only a handle back in SigHandle to use in calling the signal.

OptionWord

Input. Describes how to receive a signal. This parameter is valid only if the EntryPoint parameter is not NULL.

`SIG_CONTROL_ALWAYS`
Calling process wants control any time the signal is called.

`SIG_CONTROL_MATCH`
Calling process wants control only when the signal is called with a matching SigHanID.

SigHanID Input. This parameter is valid only if OptionWord is `SIG_CONTROL_MATCH`. The caller gets control only when the signal is called with a matching SigHanID.

SigHandle Output. Signal handle returned to requesting process. This handle is used to call the signal.

Reserved Input. Reserved parameter (must be 0).

Returns

RC_SUCCESS	RC_DUP_RES_NAME
RC_INVALID_RESERVED_PARM	RC_INVALID_MEM_ACCESS
RC_NO_MORE_RES	RC_INVALID_CALL
RC_INVALID_NAME	RC_INVALID_OPTION

Remarks

Processes that open a signal with a NULL EntryPoint are *calling* processes. Processes that open a signal with a non-NULL EntryPoint are *receiving* processes.

The EntryPoint for a receiving process should be a handler that accepts three parameters:

- SigHanID
- A pointer to a parameter block
- A parameter block size

It should also return a flag as the function value indicating what the kernel should do next.

0	Indicates that the kernel should call the rest of the receiving processes in the chain.
1	Indicates that the kernel should give control back to the calling process immediately.

For normal processes, when the handler is called, memory protection is turned on if global memory protection is enabled. For device drivers and subsystems, the state of memory protection depends on the OptionWord specified in CreateDev.

A signal can have multiple receiving processes. Each can be distinguished with the SigHanID. Calling processes can also be receiving processes for the same signal.

OpenSig—Open a Signal

This service opens a signal and optionally registers a signal handler.

Functional Prototype

```
RIC_ULONG OpenSig (char           *SigName,
                   SIGHANLDER      EntryPoint,
                   RIC_ULONG       OptionWord,
                   RIC_ULONG       SigHanID,
                   RIC_SIGHANDLE  *SigHandle,
                   RIC_ULONG       Reserved);
```

Parameters

SigName Input. Name of signal to access.

EntryPoint Input. Entry address on which user gets control on calling of the signal. If this parameter is NULL, the calling process does not get control through this signal. It gets only a handle back in SigHandle, which it can use in calling the signal.

OptionWord

Input. Describes how to receive a signal. This parameter is valid only if the EntryPoint parameter is not NULL.

`SIG_CONTROL_ALWAYS`
Calling process wants control any time the signal is called.

`SIG_CONTROL_MATCH`
Calling process wants control only when the signal is called with a matching SigHanID.

SigHanID Input. This parameter is valid only if OptionWord is `SIG_CONTROL_MATCH`. The caller gets control only when the signal is called with a matching SigHanID. The SigHanID cannot have a value of 0 because it is used for broadcasts.

SigHandle Output. Handle for the signal requested by the process. This handle is used to call the signal.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                  RC_INVALID_NAME
RC_INVALID_RESERVED_PARM    RC_INVALID_MEM_ACCESS
RC_NO_MORE_RES              RC_INVALID_CALL
RC_NAME_NOT_FOUND           RC_INVALID_OPTION
```

Remarks

Processes that open a signal with a NULL EntryPoint are *calling processes*. Processes that create a signal using a non-NULL EntryPoint are *receiving processes*.

The EntryPoint for a receiving process should be a handler that accepts three parameters:

- SigHanID
- A pointer to a parameter block
- A parameter block size

It should also return a flag as the function value indicating what the kernel should do next.

0 The kernel should call the rest of the receiving processes in the chain.

1 The kernel should give control back to the calling process immediately.

For normal processes, when the handler is called, memory protection is turned on if global memory protection is enabled. For device drivers and subsystems, the state of memory protection depends on the OptionWord specified in CreateDev.

CloseSig—Close a Signal

This service releases access to a signal and deletes the signal if no other processes have access.

Functional Prototype

```
RIC_ULONG CloseSig (RIC_SIGHANDLE SigHandle,
                    RIC_ULONG     Reserved);
```

Parameters

SigHandle Input. Signal handle of signal to release.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_HANDLE
RC_INVALID_RESERVED_PARM
RC_INVALID_CALL
```

Remarks

If a process attempts to close a signal while other processes still have access to the signal, the service removes access rights for the process issuing the call. When the last process with access rights calls this service, the signal ceases to exist.

InvokeSig—Call a Signal

This service calls a signal.

Functional Prototype

```
RIC_ULONG InvokeSig (RIC_SIGHANDLE  SigHandle,
                     RIC_ULONG      SigHanID,
                     void          *Parms,
                     RIC_ULONG      ParmLen,
                     RIC_ULONG      Reserved);
```

Parameters

SigHandle Input. Handle of signal returned from CreateSig or OpenSig.

SigHanID Input. A value of 0 is interpreted as a broadcast. Every receiving process gets control unconditionally. Any other value is interpreted as a conditional call. Only receiving processes that have a matching SigHanID or that set their Always flag get control.

Parms Input. Pointer to parameters to pass to receiving processes.

ParmLen Input. Size of parameters to pass to receiving processes.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_CALL_TERMINATED
RC_INVALID_RESERVED_PARM
RC_INVALID_HANDLE
RC_NO_SUCH_SIG_ID
RC_INVALID_MEM_ACCESS
```

Remarks

Before passing control to a receiving process, the kernel changes the memory protection to allow the receiving process to access the parameter block as well as its own memory, code, data, and stack. This is also true for a subsystem or device driver with memory protection turned on. The `OPTION_PROT_OFF` option in the OptionWord parameter of CreateDev is used to determine if memory protection is enabled for signals received by a device driver or subsystem.

Device Driver/Subsystem Services

The following are the device driver/subsystem services.

Refer to the *ARTIC960 Programmer's Guide* for additional information.

CreateDev—Register a Subsystem or Device Driver

This service registers the process as a subsystem or device driver.

Functional Prototype

```
RIC_ULONG CreateDev (char           *DDName,
                     RIC_DOHANDLER   OpenEntry,
                     RIC_DCHANDLER   CloseEntry,
                     RIC_DIHANDLER   InvokeEntry,
                     RIC_ULONG       OptionWord,
                     RIC_DEVHANDLE  *DDHandle,
                     RIC_ULONG       Reserved);
```

Parameters

DDName Input. A device name to assign to this subsystem or device driver so that other processes can access this subsystem by name. The kernel's subsystems allocate all resources with the first four characters being "RIC_" for the resource name. User device driver and subsystem names should not start with this prefix.

OpenEntry Input. Address of open entry point of subsystem or device driver. It gets control on this entry point when an application uses OpenDev. See *OpenEntry Prototype* on page 123.

CloseEntry Input. Address of close entry point of subsystem or device driver. It gets control on this entry point when an application uses CloseDev. See *CloseEntry Prototype* on page 124.

InvokeEntry Input. Address of strategy entry point of subsystem or device driver. It gets control on this entry point when an application uses InvokeDev. See *InvokeEntry Prototype* on page 124.

OptionWord

Input. Bit field that gives various create options. These constants may be ORed together to create the device driver options.

`OPTION_DEV_DRV`
: Registers the process as a device driver

`OPTION_SUB_SYS`
: Registers the process as a subsystem.

`OPTION_PROT_ON`
: Turns on memory protection before the kernel gives control to the process at one of its entry points. This constant does not apply to vectors owned by the subsystem or device driver.

`OPTION_PROT_OFF`
: Turns off memory protection. This constant does not apply to vectors owned by the subsystem or device driver.

DDHandle Output. Device handle returned to the requesting process.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                    RC_NO_MORE_RES
RC_INVALID_RESERVED_PARM      RC_INVALID_MEM_ACCESS
RC_INVALID_NAME               RC_INVALID_CALL
RC_DUP_RES_NAME               RC_INVALID_OPTION
```

Remarks

An application process communicates with a subsystem or device driver using OpenDev, CloseDev, and InvokeDev.

If memory protection is enabled using the `OPTION_PROT_ON` in OptionWord, the kernel keeps the memory protection enabled and maps the subsystem or device driver's code, data, and application-passed parameters before giving control to the subsystem or device driver.

Memory protection for subsystem or device driver is expected only during early development. Because subsystem or device driver code is more *trusted* and performance will improve, it is expected that each subsystem or device driver will run with memory protection disabled in production systems.

The following are examples of function prototypes for OpenEntry, CloseEntry, and InvokeEntry that must be followed when writing a device driver or subsystem.

OpenEntry Prototype

The function prototype for OpenEntry must be:

```
RIC_ULONG Open_Name  (void          *DDParams,
                      RIC_ULONG      Size,
                      RIC_PROCESSID  ProcessID,
                      RIC_ULONG     *DevMemo);
                      RIC_ULONG     *DevMemo);
```

Parameters

DDParams Input. Address of subsystem or device driver defined parameters.

Size Input. Size of subsystem or device driver defined parameters. The size of the buffer pointed to by DDParams.

ProcessID Input. The ProcessID of the process in execution.

DevMemo Output. Device memo returned to the kernel from the driver or subsystem.

Returns

Must return `RC_SUCCESS` if it is successful or a non-zero value (between `0xFFFF0000` and `0xFFFFFFFF`) if it fails.

CloseEntry Prototype

The function prototype for CloseEntry must be:

```
RIC_ULONG Close_Name   (RIC_PROCESSID  ProcessID,
                        RIC_ULONG      DevMemo);
```

Parameters

ProcessID Input. The ProcessID of the process in execution.

DevMemo Input. Device-memo value previously provided by the subsystem.

Returns

Must return `RC_SUCCESS` if it is successful or a non-zero value (between `0xFFFF0000` and `0xFFFFFFFF`) if it fails.

InvokeEntry Prototype

The function prototype for InvokeEntry must be:

```
RIC_ULONG Invoke_Name (void           *DDParams,
                       RIC_ULONG      Size,
                       RIC_PROCESSID  ProcessID,
                       RIC_ULONG      DevMemo);
```

Parameters

DDParams Input. Address of subsystem or device driver defined parameters.

Size Input. Size of subsystem or device driver defined parameters. The size of the buffer pointed to by DDParams.

ProcessID Input. The ProcessID of the process in execution.

DevMemo Input. Device-memo value previously provided by the subsystem.

Returns

Must return `RC_SUCCESS` if it is successful or a non-zero value (between `0xFFFF0000` and `0xFFFFFFFF`) if it fails.

OpenDev—Open a Subsystem or Device Driver

This service opens a previously registered subsystem or device driver.

Functional Prototype

```
RIC_ULONG OpenDev (char          *DDName,
                   void          *DDParams,
                   RIC_ULONG      Size,
                   RIC_DEVHANDLE *DDHandle,
                   RIC_ULONG      Reserved);
```

Parameters

DDName	Input. A device name used to create the subsystem or device driver.
DDParams	Input. Address of subsystem or device driver defined parameters.
Size	Input. Size of subsystem or device driver defined parameters. The size of the buffer pointed to by DDParams.
DDHandle	Output. Device handle returned to the requesting process. This handle is passed to all other services related to subsystem or device driver.
Reserved	Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_CALL_TERMINATED
RC_INVALID_RESERVED_PARM
RC_INVALID_NAME
RC_NAME_NOT_FOUND
RC_NO_MORE_RES
RC_INVALID_MEM_ACCESS
RC_INVALID_CALL
```

Remarks

This service gets access to the already registered subsystem or device driver.

The kernel gives control to subsystem or device driver on its OpenEntry entry point with the parameters specified on page 123. The subsystem or device driver can return a 32-bit device memo to the kernel on the exit from its OpenEntry function. The kernel passes this memo back to the subsystem or device driver on any call for this process.

Multiple opens of a device driver are allowed, but the number of closes by a single process should match the number of opens by that process.

Return codes returned by the OpenEntry function of a subsystem or device driver are passed back to the calling process as the return code of OpenDev. These return codes must be either `RC_SUCCESS` or within the range `0xFFFF0000` to `0xFFFFFFFF`. Return codes outside this range are discarded. The return code from OpenEntry is used by the kernel to remove access to the device driver if the return code is not `RC_SUCCESS`.

CloseDev—Close a Subsystem or Device Driver

This service releases the access of this process to the subsystem or device driver. It also *deregisters* a device driver or subsystem.

Functional Prototype

```
RIC_ULONG CloseDev (RIC_DEVHANDLE  DDHandle,
                    RIC_ULONG      Reserved);
```

Parameters

DDHandle Input. Handle of subsystem or device driver to close.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_CALL_TERMINATED
RC_INVALID_RESERVED_PARM
RC_INVALID_HANDLE
RC_INVALID_MEM_ACCESS
RC_INVALID_CALL
```

Remarks

When this service is issued by a process that had previously issued an OpenDev for the subsystem or device driver, the kernel gives control to the subsystem or device driver at the CloseEntry entry point with the parameters specified on page 124.

If this service is called by the subsystem or device driver itself (using the handle received from CreateDev), access to the subsystem or device driver is removed and all other processes having access to this subsystem or device driver are notified through an asynchronous-event notification. In this case, the CloseEntry function is not called.

Return codes returned by the CloseEntry point of a subsystem or device driver are passed back to the calling process as the return code of CloseDev. These return codes must be either `RC_SUCCESS` or in the range `0xFFFF0000` to `0xFFFFFFFF`. Return codes outside this range are discarded.

The kernel removes the access of the process to the subsystem or device driver, even if CloseEntry failed.

InvokeDev—Call a Subsystem or Device Driver

This service calls the subsystem or device driver on its strategy entry point.

Functional Prototype

```
RIC_ULONG InvokeDev (RIC_DEVHANDLE  DDHandle,
                     void          *DDParams,
                     RIC_ULONG      Size,
                     RIC_ULONG      Reserved);
```

Parameters

DDHandle	Input. Handle of subsystem or device driver to call.
DDParams	Input. Address of subsystem or device driver defined parameters.
Size	Input. Size of subsystem or device driver defined parameters. The size of the buffer pointed to by DDParams.
Reserved	Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_CALL_TERMINATED
RC_INVALID_RESERVED_PARM
RC_INVALID_HANDLE
RC_INVALID_MEM_ACCESS
RC_INVOKE_ENTRY_FAILURE
RC_DD_RC_OUT_OF_RANGE
```

Remarks

The kernel gives control to subsystem or device driver at the InvokeEntry entry point with the parameters specified in the *InvokeEntry Prototype* on page 124 with driver memo (returned by the subsystem or device driver on call of OpenEntry by kernel) as parameters.

If the device driver or subsystem has specified memory protection be disabled, it is disabled when its call handler gets control. If the device driver or subsystem requested that memory protection be enabled, the device driver or subsystem will have access to the call parameter block, as well as its own code, data, stack, allocated memory, and so forth.

Return codes returned by the InvokeEntry function are passed back to the calling process as the return code of InvokeDev . These return codes must be either `RC_SUCCESS`, `RC_INVOKE_ENTRY_FAILURE`, or in the range `0xFFFF0000` to `0xFFFFFFFF`. Return codes not in this range are discarded and the `RC_DD_RC_OUT_OF_RANGE` error is returned.

AllocVector—Allocate an Interrupt Vector

This service allocates an interrupt vector to the calling subsystem or device driver.

Functional Prototype

```
RIC_ULONG AllocVector (RIC_ULONG      VectorNum,
                       RIC_VECTOR     EntryPoint,
                       RIC_ULONG      OptionWord,
                       RIC_ULONG      Reserved);
```

Parameters

VectorNum Input. The interrupt vector number to be allocated.

EntryPoint Input. Pointer to the interrupt-handling routine for the requested interrupt vector.

OptionWord Input.

`OPTION_PROT_ON`
The kernel enables memory protection before passing control to the EntryPoint.

`OPTION_PROT_OFF`
The kernel does not enable memory protection.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_VECTOR
RC_INVALID_OPTION
RC_NO_MORE_RES
RC_VECTOR_NOT_AVAILABLE
RC_NOT_DD_OR_SS
RC_INVALID_MEM_ACCESS
RC_INVALID_CALL
```

Remarks

The kernel allocates the requested vector to the calling process as non-shared. If the vector was previously allocated as non-shared, `RC_VECTOR_NOT_AVAILABLE` is returned. Refer to the *ARTIC960 Programmer's Guide* for information about vector sharing.

Memory protection for an interrupt handler is disabled when global memory protection is disabled, regardless of the state of the OptionWord.

The calling process must be a device driver or subsystem.

AllocVectorMux—Allocate an Interrupt Vector

This service allocates an interrupt vector to the calling subsystem or device driver

Functional Prototype

```
RIC_ULONG AllocVectorMux (RIC_ULONG       VectorNum,
                          RIC_VECTOR_MUX  EntryPoint,
                          RIC_ULONG       OptionWord,
                          RIC_ULONG       Reserved);
```

Parameters

VectorNum Input. The interrupt vector number to be allocated.

EntryPoint Input. Pointer to the interrupt-handling routine for the requested interrupt vector. The interrupt-handling routine must return a value of 0 if the interrupt was not claimed or a non-zero value if the interrupt was claimed.

OptionWord

Input. Bit field to describe options. Use the OR operation on the following constants to build the appropriate set of options:

`OPTION_PROT_ON`
The kernel enables memory protection prior to passing control to the EntryPoint.

`OPTION_PROT_OFF`
The kernel does not enable memory protection.

`OPTION_VECTOR_SHARED`
Allocates the vector as shared.

`OPTION_VECTOR_NOT_SHARED`
Allocates the vector as nonshared. This is the default.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                    RC_VECTOR_NOT_AVAILABLE
RC_INVALID_RESERVED_PARM      RC_NOT_DD_OR_SS
RC_INVALID_VECTOR             RC_INVALID_MEM_ACCESS
RC_INVALID_OPTION             RC_INVALID_CALL
RC_NO_MORE_RES
```

Remarks

The kernel allocates the requested vector to the calling process. If `OPTION_VECTOR_NOT_SHARED` is requested and the vector was previously allocated as nonshared, `RC_VECTOR_NOT_AVAILABLE` is returned. Refer to the *ARTIC960 Programmer's Guide* for information about vector sharing.

Memory protection for an interrupt handler is disabled when global memory protection is disabled, regardless of the state of the OptionWord.

The calling process must be a device driver or subsystem.

A process may not allocate the same vector multiple times.

TheSetInterruptPriority macro can be used from within an interrupt handler to set a new interrupt priority level for the processor. This macro gives an interrupt handler the ability to lower its priority and allow other interrupts at the same level or lower levels to be serviced.

The macro is defined as follows:

```
#define SetInterruptPriority(priority, 0)
```

Valid priority values are 0 to 30. In addition, a priority value of `0xFFFFFFFF` sets the new priority level to the current priority level minus 1. A priority value of 0 can be used to get off of interrupt priority, but remain within the interrupt context.

The caller must clear the interrupt source before lowering the interrupt priority.

EntryPoint Prototype

The function prototype for the EntryPoint must be:

```
RIC_ULONG EntryPoint (RIC_ULONG VectorNum);
```

Returns

Must return 0 if the interrupt was not claimed or non-zero if the interrupt was claimed.

SetVector—Set a New Interrupt Vector Entry Point

This service sets a new entry point for a previously allocated interrupt vector.

Functional Prototype

```
RIC_ULONG SetVector   (RIC_ULONG      VectorNum,
                       RIC_VECTOR     EntryPoint,
                       RIC_ULONG      OptionWord,
                       RIC_ULONG      Reserved);
```

Parameters

VectorNum Input. The interrupt vector number whose entry address is to be changed.

EntryPoint Input. Pointer to the interrupt-handling routine for the interrupt vector.

OptionWord

Input.

`OPTION_PROT_ON`
: Causes the kernel to enable memory protection prior to passing control to the EntryPoint.

`OPTION_PROT_OFF`
: Does not enable memory protection.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_VECTOR
RC_INVALID_CALL
RC_INVALID_MEM_ACCESS
RC_INVALID_OPTION
RC_VECTOR_NOT_ALLOCATED
```

Remarks

The application must allocate the vector before calling this service.

SetVectorMux—Set an Interrupt Vector

This service sets a new entry point for a previously-allocated interrupt vector.

Functional Prototype

```
RIC_ULONG SetVectorMux (RIC_ULONG      VectorNum,
                        RIC_VECTOR_MUX EntryPoint,
                        RIC_ULONG      OptionWord,
                        RIC_ULONG      Reserved);
```

Parameters

VectorNum Input. The interrupt vector number whose entry address is to be changed.

EntryPoint Input. Pointer to the interrupt-handling routine for the shared interrupt vector. The interrupt-handling routine must return a value of 0 if the interrupt was not claimed or a non-zero value if the interrupt was claimed.

See *EntryPoint Prototype* on page 132.

OptionWord

Input.

`OPTION_PROT_ON`
The kernel enables memory protection prior to passing control to the EntryPoint.

`OPTION_PROT_OFF`
The kernel does not enable memory protection.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_VECTOR
RC_INVALID_CALL
RC_INVALID_MEM_ACCESS
RC_INVALID_OPTION
RC_VECTOR_NOT_ALLOCATED
```

Remarks

The calling process must have allocated the vector before calling this service.

EntryPoint Prototype

The function prototype for the EntryPoint must be:

```
RIC_ULONG EntryPoint (RIC_ULONG VectorNum);
```

Returns

Must return 0 if the interrupt was not claimed or non-zero if the interrupt was claimed.

ReturnVector—Return an Interrupt Vector

This service returns a previously allocated interrupt vector.

Functional Prototype

```
RIC_ULONG ReturnVector (RIC_ULONG     VectorNum,
                        RIC_ULONG     Reserved);
```

Parameters

VectorNum Input. Vector number of vector returned by this service.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_VECTOR_NOT_ALLOCATED
RC_INVALID_VECTOR
RC_INVALID_CALL
```

Remarks

None

AllocHW—Allocate a Hardware Device

This service allocates a hardware device to the calling subsystem or device driver.

Functional Prototype

```
RIC_ULONG AllocHW (char            *DeviceName,
                   RIC_ULONG        BufferSize,
                   RIC_ULONG       *POSTStatus,
                   unsigned char   *DeviceDataPtr,
                   RIC_ULONG        Reserved);
```

Parameters

DeviceName
: Input. Name of the hardware device requested by this call. This name is predefined for each type of device.

BufferSize
: Input. Size of the buffer pointed to by DeviceDataPtr. The kernel copies device-related data to this buffer. If the buffer is too small, the kernel copies BufferSize amount of data into the buffer and returns an error.

POSTStatus
: Output. A zero in this field indicates power-on self test (POST) code for this device completed successfully. A non-zero value is device specific but indicates that some form of error occurred during POST.

DeviceDataPtr
: Output. Pointer to a buffer to which the kernel copies device-dependent data (see *Device-Dependent Data* on page 135 for more information).

Reserved
: Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                     RC_BUFFER_TOO_SMALL
RC_INVALID_RESERVED_PARM       RC_INVALID_NAME
RC_NO_MORE_RES                 RC_NOT_DD_OR_SS
RC_HW_ALREADY_ALLOCATED        RC_INVALID_CALL
RC_NAME_NOT_FOUND              RC_INVALID_MEM_ACCESS
```

Remarks

The kernel allocates the requested hardware device to the calling process, if available. For device names, refer to the documents for the applicable daughter card. (For example, for the 4-Port Multi-Interface Application Interface Board, see the related chapter in the *ARTIC960 Co-Processor Platforms: Hardware Technical Reference*.)

Device-Dependent Data

When the adapter is powered on or reset, POST code on the adapter or daughter card updates the Resource Descriptor Table (RDT) with device information. The kernel returns this device information on this call. The following is the structure of the Resource Descriptor Table.

```
struct RIC_RDTEntry
{
   char           DeviceName[DEVICE_NAME_SIZE];
   RIC_PROCESSID  ProcessID;
   RIC_ULONG      PostStatus;
   RIC_ULONG      DataSize;
   unsigned char  DeviceData[MAX_DATA_SIZE];
}
```

ReturnHW—Return a Hardware Device

This service returns a previously-allocated hardware device.

Functional Prototype

```
RIC_ULONG ReturnHW (char        *DeviceName,
                    RIC_ULONG    Reserved);
```

Parameters

DeviceName
Input. Name of the hardware device to return.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_HW_NOT_FOUND
RC_HW_NOT_ALLOCATED
RC_INVALID_CALL
RC_INVALID_NAME
RC_INVALID_MEM_ACCESS
```

Remarks

None

QueryHW—Query Status of Hardware Device

This service returns the status of a hardware device to the calling subsystem or device driver.

Functional Prototype

```
RIC_ULONG QueryHW (char            *DeviceName,
                   RIC_ULONG        BufferSize,
                   RIC_ULONG       *Status,
                   RIC_ULONG       *POSTStatus,
                   unsigned char   *DeviceDataPtr,
                   RIC_ULONG        Reserved);
```

Parameters

DeviceName Input. Name of the hardware device requested by this service. This name is predefined for each type of device.

BufferSize Input. Size of the buffer pointed to by DeviceDataPtr. The kernel copies device-related data to this buffer. If the buffer is too small, the kernel copies BufferSize amounts of data into the buffer and returns an error.

Status Output. If the return code is RC_SUCCESS, this field is set to indicate the allocation status of the hardware device.

HW_AVAILABLE	Resource is available.
HW_NOT_AVAILABLE	Resource is not available.

POSTStatus Output. A zero in this field indicates this device completed POST successfully. A non-zero value is device specific but indicates that an error occurred during POST.

DeviceDataPtr

Output. Pointer to a buffer to which the kernel copies device-dependent data (see *Device-Dependent Data* on page 135 for more information).

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_NOT_DD_OR_SS
RC_NAME_NOT_FOUND
RC_INVALID_CALL
RC_INVALID_MEM_ACCESS
RC_INVALID_NAME
RC_BUFFER_TOO_SMALL
```

Remarks

None

Asynchronous Event Notification Services

The following are the asynchronous event notification services.

Refer to the *ARTIC960 Programmer's Guide* for additional information.

RegisterAsyncHandler—Register an Async Handler

This service registers the asynchronous event notification handler of a process for specified events.

Functional Prototype

```
RIC_ULONG RegisterAsyncHandler (RIC_ULONG         SoftwareEvents,
                                RIC_ULONG         AdapterEvents,
                                RIC_ULONG         ProcessorEvents,
                                RIC_ASYNCHANDLER  AsyncHandler,
                                RIC_ULONG         Reserved);
```

Parameters

SoftwareEvents

Input. Mask specifying of which software events the process wants to be notified. The software event mask is built by ORing the following event flags together. The event flags can be used to build the software event mask.

AEN_DEV_TERM	Device driver or subsystem termination
AEN_PROCESS_START	Process start
AEN_PROCESS_STOP	Process stop
AEN_SHARED_RESOURCE	Closing a shared resource

AdapterEvents

Input. Mask specifying of which adapter events the process wants to be notified. You can OR the following event flags together to form the adapter event mask.

AEN_WATCHDOG	Watchdog timer expiration
AEN_PARITY	Parity error • Multiple-bit ECC error • AIB bus read parity error with 80960 master • Local bus parity for: – RadiSys ARTIC 32-bit Memory Controller Chip – System bus Interface Chip – CFE Local Bus/AIB Interface Chip
AEN_MEM_PROCESSOR	Memory-protection violation (80960 processor)
AEN_MEM_MICROCHANNEL	Memory-protection violation (system bus master)
AEN_MEM_AIB	Memory-protection violation (AIB master)
AEN_MEM_VIOLATION	Non-existent memory access by the 80960
AEN_PCI_ERROR	PCI bus error

ProcessorEvents
Input. Mask specifying of which processor events the process wants to be notified. You can OR the following event flags together to form the processor event mask.

`AEN_80960_ARITHMETIC`	Arithmetic
`AEN_80960_CONSTRAINT`	Constraint
`AEN_80960_OPERATION`	Operation
`AEN_80960_PROTECTION`	Protection
`AEN_80960_TRACE`	Trace
`AEN_80960_TYPE`	Type

AsyncHandler
Input. Address of user-defined asynchronous-event notification handler. This handler is called when any of the events specified in the masks occur. (See *Asynchronous Event Notification Handler* on page 141.)

Reserved Input. Reserved parameter (must be 0).

Returns

`RC_SUCCESS`	`RC_NO_MORE_RES`
`RC_DUP_ASYNC_EVENT`	`RC_INVALID_MEM_ACCESS`
`RC_INVALID_RESERVED_PARM`	`RC_INVALID_CALL`

Remarks

See the Intel *80960CA User's Manual* for more information on processor faults. Parallel faults are not reported to users directly. Instead, the processes are notified separately of each fault that is part of the parallel fault.

If a process calls this service more than once, the process is notified of all events requested in all calls. However, the process cannot register for a particular event again unless it first deregisters for that event. If the process issues a call to register for an event again, the process is not registered for any of the events specified in the call.

If the calling process is a device driver or subsystem, the asynchronous event notification handler is called with the memory protection specified when the calling process issued CreateDev. Otherwise, the handler is called with the memory protection set in accordance with global memory protection. Refer to the *ARTIC960 Programmer's Guide* for additional information on the use of memory protection.

Asynchronous Event Notification Handler

The AsyncHandler should be thought of as an interrupt handler. It has access to the same subset of services as an interrupt handler. An AsyncHandler should accept one parameter (a pointer to an asynchronous event notification record) and should not return a return value. The record is defined as follows.

```
struct RIC_AsyncEvent
{
   RIC_ULONG       Class;
   RIC_ULONG       IntStat;
   RIC_PROCESSID   ProcessID;
   RIC_ULONG       Type;
   union
   {
      struct RIC_ProcessorEvent Pr;
      struct RIC_AdapterEvent   Ad;
      struct RIC_SoftwareEvent  Sw;
   }ClassInfo;
};
```

where:

Class Is the event class. Valid values are:

```
AEN_CLASS_SOFTWARE
AEN_CLASS_ADAPTER
AEN_CLASS_PROCESSOR
```

IntStat Set to 1 if fault occurred during an interrupt or a handler.

ProcessID ID of the process that caused the event.

Type Type of event within the Class. Refer to the event masks listed in sections *SoftwareEvents* and *AdapterEvents* on page 139, and *ProcessorEvents* on page 140.

Pr Information specific to processor events (see *Pr Field* on page 142).

Ad Information specific to adapter events (see *Ad Field* on page 143).

Sw Information specific to software events (see *Sw Field* on page 144).

Pr Field

The Pr field in AsyncEvent has the following definition. For maximum portability, applications should limit their accesses of this structure to the Type and CodeAddress fields. The other fields are processor-specific. All fields except the StackFrame field are defined in the Intel *80960CA User's Manual*.

```
struct RIC_ProcessorEvent
{
   RIC_ULONG   FaultType;
   RIC_ULONG   SubType;
   RIC_ULONG   CodeAddr;
   RIC_ULONG   StackFrame;
   RIC_ULONG   ProcessCtrl;
   RIC_ULONG   ArithCtrl;
   RIC_ULONG   Reserved1;
   RIC_ULONG   Reserved2;
};
```

where:

FaultType Fault type given by the processor

Subtype Fault subtype given by the processor

CodeAddr Code address of the fault (undefined for some faults)

StackFrame Pointer to the process' registers on the stack. This field is valid only for Trace faults.

ProcessCtrl Contents of the process-controls (PC) register.

ArithCtrl Contents of the arithmetic-controls (AC) register.

Reserved1, *Reserved2*
Reserved for future use.

Ad Field

The *Ad* field in *AsyncEvent* has the following definition.

```
struct RIC_AdapterEvent
{
   void       *CodeAddr;
   void       *MemAddr;
   struct RIC_PCIError PCIError;
};
```

where:

CodeAddr Code address after and near the faulting instruction.

MemAddr Memory address that the code was attempting to access. If the value is 0xFFFFFFFF, the address is unknown.

PCIError Structure of information related to the PCI bus error. The PCIError field in AsyncEvent has the following definition. For a definition of *RPInfo* and *HxInfo,* refer to the *ARTIC960 Programmer's Guide*. This information should be checked to determine the specific cause of the interrupt.

```
struct RIC_PCIError
{
  union
  {
     struct RIC_RPErrInfo RPInfo;
     struct RIC_HxErrInfo HxInfo;
  }  TermErrInfo;
RIC_ULONG   TermErrCode;
RIC_ULONG   ReturnCode;
}
```

where:

TermErrInfo

A union containing the exception data that will be posted after all asynchronous handlers have been called.

ARTIC960RxD information will be filled in the RPInfo field.

TermErrCode

The exception code to be posted (either `TERMERR_PLX_INTERRUPT` or `TERMERR_NMI_INTERRUPT`).

ReturnCode

A field that the asynchronous handler may set to 1 to force the kernel not to generate a terminal error. Otherwise, handlers should not modify this field.

Sw Field

The SW field in AsyncEvent has the following definition.

```
struct RIC_SoftwareEvent
{
   union
   {
      RIC_DEVHANDLE              DevHandle;
      struct RIC_SharedRsrcClose ShrRes;
   }SwInfo;
};
```

where:

DevHandle Device handle in the case of device driver termination.

ShrRes Structure of information related to the closing of shared resources. The ShrRes field in SoftwareEvent has the following definition.

```
struct RIC_SharedRsrcClose
{
  RIC_ULONG       ResType;
  RES_HANDLE      ResHandle;
  RIC_ULONG       OpenCount;
  RIC_ULONG       Resinfo;
};
```

where:

ResType Number indicating the type of the resource being closed

ResHandle Resource handle

OpenCount Number of processes that have the resource open

Resinfo Resource specific information:

Memory	Contains a base pointer
Mailbox	TRUE if the creator is closing
Semaphore	TRUE if the semaphore is MUTEX and the owner is closing
Events	FALSE, always
Signals	The number of receivers remaining
Queues	FALSE, always

DeregisterAsyncHandler—Deregister an Async Handler

This service deregisters the asynchronous event notification handler of a process for specified events.

Functional Prototype

```
RIC_ULONG DeregisterAsyncHandler (RIC_ULONG   SwEvents,
                                  RIC_ULONG   HwEvents,
                                  RIC_ULONG   PrEvents,
                                  RIC_ULONG   Reserved);
```

Parameters

SwEvents Input. Mask specifying of which software events the process should no longer be notified.

HwEvents Input. Mask specifying of which adapter hardware events the process should no longer be notified.

PrEvents Input. Mask specifying of which processor events the process should no longer be notified.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_NOT_REGISTERED
RC_INVALID_CALL
```

Remarks

The structure of the masks is defined under *RegisterAsyncHandler—Register an Async Handler* on page 139. If the process is not registered for one of the events specified in the masks, the process is not deregistered for any of the events.

Hook Services

The kernel provides hooks so processes can be notified of special actions. These hooks have the option of preprocessing or post-processing notification. In other words, processes can be notified either before the action occurs or after the action occurs. This notification takes the form of calling a hook handler registered by the process. Within the hook handler, the process can take whatever actions are required.

The following are the hook services.

Service Name	Page
RegisterHook	147
DeregisterHook	148

Only one hook is initially provided and it is for the dispatcher. A dispatcher hook handler might want to save and restore an environment for processes as they are dispatched.

RegisterHook—Register an Entry Point for a Hook

This service registers an entry point for a hook.

Functional Prototype

```
RIC_ULONG RegisterHook (RIC_HOOKHANDLER EntryPoint,
                        RIC_ULONG       HookNum,
                        RIC_ULONG       OptionWord,
                        RIC_ULONG       Reserved);
```

Parameters

EntryPoint
: Input. The entry point where the process wants control when it is called from the dispatcher.

HookNum
: Input. Number of the hook to register. Initially, only one hook is available: `HOOK_DISPATCH`.

OptionWord
: Input. If the OptionWord is ORed with `HOOK_PREPROCESS`, the entry point of the process is called before the action. If the OptionWord is ORed with `HOOK_POSTPROCESS`, the entry point is called after the action. A process can register for preprocessing and post-processing in the same call.

Reserved
: Input. Reserved parameter (must be 0).

Returns

RC_SUCCESS	RC_INVALID_OPTION
RC_HOOK_ALREADY_REGISTERED	RC_INVALID_RESERVED_PARM
RC_INVALID_MEM_ACCESS	RC_INVALID_HOOK
RC_INVALID_CALL	

Remarks

The hook entry point should be defined in this way:

```
void HookEntry (union HookDataStruc *HookData);
```

where:

`HookDataStruc` is defined as follows:

```
union HookDataStruc
{
   RIC_PROCESSID ProcessInExec;    /* for Dispatch hook */
}
```

DeregisterHook—Deregister an Entry Point for a Hook

This service deregisters an entry point for a hook.

Functional Prototype

```
RIC_ULONG DeregisterHook (RIC_ULONG  HookNum,
                          RIC_ULONG  OptionWord,
                          RIC_ULONG  Reserved);
```

Parameters

HookNum Input. Number of the hook that was registered.

OptionWord
Input. If the OptionWord is ORed with `HOOK_PREPROCESS`, the preprocessing entry point of the process should be deregistered. If the OptionWord is ORed with `HOOK_POSTPROCESS`, the post-processing entry point is deregistered.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                     RC_INVALID_RESERVED_PARM
RC_HOOK_NOT_REGISTERED         RC_INVALID_OPTION
RC_INVALID_MEM_ACCESS          RC_INVALID_HOOK
RC_INVALID_CALL
```

Remarks

None

Kernel Trace Services

The following are the kernel trace services. These services let the user define a trace buffer and to selectively enable and disable trace on different service classes.

Refer to the *ARTIC960 Programmer's Guide* for additional information on trace services.

InitTrace—Initialize a Trace Buffer

This service sets up a trace buffer for logging information during execution.

Functional Prototype

```
RIC_ULONG InitTrace (RIC_ULONG  BufferSize,
                     RIC_SLONG  WrapAroundCount);
```

Parameters

BufferSize Input. Size of the trace buffer in KB.

WrapAroundCount
Input. Number of times the trace buffer should do a wrap around before the trace gets disabled. Use a value of –1 for infinite. A negative value not equal to –1 is invalid.

Returns

```
RC_SUCCESS
RC_INVALID_CALL
RC_NO_MORE_MEM
RC_NO_MORE_RES
RC_INVALID_OPTION
```

Remarks

This service allocates memory for the trace buffer and must be called before any call to EnableTrace or DisableTrace. If called twice, the previous trace buffer is purged and a fresh buffer of size specified in the second call is allocated. In this case, all the previous logs are lost. EnableTrace must be called to enable the logging of the trace data after each call to InitTrace.

EnableTrace—Enable Tracing of Service Classes

This service enables the logging of the trace information for the given set of service classes.

Functional Prototype

`RIC_ULONG EnableTrace (RIC_ULONG ParamCount, ... );`

Parameters

ParamCount
: Input. Number of service classes being supplied as arguments to EnableTrace.

`...`
: Input. List of all the service classes, separated by commas, for which the trace is to be enabled. See the *Remarks* section for details.

Returns

```
RC_SUCCESS
RC_TRACE_NOT_INITIALIZED
RC_INVALID_SERVICECLASS
```

Remarks

InitTrace must be called before any call to EnableTrace.

The EnableTrace service enables the logging of the trace for the service classes given as argument. It does not report errors if the trace on a particular service was already enabled. It enables the trace on the given services, in addition to those for which trace is already enabled. The valid service classes for the kernel are:

```
ALL_SERVICES
C_ASYNC_EVENT_SERVICE
C_CLIB
C_DEVICE_DRIVER_SERVICE
C_EVENT_SERVICE
C_HOOK_SERVICE
C_INTERRUPT_SERVICE
C_KERN_COMMANDS_SERVICE
C_MAILBOX_SERVICE
C_MEMORY_SERVICE
C_MEMPROT_SERVICE
C_PROCESS_SERVICE
C_QUEUE_SERVICE
C_SEMAPHORE_SERVICE
C_SIGNAL_SERVICE
C_SUBALLOC_SERVICE
C_SWTIMER_SERVICE
C_TIMER_SERVICE
```

DisableTrace—Disable Tracing of Service Classes

This service disables the logging of the trace information for the given set of service classes.

Functional Prototype

`RIC_ULONG DisableTrace (RIC_ULONG ParamCount, ... );`

Parameters

ParamCount
: Input. Number of service classes being supplied as arguments to DisableTrace.

`...`
: Input. List of all the service classes, separated by commas, for which the trace is to be disabled. See the *Remarks* section for details.

Returns

```
RC_SUCCESS
RC_TRACE_NOT_INITIALIZED
RC_INVALID_SERVICECLASS
```

Remarks

This service disables the logging of the trace for the service classes given as argument. It does not report errors if the trace on a particular service was already disabled. It disables the trace on the given services, in addition to those for which trace is already disabled. The valid service classes for the kernel are:

```
ALL_SERVICES
C_ASYNC_EVENT_SERVICE
C_CLIB
C_DEVICE_DRIVER_SERVICE
C_EVENT_SERVICE
C_HOOK_SERVICE
C_INTERRUPT_SERVICE
C_KERN_COMMANDS_SERVICE
C_MAILBOX_SERVICE
C_MEMORY_SERVICE
C_MEMPROT_SERVICE
C_PROCESS_SERVICE
C_QUEUE_SERVICE
C_SEMAPHORE_SERVICE
C_SIGNAL_SERVICE
C_SUBALLOC_SERVICE
C_SWTIMER_SERVICE
C_TIMER_SERVICE
```

LogTrace—Log Trace Information

This service logs the trace information in the trace buffer.

Functional Prototype

```
RIC_ULONG LogTrace (RIC_ULONG  ServiceClass,
                   RIC_ULONG       ProcedureID,
                   RIC_ULONG       CallerPosition,
                   RIC_ULONG       TraceOption,
                   RIC_ULONG       DataSize,
                   void           *Address);
```

Parameters

ServiceClass

Input. Identifies the class of the calling procedure and decides whether the trace is to be logged as set by EnableTrace and DisableTrace calls. The range is from 0 to 255. Range 0 to 127 is reserved for the kernel and its subsystems. However, the kernel does not perform checking to enforce the reserved range.

ProcedureID

Input. Identifies the procedure in the given service class. The ServiceClass and the ProcedureID together form a unique identification for any procedure. Range is from 0 to 255.

CallerPosition

Input. Provides information regarding the position of the caller inside the procedure. The following values are supported.

`TRACE_ENTRY`
To mark the entry into any procedure.

`TRACE_EXIT`
To mark the exit from any procedure.

Values `0x00000001` to `0xFE`
To mark different positions inside any procedure.

TraceOption

Input. Decides what is to be logged and how it is displayed after formatting by `RICFMTTR`.

You can OR more than one option together to form a TraceOption. If both `TRACE_INT` and `TRACE_CHAR` are used, the data is displayed in both forms in two consecutive trace records.

`TRACE_INT`
Take the data from Address and display as integers.

`TRACE_CHAR`
Take the data from Address and display in hexadecimal and ASCII.

`TRACE_NOINFO`
No data is associated with this trace record.

DataSize Input. Number of bytes of data to be logged from Address. The DataSize must be 0 and the address must be NULL if TraceOption is `TRACE_NOINFO`.

Address Input. Pointer to the buffer containing the data to be logged.

Returns

```
RC_SUCCESS                      RC_INVALID_SERVICECLASS
RC_INVALID_MEM_ACCESS           RC_INVALID_PROCEDURE_ID
RC_INVALID_OPTION               RC_INVALID_CALLER_POSITION
RC_TRACE_NOT_INITIALIZED
```

Remarks

This service logs the trace information for the calling procedure, if the trace was enabled for the service class of the calling procedure. The task calling This service must be compiled with the –DTRACE option. The service classes defined for the kernel are:

```
C_ASYNC_EVENT_SERVICE
C_CLIB
C_DEVICE_DRIVER_SERVICE
C_EVENT_SERVICE
C_HOOK_SERVICE
C_INTERRUPT_SERVICE
C_KERN_COMMANDS_SERVICE
C_MAILBOX_SERVICE
C_MEMORY_SERVICE
C_MEMPROT_SERVICE
C_PROCESS_SERVICE
C_QUEUE_SERVICE
C_SEMAPHORE_SERVICE
C_SIGNAL_SERVICE
C_SUBALLOC_SERVICE
C_SWTIMER_SERVICE
C_TIMER_SERVICE
```

Kernel Trace Information

The following tables indicate the procedures that are traced when a particular service class is enabled. They also indicate the contents of the trace records associated with each procedure.

.

Table 3-1. Service Class: C_ASYNC_EVENT_SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_REGISTER_ASYNC_HNDLER	RegisterAsyncHandler	Entry Exit	SoftwareEvents AdapterEvents ProcessorEvents AsyncHandler rc	integer integer integer integer integer
P_DEREGISTER_ASYNCH_ HNDLER	DeregisterAsyncHandler	Entry Exit	SoftwareEvents AdapterEvents ProcessorEvents rc	integer integer

Table 3-2. Service Class: C_DEVICE_DRIVER_SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_SETVECTOR	SetVector/SetVectorMux	Entry Exit	VectorNum rc	integer integer
P_ALLOCVECTOR	AllocVector/AllocVectorMux	Entry Exit	VectorNum rc	integer integer
P_RETURNVECTOR	ReturnVector	Entry Exit	VectorNum rc	integer integer
P_ALLOCHW	AllocHW	Entry Exit	DeviceName rc	character integer
P_RETURNHW	ReturnHW	Entry Exit	DeviceName rc	character integer
P_QUERYHW	QueryHW	Entry Exit	DeviceName rc	character integer
P_CREATEDEV	CreateDev	Entry Exit	DDName rc	character integer
P_OPENDEV	OpenDev	Entry Exit	DDName rc	character integer
P_INVOKEDEV	InvokeDev	Entry Exit	DDHandle rc	integer integer
P_CLOSEDEV	CloseDev	Entry Exit	DDHandle rc	integer integer

Table 3-3. Service Class: C_EVENT_SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_CREATEEVENT	CreateEvent	Entry Exit	EvnName rc	character integer
P_OPENEVENT	OpenEvent	Entry Exit	EvnName rc	character integer
P_CLOSEEVENT	CloseEvent	Entry Exit	EvnHandle rc	integer integer
P_WAITEVENT	WaitEvent	Entry Exit	EvnHandle rc	integer integer

Table 3-4. Service Class: C_HOOK_SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_REGISTERHOOK	RegisterHook	Entry Exit	HookNum rc	integer integer
P_DEREGISTERHOOK	DeregisterHook	Entry Exit	HookNum rc	integer integer

Table 3-5. Service Class: C_INTERRUPT_SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_FIRSTLEVELINT	First level interrupt handler	Entry	Vector number	integer

Table 3-6. Service Class: C_KERN_COMMANDS_SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_PROCESSMBXCOMMAND	Receiving a command in the kernel mailbox	Entry Exit	CommandNum none	integer

A NULL resource name is displayed as the string “Null Pointer.” An invalid resource name is displayed as the string “Invalid Pointer.”

Table 3-7. Service Class: C_MAILBOX_SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_CREATEMBX	CreateMbx	Entry Entry Exit	MbxName MbxRxMemName rc	character character integer
P_OPENMBX	OpenMbx	Entry Entry Exit	MbxName SendMemName rc	character character integer
P_SENDMBX	SendMbx	Entry Exit	MbxHandle rc	integer integer
P_GETMBXBUFFER	GetMbxBuffer	Entry Exit	MbxHandle rc	integer integer
P_FREEMBXBUFFER	FreeMbxBuffer	Entry Exit	MbxHandle rc	integer integer
P_RECEIVEMBX	ReceiveMbx	Entry Exit	MbxHandle rc	integer integer
P_CLOSEMBX	CloseMbx	Entry Exit	MbxHandle rc	integer integer

Table 3-8. Service Class: C_MEMORY_SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_CREATEMEM	CreateMem	Entry Exit	MemName rc	character integer
P_OPENMEM	OpenMem	Entry Exit	MemName rc	character integer
P_CLOSEMEM	CloseMem	Entry Exit	Baseptr rc	integer integer
P_RESIZEMEM	ResizeMem	Entry Exit	Baseptr rc	integer integer
P_SETMEMPROT	SetMemProt	Entry Exit	BlockPtr rc	integer integer
P_QUERYMEMPROT	QueryMemProt	Entry Exit	BlockPtr rc	integer integer
P_QUERYFREEMEM	QueryFreeMem	Entry Exit	OptionWord rc	integer integer
P_MALLOCMEM	MallocMem	Entry Exit	Size Baseptr	integer integer
P_FREEMEM	FreeMem	Entry Exit	Blockptr rc	integer integer
P_COLLECTMEM	CollectMem	Entry Exit	Option rc *FreeUnits *FreePages	integer integer integer integer

Table 3-9. Service Class: C_PROCESS_SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_QUERYPROCESSSTATUS	QueryProcessStatus	Entry Entry Exit	OptionWord ProcessName or ProcessID rc	integer character integer integer
P_SETPRIORITY	SetPriority	Entry Entry Exit	ProcessID Priority rc	integer integer integer
P_QUERYPRIORITY	QueryPriority	Entry Exit Exit	ProcessID Priority rc	integer integer integer
P_STOPPROCESS	StopProcess	Entry Exit	ProcessID rc	integer integer
P_UNLOADPROCESS	UnloadProcess	Entry Exit	ProcessID rc	integer integer
P_STARTPROCESS	StartProcess	Entry Exit	ProcessID rc	integer integer
P_CREATEPROCESS	CreateProcess	Entry Exit	ProcessName rc	character integer
P_COMPLETEINIT	CompleteInit	Entry Exit	none rc	integer
P_SUSPENDPROCESS	SuspendProcess	Entry Exit	ProcessID rc	integer integer
P_RESUMEPROCESS	ResumeProcess	Entry Exit	ProcessID rc	integer integer
P_QUERYPROCESSINEXEC	QueryProcessInExec	Entry Exit Exit	none ProcessID rc	 integer integer
P_QUERYCARDINFO	QueryCardinfo	Entry Exit	none rc	 integer
P_QUERYCONFIGPARAMS	QueryConfigParams	Entry Exit	none rc	 integer
P_SETPROCESSDATA	SetProcessData	Entry Exit	ApplID rc	character integer
P_GETPROCESSDATA	GetProcessData	Entry Exit	ApplID *ProcessDataPtr	character integer
P_ENTERCRITSEC	EnterCritSec	Entry Exit	OptionWord rc	integer integer
P_EXITCRITSEC	ExitCritSec	Entry Exit	OptionWord rc	integer integer

Table 3-10. Service Class: C_MEMPROT_SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_SETPROCMEMPROT	SetProcMemProt	Entry Exit	ProcessID rc	integer integer
P_QUERYPROCMEMPROT	QueryProcMemProt	Entry Exit	ProcessID rc	integer integer

Table 3-11. Service Class: C_QUEUE_SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_CREATEQUEUE	CreateQueue	Entry Exit	QueueName rc	character integer
P_OPENQUEUE	OpenQueue	Entry Exit	QueueName rc	character integer
P_CLOSEQUEUE	CloseQueue	Entry Exit	QueueHandle rc	integer integer
P_PUTQUEUE	PutQueue	Entry Exit	QueueHandle rc	integer integer
P_GETQUEUE	GetQueue	Entry Exit	QueueHandle rc	integer integer
P_SEARCHQUEUE	SearchQueue	Entry Exit	QueueHandle rc	integer integer

Table 3-12. Service Class: C_SEMAPHORE_SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_CREATESEM	CreateSem	Entry Exit	SemName rc	character integer
P_OPENSEM	OpenSem	Entry Exit	SemName rc	character integer
P_CLOSESEM	CloseSem	Entry Exit	SemHandle rc	integer integer
P_RELEASESEM	ReleaseSem	Entry Exit	SemHandle rc	integer integer
P_REQUESTSEM	RequestSem	Entry Exit	SemHandle rc	integer integer
P_QUERYSEMCOUNT	QuerySemCount	Entry Exit	SemHandle rc	integer integer
P_SETSEMCOUNT	SetSemCount	Entry Exit	SemHandle rc	integer integer

Table 3-13. Service Class: C_SIGNAL_SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_CREATESIG	CreateSig	Entry Exit	SigName rc	character integer
P_OPENSIG	OpenSig	Entry Exit	SigName rc	character integer
P_INVOKESIG	InvokeSig	Entry Exit	SigHandle rc	integer integer
P_CLOSESIG	CloseSig	Entry Exit	SigHandle rc	integer integer

Table 3-14. Service Class: C_SUBALLOC_SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_GETSUBALLOCSIZE	GetSuballocSize	Entry Exit	UnitSize rc	integer integer
P_INITSUBALLOC	InitSuballoc	Entry Exit	BlockPtr rc	integer integer
P_GETSUBALLOC	GetSuballoc	Entry Exit	BlockPtr rc	integer integer
P_FREESUBALLOC	FreeSuballoc	Entry Exit	BlockPtr rc	integer integer

Table 3-15. Service Class: C_SWTIMER SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_CREATESWTIMER	CreateSwTimer	Entry Exit	TimerName rc	character integer
P_CLOSESWTIMER	CloseSwTimer	Entry Exit	TimerHandle rc	integer integer
P_STARTSWTIMER	StartSwTimer	Entry Exit	TimerHandle rc	integer integer
P_STOPSWTIMER	StopSwTimer	Entry Exit	TimerHandle rc	integer integer

Table 3-16. Service Class: C_TIMER_SERVICE

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_SETSYSTIME	SetSystemTime	Entry Exit	SysTimeInfo.Time rc	integer integer
P_QUERYSYSTIME	QuerySystemTime	Entry Exit	none rc	 integer
P_STARTPERFTIMER	StartPerfTimer	Entry Exit	none rc	 integer
P_STOPPERFTIMER	StopPerfTimer	Entry Exit	none rc	 integer
P_READPERFTIMER	ReadPerfTimer	Entry Exit	none rc	 integer

Table 3-17. Service Class: C_CLIB

Procedure ID	Kernel Service	Trace Position	Trace Data	Format of Data
P_FD_STDOUT	printf	–	–	character
P_FD_STDERR	printf	–	–	character

4 Kernel Commands

Processes located on a remote card or in the system unit can send command and status requests to the kernel, using the kernel command facility. These requests are sent to the kernel through its mailbox, named "RIC_KERNMBX*n*" (*n* is a logical card number). As an example, commands destined for the kernel on logical card 0 would be sent to mailbox "RIC_KERNMBX0".

Command completion status is returned to the requester in a response mailbox specified using the RegisterResponseMbx command. Requesters should check the status provided in the response mailbox to verify successful command completion.

The following kernel commands have been defined. Refer to the *ARTIC960 Programmer's Guide* for additional information.

Service Group	**Page**
RegisterResponseMbx	166
DeRegisterResponseMbx	167
QueryProcessStatus	168
UnloadProcess	169
StopProcess	170
StartProcess	171

Common Headers for Commands and Responses

Commands

All commands have a common header with a variant part unique for each command. The format is:

```
struct RIC_KernCommand
{
struct  RIC_KernMbxCmd                Header;
  union
  {
struct RIC_RegisterResponseMbxCmd     Cmd0;
struct RIC_DeregisterResponseMbxCmd   Cmd1;
struct RIC_QueryProcessStatusCmd      Cmd2;
struct RIC_StopProcessCmd             Cmd3;
struct RIC_StartProcessCmd            Cmd4;
struct RIC_UnloadProcessCmd           Cmd5;
   }Cmds;
};

struct RIC_KernMbxCmd
   {
   RIC_ULONG   CommandNum;
   RIC_RESPMBX RespMbxID;
   RIC_ULONG   CorrelationID;
   RIC_ULONG   ReturnCode;
   RIC_ULONG   Reserved;
};
```

where:

CommandNum
: Command number unique to each kernel command.

RespMbxID
: ID returned on RegisterResponseMbx that indicates the mailbox where the command response is to be sent.

CorrelationID
: Value that is passed on with the command and is not interpreted by the kernel. The requester can use the field to correlate command responses.

ReturnCode
: Reserved field (must be 0)

Reserved Resereved field (must be 0)

Responses

Like commands, responses have a common header with a variant part unique to each response. Some responses have no variant part. The format is:

```
struct RIC_KernResponse
{
   RIC_ULONG     CorrelationID;
   RIC_ULONG     ReturnCode;
   RIC_ULONG     Reserved;
   union
   {
      struct RIC_RegisterResponseMbxResp    Resp0;
      struct RIC_QueryProcessStatusResp     Resp1;
   }Resp;
};
```

where:

CorrelationID
: Value passed in the command. The field can be used to correlate command responses.

ReturnCode
: Return code returned by the kernel to indicate the completion status of the command.

Reserved
: Reserved field (must be 0)

If a bad RespMbxID is passed on a command, the kernel ignores the command and a timeout on the reply occurs.

RegisterResponseMbx—Register a Command Response Mailbox

This command returns the response mailbox ID associated with the specified response mailbox name.

Command Parameters

CommandNum in the common header must be set to KERN_REG_RESP_MBX.

RespMbxId in the common header is not defined for this command.

Structures

```
struct RIC_RegisterResponseMbxCmd
{
  char        MbxName[MAX_RES_USER];
  RIC_ULONG   Reserved;
}
```

where:

MbxName Response mailbox name

Reserved Reserved field (must be 0)

Response Parameters

ReturnCode values in RIC_KernResponse are:

RC_SUCCESS

The following shows the variant part of the response for this command.

```
struct RIC_RegisterResponseMbxResp
{
  RIC_RESPMBX    RespMbxID;
  RIC_ULONG      Reserved;
}
```

where:

RespMbxID
Identifier used on all subsequent kernel commands

Reserved Reserved field (must be 0)

Remarks

This command must be issued prior to any other commands being issued. It is the user's responsibility to issue a DeRegisterResponseMbx command when the application terminates.

DeRegisterResponseMbx—Deregister a Command Response Mailbox

This command removes a response mailbox when its ID is specified.

Command Parameters

CommandNum in the common header must be set to `KERN_DEREG_RESP_MBX`.

Structures

```
struct RIC_DeRegisterResponseMbxCmd
{
   RIC_RESPMBX     RespMbxID;
   RIC_ULONG       Reserved;
}
```

where:

RespMbxID
: Response mailbox identifier

Reserved
: Reserved field (must be 0)

Response Parameters

ReturnCode values in RIC_KernResponse are:

```
RC_INVALID_RESERVED_PARM
RC_INVALID_HANDLE
```

There is no variant response part for this command.

Remarks

None

QueryProcessStatus—Get the Process Status

This command returns the process status and process identification, when the process name is specified.

Command Parameters

CommandNum in the common header must be set to KERN_QUERY_PROC_STAT.

Structures

```
struct RIC_QueryProcessStatusCmd
{
  char                          ProcName[MAX_RES_USER];
  struct RIC_ProcessStatusBlock ProcSB;
  RIC_ULONG                     Reserved;
}
```

where:

ProcName Process name

ProcSB Reference to the structure containing status information. See *QueryProcessStatus—Get the Process Status* on page 25 for the format of the process status block.

Reserved Reserved field (must be 0)

Response Parameters

ReturnCode values in RIC_KernResponse are:

```
RC_SUCCESS
RC_INVALID_NAME
RC_INVALID_RESERVED_PARM
RC_NAME_NOT_FOUND
```

The following shows the variant part of the response for this command.

```
struct RIC_QueryProcessStatusResp
{
  struct RIC_ProcessStatusBlock ProcSB;
  RIC_ULONG                     Reserved;
}
```

where:

ProcSB Reference to the structure containing status information. See *QueryProcessStatus—Get the Process Status* on page 25 for the format of the process status block.

Reserved Reserved field (must be 0)

Remarks

None

UnloadProcess—Unload a Process

This command unloads a process, given its process ID.

Command Parameters

CommandNum in the common header must be set to `KERN_UNLOAD_PROC`.

Structures

```
struct RIC_UnloadProcessCmd
{
  RIC_PROCESSID  ProcessID;
  RIC_ULONG      Reserved;
}
```

where:

ProcessID	Process identification
Reserved	Reserved field (must be 0)

Response Parameters

ReturnCode values in RIC_KernResponse are:

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_PROCESSID
RC_PERMANENT_PROCESS
```

There is no variant response part for this command.

Remarks

None

StopProcess—Stop a Process

This command stops a process, given its process ID.

Command Parameters

CommandNum in the common header must be set to `KERN_STOP_PROC`.

Structures

```
struct RIC_StopProcessCmd
{
  RIC_PROCESSID  ProcessID;
  RIC_ULONG      Reserved;
}
```

where:

ProcessID Process identification

Reserved Reserved field (must be 0)

Response Parameters

ReturnCode values in RIC_KernResponse are:

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_PROCESS_NOT_STARTED
RC_INVALID_PROCESSID
RC_PERMANENT_PROCESS
```

There is no variant response part for this command.

Remarks

None

StartProcess—Start a Process

This command starts the process specified by the process ID.

Command Parameters

CommandNum in the common header must be set to `KERN_START_PROC`.

Structures

```
struct RIC_StartProcessCmd
{
  RIC_PROCESSID  ProcessID;
  RIC_ULONG      OptionWord;
  RIC_SLONG      TimeOut;
  RIC_ULONG      Reserved;
}
```

where:

ProcessID Process identification.

OptionWord

Bit field indicating whether the requester wants the kernel to wait for the process being started to perform a CompleteInit before the kernel returns a return code (and thus completes the command).

`WAIT_FOR_COMPLETEINIT`
The kernel waits for the starting process to issue the CompleteInit call.

`NO_WAIT_FOR_COMPLETEINIT`
The kernel does not wait.

TimeOut Time, specified in seconds, that the kernel waits for the process to perform the CompleteInit. The actual time waited is a multiple of approximately 1/4 seconds. A value of zero indicates that the requester does not want to wait. There is no infinite timeout.

Reserved Reserved field (must be 0)

Response Parameters

ReturnCode values in RIC_KernResponse are:

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_INVALID_PROCESSID
RC_PROCESS_ALREADY_STARTED
RC_PROCESS_STOPPED
RC_TIMEOUT
Error code used by a process on CompleteInit
```

There is no variant response part for this command.

Remarks

None

5 Adapter Library Routines

This chapter lists ANSI C library calls and describes the Miscellaneous Service, the System Bus Interface Services, and the PCI Services.

ANSI C Functions

The following ANSI C library calls are supported by the kernel.

Use of some functions may require that additional libraries be used. Refer to your compiler documentation for details. You can modify your **ricproc.ld** file to include additional libraries. Refer to the *ARTIC960 Programmer's Guide* for information about doing this.

Character Handling

isalnum	isgraph	ispunt	isxdigit
isalpha	islower	isspace	tolower
iscntrl	isprint	isupper	toupper
isdigit			

Mathematics

acos	cosh	ldexp	sinh
asin	exp	log	sqrt
atan	fabs	log10	tan
atan2	floor	modf	tanh
ceil	fmod	pow	
cos	frexp	sin	

Variable Arguments

va_arg	va_end	va_start	

Input/Output

fflush[2,3]	printf[2,3]	sprintf	sscanf

General Utilities

abs	atol	free[2]	srand
atexit[1]	bsearch	malloc[2]	strtod
atof	div	qsort[1]	strtol
atoi	exit[1,2]	rand	strtoul

1. Some ANSI C functions cannot be called from interrupt handlers.
2. These functions are implemented in libricc.a (OS/2) and libriccx.a (AIX) along with other kernel services.
3. Refer to the *ARTIC960 Programmer's Guide* for information on using this C function.

String Handlings

memchr	strcat	strerror	strpbrk
memcmp	strchr	strlen	strrchr
memcpy	strcmp	strncat	strspn
memmove	strcpy	strncmp	strstr
memset	strcspn	strncpy	strtok

Date and Times

asctime	difftime	localtime	time[2]
ctime	gmtime	mktime	

1. Some ANSI C functions cannot be called from interrupt handlers.
2. These functions are implemented in libricc.a (OS/2) and libriccx.a (AIX) along with other kernel services.
3. Refer to the *ARTIC960 Programmer's Guide* for information on using this C function.

Miscellaneous Service

ProcessSleep—Sleep a Process

This service blocks a process for the specified length of time.

Functional Prototype

```
RIC_ULONG ProcessSleep (RIC_TIMEOUT Timevalue,
                        RIC_ULONG   Reserved);
```

Parameters

Timevalue Input. The length of time in milliseconds for the process to sleep.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_CALL
RC_INVALID_RESERVED_PARM
RC_INVALID_TIMEOUT
RC_NO_MORE_SEM
RC_NO_MORE_TIMERS
```

Remarks

This library routine allows users to block for a specified period of time without first creating a semaphore. It does the equivalent of creating a semaphore, blocking on it for the requested time, and then closing the semaphore. This routine is contained in the file **libricc.a**.

System Bus Interface Services

The kernel provides the following system-bus interface services.

Service	Page
MoveMCData	177
ConvertMCToCard	181
ConvertCardToMC	182

These services allow a process to perform system bus operations. They are provided by the System Bus I/O Subsystem RIC_MCIO.REL.

Programs that use the system bus interface services must define the constant `INCL_MCHL` prior to the `#include <ric.h>` statement to obtain the proper declarations.

The libraries for these services are contained in the file **libmclib.a** for OS/2 and **libmclibx.a** for AIX. To link a program with this library, add `-lmclib` or `-lmclibx` to the LNK960 call. Refer to the *ARTIC960 Programmer's Guide* for more information.

MoveMCData—Move System Bus Data

This service moves data to/from a buffer on the local unit from/to a buffer on another unit (a remote unit) through a system bus operation. It blocks the requesting process until the operation is complete.

Functional Prototype

```
RIC_ULONG MoveMCData (RIC_DEVHANDLE         DDHandle,
                      struct RIC_MoveBlock *PPtr,
                      RIC_ULONG              Reserved);
```

Parameters

DDHandle Input. Handle of subsystem returned by OpenDev of System Bus I/O Subsystem.

PPtr Input. Pointer to the move block

Reserved Input. Reserved parameter (must be 0)

The move block structure is:

```
struct RIC_MoveBlock
{
   RIC_ULONG              OptionWord;
   RIC_CARDNUM            SourceCard;
   RIC_CARDNUM            DestCard;
   void                  *SourceAddptr;
   void                  *DestAddptr;
   RIC_ULONG              Size;
   struct RIC_MoveBlock  *ChainPtr;
}
```

OptionWord

Input. Specifies options that can be selected for system bus operations. See the Remarks section for option information.

SourceCard

Input. The logical card number for the ARTIC960 adapter source unit. A valid ARTIC960 logical card number indicates that the source address specifies a local address on that unit.

`MC_SU_ADDR_CARD_NUMBER`

This value indicates that the source unit is the system unit and that the source address specifies a physical system bus address for the system unit.

MC_ADPT_ADDR_CARD_NUMBER

This value indicates that the source unit is not an ARTIC960 adapter and that the source address specifies a physical system bus address for that adapter.

- An ARTIC960 address cannot be expressed as a physical system bus address.
- ARTIC960 Support for AIX Version 1.1 or higher and ARTIC960 Support for NT Version 1.0 do not support DMA transfers between two peer adapters. They support DMA transfers only between the adapter and system unit.

DestCard Input. The logical card number for the ARTIC960 adapter destination unit. A valid ARTIC960 logical card number indicates that the destination address specifies a local address on that unit.

MC_SU_ADDR_CARD_NUMBER

This value indicates that the destination unit is the system unit and that the destination address specifies a physical system bus address for the system unit.

MC_ADPT_ADDR_CARD_NUMBER

This value indicates that the destination unit is not an ARTIC960 adapter and that the destination address specifies a physical system bus address for that adapter.

An ARTIC960 address cannot be expressed as a physical system bus address.

SourceAddptr

Input. The address of the source buffer on the source unit. The address format is determined by the source unit field.

DestAddptr Input. The address of the destination buffer on the destination unit. The address format is determined by the destination unit field.

Size Input. The number of bytes of data to be moved. The maximum value is 1M–1 for the MCA adapter. The maximum value is 16M–1 for PCI adapters. 0 is not valid.

ChainPtr Input. The pointer to the next Move structure. If this is the last block in the chain, this field is NULL.

Returns

RC_SUCCESS
RC_INVALID_ADDRESS
RC_INVALID_COMBINATION
RC_INVALID_SIZE
RC_INVALID_OPTION
RC_INVALID_CARD_NUMBER
RC_INVALID_MEM_ACCESS
RC_INVALID_CALL
RC_MC_DATA_PARITY_ERR
RC_MC_CHCK_ERR
RC_MC_CARD_SEL_FDBACK_ERR
RC_MC_LOSS_OF_CHANNEL_ERR
RC_MC_LOCAL_BUS_PARITY_ERR
RC_MC_EXCEPTION_ERR
RC_MC_TIMEOUT
RC_MC_INVALID_COMBINATION
RC_MC_CHAINING_EX_ERR
RC_MC_POSTSTAT_EX_ERR
RC_RESET_ACTIVE
RC_MC_MASTER_ABORT
RC_MC_BUS_FAULT
RC_MC_MEM_FAULT

Remarks.

The caller of this service must ensure that a reset does not occur during a system bus operation.

- This function may block the requesting process. The function returns to the caller when the move is complete.
- To open the system bus I/O subsystem, issue the following command:

```
OpenDev (MCSSNAME, (void *) NULL, 0 , &DDhandle);
```

- The source and destination units must be different, and one must be the requester unit.
- No validation is done on the physical system bus addresses.
- The logical card number is checked for validity.
- If memory protection is enabled, the local memory address is checked for system bus access and a RC_INVALID_MEM_ACCESS error is returned if access is not correct.
- An unsupported system-unit address can generate a channel check.
- Because requests can be passed to two different system bus DMA channels, the order of message delivery is not guaranteed. The order is guaranteed only within a chain.

  This function does not support mixing of card-to-card and card-to-system unit moves chained in the same MoveMCData request. To ensure correct operation, such requests should be issued in separate MoveMCData requests.

- The following constants have been defined for the OptionWord parameter.

  `MOV_MEMORY`
  : Move is for a memory address (default).

  `MOV_IO`
  : Move is for an I/O address.

    `RC_INVALID_OPTION` is returned if:

    - The peer card is an ARTIC960Rx PCI adapter or ARTIC960RxD PCI adapter. These adapters do not support I/O.
    - The initiator card is an ARTIC960Rx PCI or ARTIC960RxD PCI adapter, and the peer card does not have memory-mapped I/O.

  `MOV_INCR`
  : Increment remote-unit address after each byte transfer (default).

  `MOV_NO_INCR`
  : Do not increment remote-unit address after each byte transfer. This option may be used to move consecutive bytes to an I/O address. This option is ignored by PCI devices.

ConvertMCToCard—Convert System Bus Address to Card Address

This service converts a system bus address to a logical card number and local address pointer.

Functional Prototype

```
RIC_ULONG  ConvertMCToCard  (RIC_DEVHANDLE      DDHandle,
                             void              *MCAddress,
                             RIC_CARDNUM       *Card,
                             void             **LocalAddptr,
                             RIC_ULONG          Reserved);
```

Parameters

DDHandle Input. Handle of subsystem returned by OpenDev of system bus I/O Subsystem.

MCAddress Input. System bus address to be converted.

Card Output. Logical card number represented by system bus address.

LocalAddptr
Output. Local address on the indicated logical card.

Reserved Input. Must be 0.

Returns

```
RC_SUCCESS
RC_UNABLE_TO_CONVERT_ADDRESS.
```

The compatibility of this function is not guaranteed across future releases.

ConvertCardToMC—Convert Card Address to System Bus Address

This service converts a logical card number and local address pointer to a system bus address.

Functional Prototype

```
RIC_ULONG  ConvertCardToMC  (RIC_DEVHANDLE      DDHandle,
                             RIC_CARDNUM        Card,
                             void              *LocalAddptr,
                             void             **MCAddress,
                             RIC_ULONG          Reserved);
```

Parameters

DDHandle Input. Handle of subsystem returned by OpenDev of System Bus I/O Susbystem.

Card Input. Logical card number for local address.

LocalAddptr
Input. Local address to be converted.

MCAddress
Output. Converted system bus address.

Reserved Input. Must be 0.

Returns

```
RC_SUCCESS
RC_INVALID_CARD_NUMBER.
```

The compatibility of this function is not guaranteed across future releases.

PCI Local Bus Configuration Device Driver Services

The kernel provides the following adapter PCI local-bus interface services.

These services call a device driver to access PCI devices on the adapter's local PCI bus on the ARTIC960Rx and ARTIC960Hx adapters. They are provided by the PCI Device Driver RIC_PCI.REL.

Programs that use the PCI local bus interface services must define the constant `INCL_PCI` prior to the `#include <ric.h>` statement to obtain the proper declarations.

The libraries for these services are contained in the regular kernel services libraries.

pciBiosPresent—Query PCI Driver Presence

This service determines the presence of the PCI device driver, and returns version information and the number of PCI buses in the system.

Functional Prototype

```
RIC_ULONG pciBiosPresent (struct PCI_BIOS_INFO *PCI_InfoPtr);
```

Parameters

PCI_InfoPtr
: Input. Pointer to the user's structure. The PCI parameters are copied into this memory.

Returns

```
RC_SUCCESS
RC_PCI_NO_BIOS
```

Remarks

```
struct PCI_BIOS_INFO
{
  RIC_ULONG         Options;
  RIC_ULONG         DriverVersion;
  RIC_ULONG         BIOSVersion;
  RIC_ULONG         LastBus;
  RIC_ULONG         LocalMemBase;
  RIC_ULONG         LocalIO_Base;
  unsigned char     IntPinA_Vector;
  unsigned char     IntPinB_Vector;
  unsigned char     IntPinC_Vector;
  unsigned char     IntPinD_Vector;
};
```

Options Reserved parameter (currently 0)

DriverVersion
: Version number of the RIC_PCI.REL device driver

BIOSVersion
: PCI BIOS version number compatible

LastBus Number of the last PCI bus on the adapter

LocalMemBase
: Local bus base address for i960 access to a PCI-device memory (see LocalIO_Base for more information).

LocalIO_Base

Local bus base address for i960 access to a PCI-device memory-mapped I/O

The LocalMemBase and LocalIO_Base values are used as a base address when accessing a PCI device from the i960. These values should be added to the physical address read from a PCI-device base address register to obtain a local i960 address for accessing the device. The LocalMemBase value should be used for memory base address registers, and the LocalIO_Base value should be used for accessing memory-mapped I/O base address registers.

IntPinA_Vector, IntPinB_Vector, IntPinC_Vector, IntPinD_Vector

PCI interrupt-pin vector assignments

Normally, the interrupt-line-configuration register for the device should be read to determine the vector. The IntPin information is provided for deviant PMCs.

pciFindDevice—Find a PCI Device by Vendor and Device ID

This service finds the PCI device that is specified by the vendor and device ID.

Functional Prototype

```
RIC_ULONG pciFindDevice (PCI_DEVICE_ID DeviceID,
                         PCI_VENDOR_ID VendorID,
                         PCI_INSTANCE  Instance,
                         PCI_ID       *pciID);
```

Parameters

DeviceID Input. The PCI device ID.

VendorID Input. The PCI vendor ID.

Instance Input. The instance number of the device. The first device with a given device and vendor ID is instance zero. The next device with the same device and vendor ID is instance one.

pciID Output. If the device is found, a unique identifier for the device is returned. This identifier is then used when accessing the device on subsequent PCI driver calls.

Returns

```
RC_SUCCESS
RC_PCI_NO_BIOS
RC_PCI_DEVICE_NOT_FOUND
```

Remarks

To find multiple devices having the same vendor ID and device ID, the calling software should make successive calls to this function starting with Instance set to zero and incrementing it until the return code is `RC_PCI_DEVICE_NOT_FOUND`.

pciFindClassCode—Find a PCI Device by PCI Class Code

This service finds a specific PCI device given a class code.

Functional Prototype

```
RIC_ULONG pciFindClassCode (PCI_CLASS_CODE ClassCode,
                            PCI_INSTANCE   Instance,
                            PCI_ID        *pciID);
```

Parameters

ClassCode Input. The PCI device class code.

Instance Input. The instance number of the device. The first device with the given class code is instance zero. The next device with the same class code is instance one.

pciID Output. If the device is found, a unique identifier for the device is returned. This identifier is then used when accessing the device on subsequent PCI driver calls.

Returns

```
RC_SUCCESS
RC_PCI_NO_BIOS
RC_PCI_DEVICE_NOT_FOUND
```

Remarks

To find multiple devices having the same class code, the calling software should make successive calls to this function starting with Instance set to zero and incrementing it until the return code is `RC_PCI_DEVICE_NOT_FOUND`.

pciReadConfigByte—Read a Byte from PCI Configuration Space

This service reads one byte from the device PCI configuration space.

Functional Prototype

```
RIC_ULONG pciReadConfigByte (PCI_ID         pciID,
                             PCI_REG_NUM    RegNum,
                             unsigned char *Value);
```

Parameters

pciID Input. The PCI device identifier obtained by way of the pciFindDevice or pciFindClassCode service.

RegNum Input. The register number to be read (normally 0 to 255).

Value Output. The byte read is returned in this parameter.

Returns

```
RC_SUCCESS
RC_PCI_NO_BIOS
RC_PCI_BAD_REGISTER_NUMBER
```

Remarks

None

pciReadConfigWord—Read a Word from PCI Configuration Space

This service reads one 16-bit word from the device PCI configuration space.

Functional Prototype

```
RIC_ULONG pciReadConfigWord (PCI_ID          pciID,
                             PCI_REG_NUM     RegNum,
                             RIC_USHORT     *Value);
```

Parameters

pciID Input. The PCI device identifier obtained by way of either the pciFindDevice or pciFindClassCode service.

RegNum Input. The register number to be read (normally 0 to 255). The register number must be divisible by 2.

Value Output. The word read is returned in this parameter.

Returns

```
RC_SUCCESS
RC_PCI_NO_BIOS
RC_PCI_BAD_REGISTER_NUMBER
```

Remarks

None

pciReadConfigDWord—Read a Doubleword from PCI Configuration Space

This service reads one 32-bit doubleword from the device PCI configuration space.

Functional Prototype

```
RIC_ULONG pciReadConfigDWord (PCI_ID         pciID,
                              PCI_REG_NUM    RegNum,
                              RIC_ULONG     *Value);
```

Parameters

pciID Input. The PCI device identifier obtained by way of either the pciFindDevice or pciFindClassCode service.

RegNum Input. The register number to be read (normally 0 to 255). The register number must be evenly divisible by 4.

Value Output. The doubleword read is returned in this parameter.

Remarks

None

pciWriteConfigByte—Write a Byte to PCI Configuration Space

This service writes one byte to the device PCI configuration space.

Functional Prototype

```
RIC_ULONG pciWriteConfigByte (PCI_ID        pciID,
                              PCI_REG_NUM   RegNum,
                              unsigned char Value);
```

Parameters

pciID	Input. The PCI device identifier obtained by way of either the pciFindDevice or pciFindClassCode service.
RegNum	Input. The register number to be read (normally 0 to 255).
Value	Input. The byte to be written.

Returns

```
RC_SUCCESS
RC_PCI_NO_BIOS
RC_PCI_BAD_REGISTER_NUMBER
```

Remarks

None

pciWriteConfigWord—Write a Word to PCI Configuration Space

This service writes one 16-bit word to the device PCI configuration space.

Functional Prototype

```
RIC_ULONG pciWriteConfigWord (PCI_ID        pciID,
                              PCI_REG_NUM   RegNum,
                              RIC_USHORT    Value);
```

Parameters

pciID Input. The PCI device identifier obtained by way of either the pciFindDevice or pciFindClassCode service.

RegNum Input. The register number to be read (normally 0 to 255). The register number must be evenly divisible by 2.

Value Input. The word to be written.

Returns

```
RC_SUCCESS
RC_PCI_NO_BIOS
RC_PCI_BAD_REGISTER_NUMBER
```

Remarks

None

pciWriteConfigDWord—Write a Doubleword to PCI Configuration Space

This service writes one 32-bit doubleword to the device PCI configuration space.

Functional Prototype

```
RIC_ULONG pciWriteConfigDWord (PCI_ID        pciID,
                               PCI_REG_NUM   RegNum,
                               RIC_ULONG     Value);
```

Parameters

pciID Input. The PCI device identifier obtained by way of either the pciFindDevice or pciFindClassCode service.

RegNum Input. The register number to be read (normally 0 to 255). The register number must be evenly divisible by 4.

Value Input. The doubleword to be written.

Returns

```
RC_SUCCESS
RC_PCI_NO_BIOS
RC_PCI_BAD_REGISTER_NUMBER
```

Remarks

None

6 System Unit Utilities

System unit utilities are a set of command-line-driven utilities used to initialize and examine the ARTIC960 adapter.

Although it is not shown in the syntax diagrams, help is provided for all utilities by using the **?** switch. The utilities display a brief message containing the syntax diagram for the utility. The utilities also display the help messages if no parameters or switches are entered, or if they are entered incorrectly.

All numeric parameters and command line options are assumed to be decimal values, unless otherwise noted. To pass a hexadecimal value for any numeric parameter, prefix the parameter with `0x` or `0X`. For example, `0x10` and `0X10` are equivalent to the decimal parameter of `16`.

The logical card numbers referred to by the utilities are assigned by the driver during installation.

The ARTIC960 utilities do not perform any special processing to handle the OS/2 <Ctrl><Break> or <Ctrl><C> program termination signals.

- If the user breaks out of a load operation, it may be necessary to unload a partially-loaded process or reset the card to continue.
- If the user breaks out of the Configuration utility or out of an active dump, a reset is required to continue.
- If the user breaks out of a dump while waiting with an exception trigger set, no action should be necessary.

Utilities that use input files use the following as search criteria to find the required file:

- Current directory
- RICPATH environment variable
- DPATH environment variable for OS/2 and Windows NT, and PATH environment variable in AIX and Windows NT.

If the file is not found using this search criteria, the appropriate error code is returned.

For all utilities, the length of the command line is limited to 256 bytes. All lines within files processed by the utilities are limited to a length of 256 bytes, including the end-of-line sequence. The number of entries within configuration files and parameter files is unlimited.

Application Loader (ricload) Utility

The Application Loader is a command-line-driven utility that loads processes onto the ARTIC960 adapter. The Application Loader does not require the presence or absence of any optional parameters to specify other optional parameters.

Arguments passed within quotes on the command line are passed as a single parameter. Extraneous quotes within the argument parameter are deleted. See *Examples of Application Loader Calls* on page 200. Blank lines within either a configuration or parameter file are discarded. Within a parameter file, the standard C end-of-line sequence is used to separate parameters.

Application Loader Syntax

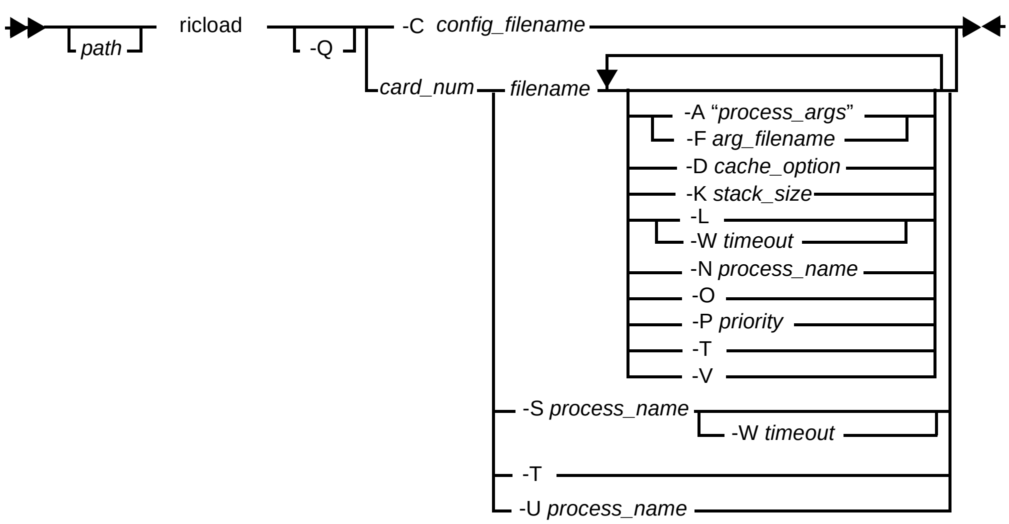


Figure 6-1. Application Loader Syntax

–Q
: Specifies quiet Application Loader operation. Normally, the Application Loader displays messages indicating a successful or unsuccessful operation on the standard output device. In quiet mode, the Application Loader does not display any messages.

–C *config_filename*
: Specifies that the configuration file *config_filename* contains a list of processes to be loaded. Each line in the configuration file is treated as an individual load request. If an error is encountered during the processing of the configuration file, the load operation is terminated and the remaining entries are not processed.

card_num
: Specifies the logical card number to be loaded.

filename
: Specifies the file containing the process to be loaded.

–A "*process_args*"

Specifies that the arguments in *process_args* (which is enclosed within a single set of quote marks), are to be passed to the process as argv[1] through argv[*n*].

The *process_args* arguments themselves must not contain the quote characters (" "). To use a quote mark within the arguments, use the –F *arg_filename* parameter. Using this parameter allows you to pass command line parameters in a file.

–F *arg_filename*

Specifies that the contents of the file *arg_filename* are passed to the process as argv[1] through argv[*n*]. Each line in the file is passed as a separate argv entry.

–D *cache_option*

Specifies data cache options for loading the process. The valid values are:

–D 0 Caching is not used (the default)

–D 1 Process stack is cached

–D 2 Process data section is cached

–D 3 Both the process stack and data sections are cached.

This option has no effect unless the i960 data cache is enabled. See the definition of `DATA_CACHE` on page 5.

–K *stack_size*

Specifies the size of the process stack in bytes. If this parameter is not specified, the kernel chooses its default stack size.

–L Specifies that the process is to be loaded but not started.

–W *timeout*

Specifies the time (in seconds) that the Application Loader waits for the loaded process to complete initialization. The Application Loader then outputs a message indicating the success or failure of that initialization. (See *CompleteInit—Mark Process as Completely Initialized* on page 23.)

The maximum timeout value is 64. If a failure occurs, the message contains the ErrorCode from the CompleteInit call. This option is automatically set by the Application Loader for all files beginning with "RIC_".

–N *process_name*

Specifies the name for the process being loaded. The process name is passed to the process as argv[0]. If this parameter is not specified, filename is passed as argv[0] (with the path information stripped). The length of the process name is limited to 16 characters including the NULL terminator.

–O Specifies creating an outfile for symbolic debugging. The outfile name is the *filename* with a file extension of .*out* instead of .*rel*. The file is created in the current directory. The intended use is to download the task that is not started (–L) and specify the –O switch. Then *filename*.out can be used with an 80960 interactive-computing environment (ICE) or a supported debugger.

–P *priority*
: Specifies that the process should be started with a priority level of *priority*. If this parameter is not specified, the kernel chooses a default priority level.

–T
: Specifies to set the time of day on the adapter. The system time is obtained and passed to the kernel on the ARTIC960 adapter.

–V
: Specifies to display verbose information about the loaded task. Displayed information includes the address of the task's entry point, code segment, data segment, BSS segment, stack address, and parameter address.

–S *process_name*
: Specifies that the process (previously loaded) is to be started.

–U *process_name*
: Specifies that the process should be unloaded.

To specify either a *config_filename*, *filename*, *process_name*, or *arg_filename* with spaces or special characters in the name, the name must be enclosed within quotes (“ ”). This allows support of the OS/2 high performance file system (HPFS).

The text files processed by the Application Loader (*config_filename* and *arg_filename*) are processed as a text stream. The ANSI C end-of-line and end-of-file sequence translation rules apply to these files.

Blank lines and comments in configuration files are ignored.

Application Loader Messages and Exit Codes

The contents of the message file used by all utilities are listed in *Appendix B: Message File* on page 295. The messages used by the Application Loader are listed in Table 6-1. The Application Loader also sets its exit code value to indicate the status of the load operation. The following table correlates the exit code of the Application Loader with the Application Loader messages.

Table 6-1. Application Loader Messages and Return Codes

Message Number	Exit Code	Notes
RIC0001	RC_UTIL_INVALID_CMDLINE_OPTION	
RIC0002	RC_UTIL_INVALID_CMDLINE_PARM	
RIC0003	RC_UTIL_FILE_NOT_FOUND	
RIC0004	RC_UTIL_FILE_ACCESS	
RIC0005	RC_UTIL_INVALID_CARD_NUMBER	
RIC0006	RC_UTIL_NO_MORE_MEM	
RIC0007	RC_UTIL_INVALID_NAME	
RIC0008	RC_UTIL_DUP_RES_NAME	
RIC0009	RC_UTIL_ADAPTER_EXCEPTION	
RIC0010	RC_UTIL_NO_ADAPTER_RESPONSE	
RIC0016	RC_UTIL_SYSTEM_ERROR	
RIC0019	RC_UTIL_NOT_INSTALLED	
RIC0022	RC_UTIL_SUCCESS	Successful unload process operation
RIC0023	RC_UTIL_NAME_NOT_FOUND	
RIC0024	RC_UTIL_SUCCESS	Successful start process operation
RIC0025	RC_UTIL_ALREADY_STARTED	
RIC0026	RC_UTIL_FILE_FORMAT	
RIC0027	RC_UTIL_WRNHELP_GIVEN	
RIC0035	RC_UTIL_INVALID_MICROCODE	Kernel not loaded first
RIC0037	RC_UTIL_MICROCODE_ERROR	Microcode error
RIC0038	RC_UTIL_ACCESS_ERROR	
RIC0042	RC_UTIL_PROC_MISMATCH	
RIC0044	RC_UTIL_PROC_DID_NOT_INIT	
RIC0045	RC_UTIL_PROC_INIT_ERROR	
RIC0052	RC_UTIL_TIMESET_ERROR	
RIC0053	RC_UTIL_SUCCESS	Successful start of System Clock
RIC0054	RC_UTIL_SUCCESS	Additional Information about task being loaded
RIC0057	RC_UTIL_SUCCESS	Successful load process operation

If the Application Loader is processing a configuration file and the entire file is processed successfully, the Application Loader returns a `RC_UTIL_SUCCESS`. If an error occurs during the processing of the file, the load operation terminates with an exit code corresponding to the error detected. If a POST error is detected during the loading of a configuration file and no subsequent errors are detected, the exit code `RC_UTIL_ADAPTER_EXCEPTION` is displayed.

Examples of Application Loader Calls

The following examples show different methods of using `ricload`.

Example—Load, Start, and Name a Process

This example loads and starts the process *c:\mydir\myproc.rel* to logical card 0 and assigns it a process name of *PROCess_1*.

```
ricload 0 c:\mydir\myproc.rel –N PROCess_1 –A "–C /T:'text'"
```

The parameters passed to the process are:

```
argv[0]  [PROCess_1]
argv[1]  [–C]
argv[2]  [/T:'text']
```

Example—Load and Start a Process with a Default Name

This example loads and starts the process *process.rel* to logical card 0. The process is named *process.rel* because a name was not specified.

```
ricload 0 process.rel –A "abc def ghi jkl mno"
```

The parameters passed to the process are:

```
argv[0]  [process.rel]
argv[1]  [abc]
argv[2]  [def]
argv[3]  [ghi]
argv[4]  [jkl]
argv[5]  [mno]
```

Example—Unload a Process

This example unloads the process *PROCess_1* from logical card 0.

```
ricload 0 –U PROCess_1
```

Example—Load a Process and Pass the Contents of a File

This example loads the process *\sub dir\proc FILE001.rel* to logical card 0. The process is not started. The contents of the file *parms.txt* are passed as parameters *argv[]*.

```
ricload 0x0 "\sub dir\proc FILE001.rel" –f parms.txt –L
```

The following shows the contents of the file *parms.txt* and the parameters passed to the process in *argv[]*.

Contents of File parms.txt

```
this is the FIRST line of parameters
  this is the second line
parameter 3
```

Parameters argv[]

```
argv[0]  [proc FILE001.rel]
argv[1]  [this is the FIRST line of parameters]
argv[2]  [  this is the second line]
argv[3]  [parameter 3]
```

Example—Load and Start a Process Using a Configuration File

The following shows the contents of the file *setup.cfg* and the resulting action. The entire load operation is done quietly (no messages displayed).

```
ricload –C setup.cfg –Q
```

Contents of File setup.cfg

```
****
* Setup configuration file
*
  0 ric_kern.rel –F ric_kern.cfg

  0 ric_base.rel
*
  0 ric_mcio.rel –F ric_mcio.cfg
  0 ric_scb.rel -F ric_scb.cfg
****
```

Resulting Action

1. The file *ric_kern.rel* is loaded to logical card 0 with *ric_kern.cfg* passed as its parameters. Then the process is started.
2. The file *ric_base.rel* is loaded to logical card 0 and started.
3. The file *ric_mcio.rel* is loaded to logical card 0, with *ric_mcio.cfg* passed as its parameters. Then the process is started.
4. The file *ric_scb.rel* is loaded to logical card 0 with *ric_scb.cfg* passed as its parameters. Then the process is started.

Dump Utility

The Dump utility dumps the state of the ARTIC960 adapter for diagnostic purposes. The Dump utility dumps all of the memory and I/O regions of the ARTIC960 adapter address space.

The Dump utility *does not* break a dump into pieces across several diskettes. The target drive must have the space necessary to capture the entire dump file or the dump fails.

The Dump utility compresses the dump data to minimize the size of the dump file. The Status utility handles dump file decompression. The Dump utility does not contain any user prompts, which enables it to run unattended.

The Dump utility has two modes of operation: triggered and immediate. In multitasking operating systems, running the Dump utility in triggered mode requires a dedicated session. While running in triggered mode, the Dump utility blocks on a triggering mechanism provided by the device drivers.

Dump Syntax

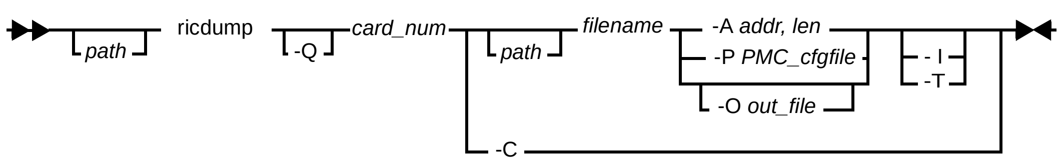


Figure 6-2. Dump Utility Syntax

–Q Specifies quiet dump operation. Normally, the Dump utility displays messages indicating a successful or unsuccessful operation on the standard output device. In quiet mode, no messages are displayed.

card_num Specifies the logical card number to be dumped.

filename Specifies the file into which the raw dump data is to be dumped. If the file already exists, it is overwritten.

–A Specifies an I/O region to be dumped. This option can be repeated up to four times. This option is not supported on the ARTIC960Rx PCI adapter.

addr Address of an I/O region to be dumped. No validity checking is done on this address.

len Length of an I/O region to be dumped. No validity checking is done on this length.

–P *PMC_cfgfile*

Specifies a PMC region to be dumped. Specifies that the configuration file *PMC_cfgfile* contains a list of addresses and lengths to dump. Each line in the configuration file is treated as an individual dump request.

The *PMC_cfgfile* can contain up to 31 lines of information. The following is an example of this configuration file. The first parameter in each line is the address to be dumped, and the second parameter is the amount of data to dump.

```
0x1ffa1000,40
0x1ffb2000,0x28
```

If an error is encountered during the processing of the configuration file, the dump operation is terminated and the remaining entries are not processed.

This option is not supported on ARTIC960 MCA and ARTIC960 PCI adapters.

–O *out_file*

Specifies that the dump output from –P option is written to a binary file named *out_file*. If *out_file* is not specified, the default file *pmcdump.bin* is created. If the file already exists, the file is overwritten.

Use a binary editor to view the file created with the –0 option.

–0 *out_file* is used only with –P *PMC_cfgfile*.

–I Specifies an immediate dump. This is the default mode of operation.

–T Specifies a triggered dump. The dump is triggered by an adapter exception.

–C Specifies the previously requested triggered dump should be canceled and the Dump utility should terminate and uninstall. A triggered dump cannot be canceled once the trigger has occurred.

PMC_cfgfile Dump File Header Structures

The –A option can be used to dump a daughter-card I/O region. This option is not needed if the daughter-card ROM or daughter-card device driver fills in its specific daughter-card I/O regions in the IORegions structure of ROMTable. Refer to the hardware technical reference for your adapter for more information on ROMTable. If the address specified by *addr* is not a valid Intel 960 local-bus card address for the length specified by *len*, a bus error may be generated, causing the system to halt.

To specify a filename with spaces or special characters in the name, the name must be enclosed within quotes. Quotes within a name are not supported.

Data begins at offset `0x00000200`.

```
typedef struct PMCFileHeader
{
   RIC_ULONG    MagicNo;   /* Magic No. to specify a dump file*/
   RIC_ULONG    HdrSize;   /* Indicates total header size     */
   RIC_ULONG    Regions;   /* Offset to next entry            */
   RIC_ULONG    TimeStamp; /* Time Date Stamp                */
   DUMPFILEHDR  DumpInfo[31];
}PMCFILEHEADER;

typedef struct DUMPFileHeader
{
   RIC_ULONG    AddressDump; /* Address to dump              */
   RIC_ULONG    LengthDump;  /* Amount to dump               */
   RIC_ULONG    Offset       /* Offset into the binary dump  */
                             /* file to locate information   */
                             /* for this entry               */
   RIC_ULONG    Reserved;
 }DUMPFILEHDR;
```

Example of a *PMC_cfgfile* Dump File

The following is an example of the binary file created with the –O option when viewed with a binary editor.

```
0x00000000: BAAB EDFE 0002 0000 0200 0000 7A08 1236
0x00000010: 0010 FA1F 2800 0000 0002 0000 0000 0000
0x00000020: 0010 1B1F 2500 0000 2802 0000 0000 0000
0x00000030: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000040: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000050: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000060: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000070: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000080: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000090: 0000 0000 0000 0000 0000 0000 0000 0000
0x000000A0: 0000 0000 0000 0000 0000 0000 0000 0000
0x000000B0: 0000 0000 0000 0000 0000 0000 0000 0000
0x000000C0: 0000 0000 0000 0000 0000 0000 0000 0000
0x000000D0: 0000 0000 0000 0000 0000 0000 0000 0000
0x000000E0: 0000 0000 0000 0000 0000 0000 0000 0000
0x000000F0: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000100: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000110: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000120: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000130: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000140: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000150: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000160: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000170: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000180: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000190: 0000 0000 0000 0000 0000 0000 0000 0000
0x000001A0: 0000 0000 0000 0000 0000 0000 0000 0000
0x000001B0: 0000 0000 0000 0000 0000 0000 0000 0000
0x000001C0: 0000 0000 0000 0000 0000 0000 0000 0000
0x000001D0: 0000 0000 0000 0000 0000 0000 0000 0000
0x000001E0: 0000 0000 0000 0000 0000 0000 0000 0000
0x000001F0: 0000 0000 0000 0000 0000 0000 0000 0000
0x00000200: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
0x00000210: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
0x00000220: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
0x00000230: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
0x00000240: FFFF FFFF FFFF FFFF FFFF FFFF FF
```

Dump Messages and Exit Codes

The contents of the message file used by all utilities are listed in *Appendix B: Message File* on page 295. The messages used by the Dump utility are listed in Table 6-2. The Dump utility also sets its exit code value to indicate the status of the dump operation. The following table correlates the exit code of the Dump utility with the dump messages.

Table 6-2. Dump Utility Messages and Exit Codes

Message Number	Exit Code	Notes
RIC0001	RC_UTIL_INVALID_CMDLINE_OPTION	
RIC0002	RC_UTIL_INVALID_CMDLINE_PARM	
RIC0003	RC_UTIL_FILE_NOT_FOUND	
RIC0004	RC_UTIL_FILE_ACCESS	
RIC0005	RC_UTIL_INVALID_CARD_NUMBER	
RIC0010	RC_UTIL_NO_ADAPTER_RESPONSE	
RIC0011	None	Information-only message
RIC0012	RC_UTIL_SUCCESS	Successful dump completion
RIC0013	None	Information-only message
RIC0014	RC_UTIL_SUCCESS	Successful dump cancellation
RIC0015	RC_UTIL_NOT_PENDING	
RIC0016	RC_UTIL_SYSTEM_ERROR	
RIC0019	RC_UTIL_NOT_INSTALLED	
RIC0028	RC_UTIL_WRNHELP_GIVEN	
RIC0038	RC_UTIL_ACCESS_ERROR	
RIC0040	RC_UTIL_ALREADY_STARTED	
RIC0056	None	Information-only message
RIC0075	RC_UTIL_INVALID_CMDLINE_OPTION	
RIC0080	RC_UTIL_UNSUPPORTED_OPTION	Warning message
RIC0082	RC_UTIL_UNSUPPORTED_OPT_ HARDWARE	Warning message
RIC0083	RC_UTIL_DUMP_PROCESS_ERROR	
RIC0084	None	
RIC0085	RC_UTIL_SUCCESS	Successful PMC dump completion
RIC0086	RC_UTIL_DUMP_CONFIG_ERROR	
RIC0087	RC_UTIL_PARM_SYNTAX_ERROR	Syntax for parameters incorrect

Configuration Utility

The Subsystem Control Block (SCB) Logical I/O Architecture specifies how units on the system bus communicate with one another. The SCB Configuration utility configures the move-mode SCB pipes between the ARTIC960 adapters and the system processor and between each ARTIC960 adapter. The SCB pipes should be configured before using Remote Mailbox services. The Configuration utility must be run after the ARTIC960 kernel, base subsystem, system-bus I/O subsystem, and SCB subsystem are loaded, but before any mailbox applications are loaded.

If you are going to use peer-to-peer communication, load the kernel with the `MAX_PEER_ADAPTERS` parameter equal to the number of peer SCB units. The kernel default is 0. In addition, if you are using the ARTIC960 Micro Channel adapter in an AIX environment, configure the adapter with the attribute DMA2Enable set to `YES` if peer-to-peer communication is needed. The attribute default is `NO`.

If an adapter is reset, this utility must be rerun.

Unless otherwise specified, a default pipe size is used. The default pipe size is 1024 bytes for a logical adapter pipe, and 2048 bytes for a system unit pipe. When a pipe size is specified, it must be a minimum of 128 bytes.

The Configuration utility prevents configuration of a pair of logical adapters if they are not physically able to communicate. If an adapter does not have a full memory window configured, other adapters cannot directly access it. If an adapter is in a 16-bit Personal System/2 (PS/2) slot and the window for the peer adapter is located above the 16 MB line, it cannot access the other adapter. The Configuration utility rejects both of these configurations with the error message RIC0041.

In AIX, the Configuration utility also prevents configuration of a pair of logical adapters if peer-to-peer activity is not supported on PCI adapters. The error message RIC0080 (“Warning: Unsupported option: *xxxxxxxx*”) is returned.

The system-unit-to-card SCB pipes have to be configured prior to configuring the card-to-card SCB pipes.

Configuration Syntax

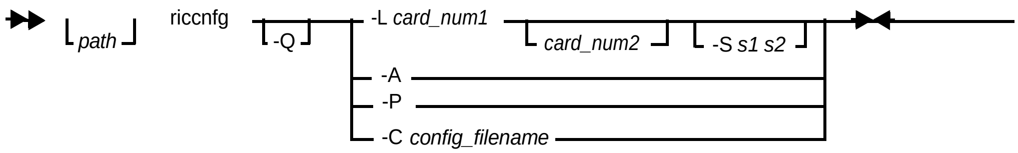


Figure 6-3. Configuration Utility Syntax

–Q Specifies quiet operation. Normally, the Configuration utility displays messages indicating the success of an operation on the standard output device. In quiet mode, no messages are displayed.

–L Specifies which logical cards are to be configured. A set of SCB delivery pipes are configured between logical *card_num1* and *card_num2*. If *card_num2* is not specified, it is assumed to be the system unit.

–S Specifies the size, in bytes, of the delivery pipe. The size *s1* corresponds to the size of the delivery pipe from *card_num1* to *card_num2*. The size *s2* corresponds to the size of the delivery pipe from *card_num2* to *card_num1*.

The minimum size for *s1* and *s2* is 128 bytes. If a size is not specified, the default size is used (1024 bytes for card-to-card pipes and 2048 bytes for system-unit-to-card pipes).

–C *config_filename*
Specifies that the contents of the file *config_filename* is to be used as input to the Configuration utility. Each line in the configuration file is treated as an individual configuration request. The format of the file is described in Figure 6-4.

To specify a *config_filename* with spaces or special characters in the name, the name must be enclosed within quotes (“ ”). Quotes within a name are not supported.

–A Specifies that all pipes (system unit/adapter and adapter/adapter) be configured using the default pipe sizes.

–P Specifies that system unit/adapter pipes be configured using the default pipe sizes.

Configuration File Entry Format

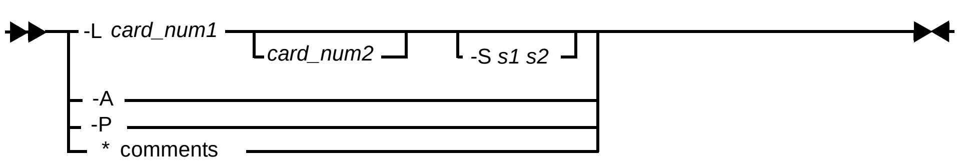


Figure 6-4. Configuration Utility File Entry Format

Blank lines and comments in configuration files are ignored.

Configuration Messages and Exit Codes

The contents of the message file used by all utilities are listed in *Appendix B: Message File* on page 295. The messages used by the Configuration utility are listed in Table 6-3. The Configuration utility also sets its exit code value to indicate the status of the configuration operation. The following table correlates the exit code of the Configuration utility with its messages.

Table 6-3. Configuration Utility Messages and Exit codes

Message Number	Exit Code	Notes
RIC0001	RC_UTIL_INVALID_CMDLINE_OPTION	
RIC0002	RC_UTIL_INVALID_CMDLINE_PARM	
RIC0003	RC_UTIL_FILE_NOT_FOUND	
RIC0004	RC_UTIL_FILE_ACCESS	
RIC0005	RC_UTIL_INVALID_CARD_NUMBER	
RIC0009	RC_UTIL_ADAPTER_EXCEPTION	
RIC0010	RC_UTIL_NO_ADAPTER_RESPONSE	
RIC0016	RC_UTIL_SYSTEM_ERROR	
RIC0019	RC_UTIL_NOT_INSTALLED	
RIC0029	RC_UTIL_WRNHELP_GIVEN	
RIC0036	RC_UTIL_SUCCESS	
RIC0037	RC_UTIL_MICROCODE_ERROR	Microcode error
RIC0038	RC_UTIL_ACCESS_ERROR	
RIC0041	RC_UTIL_PIPE_UNCONF	(PS/2 systems only)
RIC0043	RC_UTIL_PIPE_SIZE_OUT_OF_RANGE	
RIC0046	RC_UTIL_PIPE_ALREADY_CONF	
RIC0047	RC_UTIL_PIPE_CONF_FAILED	
RIC0055	RC_UTIL_UNIT_NOT_FUNCTIONING	
RIC0067	RC_UTIL_SNGL_PIPE_CONF_FAILED	
RIC0068	RC_UTIL_SUBSYSTEM_NOT_FOUND	
RIC0080	RC_UTIL_UNSUPPORTED_OPTION	

Reset Utility

The Reset utility allows users to reset ARTIC960 adapters. Multiple adapters can be reset with a single call of the Reset utility.

The Reset utility ensures that all other SCB units (system driver and adapters) are notified of the reset operation before resetting the card.

Reset Syntax

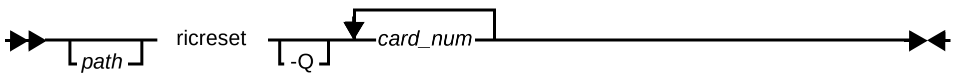


Figure 6-5. Reset Utility Syntax

–Q	Specifies quiet operation. Typically, the Reset utility displays messages indicating a successful or unsuccessful operation on the standard output device. In quiet mode, no messages are displayed.
card_num	Specifies the logical card number to be reset. If multiple adapters are specified, they are reset sequentially.

If multiple adapters are being reset with a single call, the Reset utility continues to the next adapter if an individual adapter reset fails or if an individual adapter number is invalid. The proper messages are generated for each adapter as its reset is done. If any errors are detected while resetting any of the adapters, the most severe error code is returned by the Reset utility. These exit codes from least to most severe are:

```
RC_UTIL_SUCCESS
RC_UTIL_INVALID_CARD_NUMBER
RC_UTIL_RESET_FAILED
RC_UTIL_NO_ADAPTER_RESPONSE
```

All other errors cause the Reset utility to end immediately with the proper error code.

Reset Messages and Exit Codes

The contents of the message file used by all utilities are listed in *Appendix B: Message File* on page 295. The messages used by the Reset utility are listed in Table 6-4. The Reset utility also sets its exit code value to indicate the status of the reset operation. The following table correlates the exit code of the Reset utility with the reset messages.

Table 6-4. Reset Utility Messages and Exit Codes

Message Number	Exit Code	Notes
RIC0001	RC_UTIL_INVALID_CMDLINE_OPTION	
RIC0005	RC_UTIL_INVALID_CARD_NUMBER	
RIC0009	RC_UTIL_RESET_FAILED	This message is followed by message RIC0034
RIC0010	RC_UTIL_NO_ADAPTER_RESPONSE	This message is followed by message RIC0034
RIC0019	RC_UTIL_NOT_INSTALLED	
RIC0031	RC_UTIL_WRNHELP_GIVEN	
RIC0032	None	Information-only message
RIC0033	RC_UTIL_SUCCESS	Successful reset operation
RIC0034	None	Exit code depends on preceding message (RIC009 or RIC0010)
RIC0038	RC_UTIL_ACCESS_ERROR	

Trace Utilities

The ARTIC960 kernel supports tracing of selected kernel services and user definable services. Three system unit utilities support tracing of services on the card: Set Trace, Get Trace, and Format Trace.

- Set Trace initializes, enables, and disables tracing of specified services.
- Get Trace reads the trace buffer (log) on the card and stores it on the system unit in a user-definable trace file.
- Format Trace formats the trace file into a user-readable format.

Set Trace Utility

The Set Trace utility initializes, enables, and disables tracing of various service classes on the ARTIC960 adapter. There is a defined set of kernel service classes that can be specified. The defined service classes are listed under *EnableTrace—Enable Tracing of Service Classes* on page 151. In addition, the user can define his own service classes.

The trace buffer must first be initialized before a service class can be enabled. An optional wrap count may be specified to set the maximum number of times the trace buffer can wrap. If a wrap count is not specified, the trace buffer wraps indefinitely. The wrap count is helpful in cases when tracing exceeds the trace buffer.

This utility can be used to initialize the trace buffer and enable a service class on the same command line prompt. Also, multiple, or all, service classes can be enabled and disabled on the same command line prompt.

Set Trace Syntax

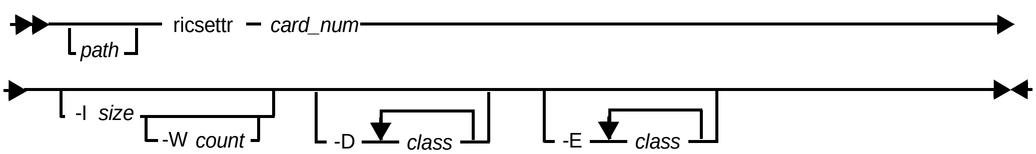


Figure 6-6. Set Trace Utility Syntax

card_num — Specifies the logical card number to be traced.

–I *size* — Specifies the size of the trace buffer (in KB) to be created and initialized on the adapter. Valid size range is 1 to 64.

–W *count* — Specifies the count after which the tracing should stop wrapping in the trace buffer. A count of –1 wraps the trace buffer infinitely.

–D *class* — Specifies the service classes for which tracing is to be disabled. Valid service classes are between 0 and 255. Service classes between 0 and 127 are reserved for kernel services. User-defined services classes are between 128 and 255. For performance reasons, the kernel does not perform any class range checking.

–E *class* — Specifies the service classes for which tracing is to be enabled. Valid service classes are between 0 and 255. Service classes between 0 and 127 are reserved for kernel services. For performance reasons, the kernel does not do any checking.

Set Trace Messages and Exit Codes

The contents of the message file used by all utilities are listed in *Appendix B: Message File* on page 295. The messages used by the Set Trace utility are listed in Table 6-5. The Set Trace utility also sets its exit code value to indicate the status of the set trace operation. The following table correlates the exit code of the Set Trace utility with the messages.

Table 6-5. Set Trace Utility Messages and Exit Codes

Message Number	Exit Code	Notes
RIC0001	RC_UTIL_INVALID_CMDLINE_OPTION	
RIC0002	RC_UTIL_INVALID_CMDLINE_PARM	
RIC0005	RC_UTIL_INVALID_CARD_NUMBER	
RIC0010	RC_UTIL_NO_ADAPTER_RESPONSE	
RIC0019	RC_UTIL_NOT_INSTALLED	
RIC0038	RC_UTIL_ACCESS_ERROR	
RIC0300	RC_UTIL_WRNHELP_GIVEN	
RIC0324	RC_UTIL_INVALID_CMDLINE_PARM	Invalid service class
RIC0326	RC_UTIL_SUCCESS	Successful set trace operation

Get Trace Utility

The Get Trace utility allows users to read data in the trace buffer on the RadiSys ARTIC960 adapter and store it in a file in binary form. The data in this file can be formatted by the Format Trace utility, discussed in section *Format Trace Utility* on page 217. Prior to running the Get Trace utility, use the Set Trace utility to initialize the trace buffer.

The trace buffer should not be read while tracing is active. Before reading the trace buffer, the Get Trace utility disables tracing of all services currently enabled, unless the –E option is specified. Tracing of services can be enabled again after the buffer is read by using the Set Trace utility.

Get Trace Syntax

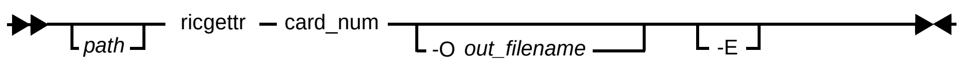


Figure 6-7. Get Trace Utility Syntax

card_num Specifies the logical card number to be traced.

–O *out_filename*

Specifies the name of the file in which data from the trace buffer is stored. If this option is not specified, the file **rictrace.bin** is created in the current directory.

–E Specifies that the trace buffer should be retrieved without first disabling the active tracing. This option should be used only when the trace cannot be retrieved otherwise, because the trace buffer could be updated as it is retrieved. This option can be used to recover the trace buffer from a card that has an exception condition. It should not be used on a card during active tracing.

Get Trace Messages and Exit Codes

The contents of the message file used by all utilities are listed in *Appendix B: Message File* on page 295. The messages used by the Get Trace utility are listed in Table 6-6. The Get Trace utility also sets its exit code value to indicate the status of the Get Trace operation. The following table correlates the exit code of the Get Trace utility with the messages.

Table 6-6. Get Trace Utility Messages and Exit Codes

Message Number	Exit Code	Notes
RIC0001	RC_UTIL_INVALID_CMDLINE_OPTION	
RIC0002	RC_UTIL_INVALID_CMDLINE_PARM	
RIC0004	RC_UTIL_FILE_ACCESS	
RIC0005	RC_UTIL_INVALID_CARD_NUMBER	
RIC0010	RC_UTIL_NO_ADAPTER_RESPONSE	
RIC0016	RC_UTIL_SYSTEM_ERROR	
RIC0019	RC_UTIL_NOT_INSTALLED	
RIC0038	RC_UTIL_ACCESS_ERROR	
RIC0301	RC_UTIL_WRNHELP_GIVEN	
RIC0302	RC_UTIL_SUCCESS	Successful Get Trace operation
RIC0303	None	Information message to run Format Trace utility
RIC0305	None	Trace is not initialized

Format Trace Utility

The Format Trace utility allows users to format the data obtained in a file by the Get Trace utility.

The formatted data consists of the fields of the Trace Control Block, which contains general information about the trace and the trace data of the service classes enabled by the Set Trace utility. The trace data is in the form of records to enhance readability.

The utility requires a service class file to correlate the service classes and procedure names to service class numbers and procedure IDs. The text file **ricclass.trc** has all the predefined kernel service classes and procedure names. This file must be present in the current directory or one of the directories defined in the RICPATH or DPATH (for OS/2) or PATH (for AIX) environment variables for the Format Trace utility to find it for trace formatting.

You have the option to specify a user trace file using the –C option. Your trace file must contain the same classes and procedure names that you have defined. You should use **ricclass.trc** as an example for the format of the file. Service class names must begin with C_ and procedure ID names must begin with P_.

If the Format Trace utility fails to find **ricclass.trc**, the warning message `RIC0003` is displayed. The trace file is formatted. However, the service class and procedure names do not appear in the output file. If the Format Trace utility fails to find the user-specified service class file, the message `RIC0003` is displayed and the Format Trace utility terminates with exit code `RC_UTIL_FILE_NOT_FOUND`.

The Format Trace utility fails with `RC_UTIL_FILE_FORMAT` if the binary data it formats is corrupted.

Format Trace Syntax

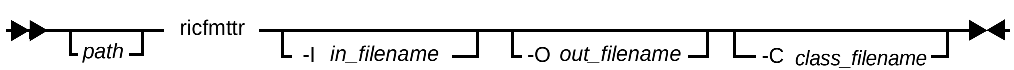


Figure 6-8. Format Trace Utility Syntax

–I *in_filename*
: Specifies the name of the file that contains data obtained by the Get Trace utility. The Format Trace utility formats the data in this file. If *in_filename* is not specified, the utility searches the current directory, then RICPATH followed by the DPATH (for OS/2) or PATH (for AIX) environment variables for **rictrace.bin**.

–O *out_filename*
: Specifies the name of the file for which the formatted information is stored. If the file already exists, the data in the file is overwritten. If *out_filename* is not specified, the formatted data is written to stdout.

–C *class_filename*
: Specifies the name of the file that contains the user's service class and procedure ID information.

Example of a Format Trace Call

The following example illustrates the use of format trace.

```
ricgettr 0
ricfmttr
```

This sample reads the trace buffer on card 0 and writes the formatted trace to stdout. A file **rictrace.bin** is created in the current directory.

Format Trace Messages and Exit Codes

The contents of the message file used by all utilities are listed in *Appendix B: Message File* on page 295. The messages used by the Format Trace utility are listed in Figure 6-6. The Format Trace utility also sets its exit code value to indicate the status of the Format Trace operation. The following table correlates the exit code of the Format Trace utility with the messages.

Table 6-7. Format Trace Utility Messages and Exit Codes

Message Number	Exit Code	Notes
RIC0001	RC_UTIL_INVALID_CMDLINE_OPTION	
RIC0002	RC_UTIL_INVALID_CMDLINE_PARM	
RIC0003	RC_UTIL_FILE_NOT_FOUND	
RIC0004	RC_UTIL_FILE_ACCESS	
RIC0026	RC_UTIL_FILE_FORMAT	
RIC0304	RC_UTIL_WRNHELP_GIVEN	
RIC0306	RC_UTIL_SUCCESS	Trace buffer is empty
RIC0307 – RIC0322, RIC0325	None	Format messages
RIC0323	RC_UTIL_SUCCESS	Successful Format Trace operation

Format Trace Record Details

The following figures depict the records displayed by the Format Trace utility.

The first record contains the trace control block information. Each successive record contains information for the required service classes. All displayed values are hexadecimal.

For examples, see *Format Trace Record Examples* on page 222.

Trace Control Block Record: The following is the first record in the formatted trace. A WrapAroundCount of `0xFFFFFFFF` indicates an infinite wrap count. Service Classes Enabled fields indicate which service classes were enabled at the time the ricgettr was called.

```
WrapAroundCount    = 0xnnnnnnnn
CurWrapAroundCount = 0xnnnnnnnn

Service Classes Enabled:

nnn nnn nnn nnn nnn nnn nnn nnn
nnn nnn
```

Figure 6-9. Trace Control Block

Record Description (Data in Bytes): In this record, the data is shown in bytes. The valid kernel ServiceClass and ProcedureID fields are obtained from the service class file **ricclass.trc** and the optional –C service class configuration file. The valid kernel CallerPosition strings are `PROCEDURE_ENTRY` and `PROCEDURE_EXIT`.

```
ServiceClass      =      0xnn                        ssssssssssssssss
ProcedureID       =      0xnn                        ssssssssssssssss
CallerPosition    =      0xnn                        ssssssssssssssss
DataSize          = 0xnnnnnnnn
ProcessInExec     =                                  ssssssssssssssss

Data Bytes:

nn nn nn nn nn nn nn nn - nn nn nn nn nn nn nn nn    cccccccccccccccc
nn nn nn nn                                          cccc
```

Figure 6-10. Record Description for a Service Class (Data in Bytes)

Record Description (Data in Words): In this record, the data is shown in words. The valid kernel ServiceClass and ProcedureID fields are obtained from the service class file **ricclass.trc** and the optional –C service class configuration file. The valid kernel CallerPosition strings are `PROCEDURE_ENTRY` and `PROCEDURE_EXIT`.

```
ServiceClass        =       0xnn                      ssssssssssssssss
ProcedureID         =       0xnn                      ssssssssssssssss
CallerPosition      =       0xnn                      ssssssssssssssss
DataSize            = 0xnnnnnnnn
ProcessInExec       =                                 ssssssssssssssss

Data Words:

nnnnnnnn nnnnnnnn nnnnnnnn nnnnnnnn
nnnnnnnn nnnnnnnn nnnnnnnn nnnnnnnn
nnnnnnnn nnnnnnnn nnnnnnnn nnnnnnnn
nnnnnnnn nnnnnnnn
```

Figure 6-11. Record Description for a Service Class (Data in Words)

Format Trace Record Examples

```
WrapAroundCount    = 0xFFFFFFFF
CurWrapAroundCount = 0x00000002

Service Classes Enabled:

  3   7
```

Figure 6-12. Trace Control Block Example

```
Service Class    =       0x07                  C_MAILBOX_SERVICE
Procedure ID     =       0x03                         P_OPENMBX
Caller Position  =       0x00                   PROCEDURE ENTRY
Data Size        = 0x00000005
Process In Exec  =                              PRC_rmblpreb.rel


Data Bytes:

4D 42 58 31 00                                         MBX1.
```

Figure 6-13. Record Description Example (Data in Bytes) Trace Record: 0x002E

```
Service Class    =       0x03                C_SEMAPHORE_SERVICE
Procedure ID     =       0x07                       P_RELEASESEM
Caller Position  =       0xFF                     PROCEDURE EXIT
Data Size        = 0x00000004
Process In Exec  =                                PRC_RIC_Mbx_SS


Data Words:

00000000
```

Figure 6-14. Record Description Example (Data in Words) Trace Record: 0x0033

Status Utility

The Status utility is a development tool used to examine the state of the ARTIC960 adapter. This utility can operate in the following states:

Live analysis
: Examines an active card in a system. (This is the default.)

Post analysis
: Examines a raw adapter dump file that was produced by the Dump utility.

The following are modes to control the display of card data:

Interactive mode
: The user can interactively request the display of specific data on the card. It uses the standard input (stdin) and standard output (stdout) devices. This is the default.

Status mode
: The utility displays a standard set of adapter structures to the standard output device. The mode is similar to the PSTAT command in OS/2. Run the Status utility in this mode using pipes. Type the following to call pipes:

```
ricstat <parameters> –S | more
```

The following items are displayed:

- Base hardware configuration (main menu option 1)
- The name, process ID, version, priority, and state of every process on the adapter (main menu option 2)
- The name, attributes, and owner of every resource on the adapter (main menu option 3)
- Exception conditions (main menu option 9)

Dump-format-mode
: The address space of the adapter is displayed on the standard output device. This mode displays all of the dumped adapter memory space in a form similar to the dump memory command in DOS. This format is intentionally raw to allow more advanced tools and utilities to scan the decompressed data while still enabling manual inspection of the dump data.

The following chart summarizes the options for using the Status utility.

Table 6-8. Status Utility Options

	Live Analysis	Post Analysis
Interactive mode	default	–F *dump_file*
Status mode	–S	–F *dump_file* –S
Dump-format mode	N/A	–D *dump_file*

Status Syntax

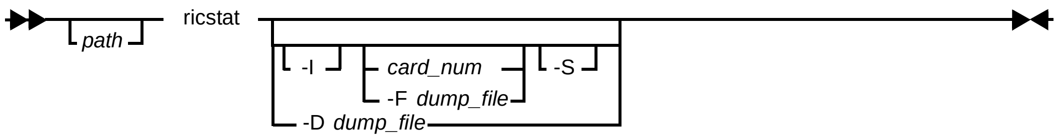


Figure 6-15. Status Utility Syntax

–I Specifies that all numeric prompts are decimal (the default is hexadecimal).

card_num Specifies the logical card number for live-analysis operation.

–F *dump_file*
Specifies a dump file for post-analysis operation.

–S Specifies non-interactive status mode.

–D *dump_file*
Specifies a dump file for dump-format mode.

To specify a *dump_file* with spaces or special characters in the name, the name must be enclosed within quotes (“ ”). Quotes within a name are not supported.

If no parameters are specified, the default is to prompt for card numbers in interactive live-analysis mode rather than to provide help. The card number is always interpreted as decimal.

Status Messages and Exit Codes

The contents of the message file used by all utilities are listed in *Appendix B: Message File* on page 295. The messages used by the Status utility are listed in Table 6-9. The Status utility also sets its exit code value to indicate the status of the operation. The following table correlates the exit code of the Status utility with the utility messages.

The menus, prompts, and displays used by the Status utility in interactive mode follow those shown in *Status Interactive Messages* on page 227.

Table 6-9. Status Utility Messages and Exit Codes

Message Number	Return Code	Notes
RIC0001	RC_UTIL_INVALID_CMDLINE_OPTION	
RIC0002	RC_UTIL_INVALID_CMDLINE_PARM	
RIC0003	RC_UTIL_FILE_NOT_FOUND	
RIC0004	RC_UTIL_FILE_ACCESS	
RIC0005	RC_UTIL_INVALID_CARD_NUMBER	
RIC0010	RC_UTIL_NO_ADAPTER_RESPONSE	
RIC0016	RC_UTIL_SYSTEM_ERROR	
RIC0019	RC_UTIL_NOT_INSTALLED	
RIC0026	RC_UTIL_FILE_FORMAT	
RIC0030	RC_UTIL_WRNHELP_GIVEN	
None	RC_UTIL_SUCCESS	Normal exit
RIC0038	RC_UTIL_ACCESS_ERROR	
RIC0100–RIC0299	None	Interactive messages, menus, and prompts

Status Dump Format

The following shows the format of data displayed when using the dump-format mode of the Status utility.

```
rraaaaaaaa hh hh hh hh hh hh hh hh-hh hh hh hh hh hh hh hh cccccccccccccccc
```

where:

rr Is either ’=>’ to indicate repeated blocks of data or ’ ’ to indicate a new block of data.

aaaaaaaa Is the 32-bit address of this 16-byte block of data

hh Is the hexadecimal value of each byte in the block.

c Is the ASCII representation of each printable character in the block, or a period (.) if the character is not printable.

The Status utility displays all of the memory address space contained in the dump file. Gaps in memory address space are shown as a blank line. See Figure 6-16 for an example of a formatted dump.

Example of a Formatted Dump

This example shows:

- Two unique blocks of data at addresses `00100000` through `0010002F`, followed by a block of `FF`s from address `00100030-001014FF`.
- The memory address `00101410` through `0010141F` is the other unique block.
- The block from `00101420–0010200F` is all `FF`s.
- The block from `00102010` through `0010210F` contains a repetitive string.
- The blank line indicates a gap in memory (from `00102110–001FFFFF`) followed by a 16-byte block of `00`s.

```
      .
      .
  00100000  30 31 32 33 34 35 36 37-41 42 43 44 61 62 63 64    01234567ABCDabcd
  00100010  0D 0A 24 23 40 21 00 00-00 00 00 00 61 61 61 61    ..$#@!......aaaa
  00100020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
=>00101400  FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF    ................
  00101410  61 20 64 75 6D 6D 79 20-70 61 74 74 65 72 6E 20    a dummy pattern
=>00102000  FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF    ................
=>00102100  01 01 01 01 01 01 01 01-30 30 30 30 31 31 31 31    ........00001111
      .
      .
  00200000  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
      .
```

Figure 6-16. Sample Formatted Dump

Status Interactive Messages

The following figures depict all of the menus, prompts, and displays of the Status utility in interactive mode. In these figures:

- All displayed values are in hexadecimal
- All numeric prompts are assumed to be hexadecimal, unless preceded by "0d"
- The full hexadecimal width of numerical values is always displayed
- If a default adapter number is passed on the command line, the “Enter adapter number” prompt does not appear and the default is used.

When responding to interactive prompts, any invalid data (entering "Z" at an adapter number prompt, for example) causes the invalid input message to be displayed, followed by a re-prompt for data.

When displaying data, the status utility keeps track of the number of lines displayed and the number of lines on the screen when the Status utility is initiated. If the number of data lines would cause the displayed data to scroll off the screen, press Enter to continue. Also, when displaying structures that have lists of data that may potentially be corrupted or very long, the Status utility allows the user to abort the display of the list by pressing the “q” key while the list is being displayed. The scroll feature is disabled if the number of lines in the current window is less than 25.

The following defines the characters used in the data:

n represents numeric data

s represents string data

t represents a memory type

Main Menu

This is the main menu for the Status utility.

```
 0)  Quit
 1)  Configuration
 2)  Process summary
 3)  Resource summary
 4)  Memory
 5)  Process details
 6)  Process resources
 7)  Process parameters
 8)  Resource details
 9)  Exception conditions
10)  VPD Information
11)  80960 registers

Enter item for display =>
```

Figure 6-17. Status Utility Main Menu

The following sections explain each of the options on the Status utility main menu.

1) Configuration on page 229

2) Process Summary on page 230

3) Resource Summary on page 231

4) Memory on page 232

5) Process Details on page 233

6) Process Resources on page 237

7) Process Parameters on page 238

8) Resource Details on page 239

9) Exception Conditions on page 250

10) VPD Information on page 251

11) 80960 Registers on page 252. This option is displayed only if the Status utility is called on a dump file with the -F switch.

Vector details are available only when specified by name or number. See *Vector Resource Details* on page 249 for an explanation of the displayed information. See Figure 6-63 on page 260 for an example.

1) Configuration

The following screen shows the prompts and items displayed when the Configuration option is chosen from the main menu.

- If a memory window is not configured, its data line is not displayed.
- On OS/2 systems with PCI cards, the Slot Number field displays `FF`.
- Valid values for Bus Type are MCA or PCI.
- Valid values for Interface Chip are Miami, MiamiP2P, or Rx.
- Valid values for Data Cache HW are Present or Not Present.
- To the right of the AIB ID is a descriptive AIB name.
  - If the daughter-card type is a PMC and the card is not present, `0X00000000 ( )` is displayed.
  - If a PMC card is present, `0XFFFFFFFF` (PMC Adapter Present) is displayed.
- The Total memory size is first shown in bytes. To the right of the size in bytes is the memory size converted to megabytes.

See Figure 6-44 on page 253 for an example.

```
Enter adapter number =>
Slot number                  nn
Card ID                      nnnnnnnn
Bus Type                     sss
Interface Chip               ssss
Data Cache HW                sssssss
Base I/O address             nnnn
Interrupt level              nn
AIB ID                       nnnnnnnn (sssssssssss)
Full window address          nnnnnnnn
Total memory size            nnnnnnnn (n.n MB)
Available memory             nnnnnnnn

Memory Region       Size                Type
-------------       ----                ----
nnnnnnnn            nnnnnnnn (n.n MB)   tttttttttttttttttttttt
nnnnnnnn            nnnnnnnn (n.n MB)   tttttttttttttttttttttt

-- Press Enter to continue --
```

Figure 6-18. Status Utility Configuration Display

2) Process Summary

The following screen shows the prompts and items displayed when the Process Summary option is chosen from the main menu. The output line is repeated for each process running on the adapter. Valid states are:

loaded
queued
blocked
suspended
stopped
driver
waiting on PMRq
expired

See Figure 6-45 on page 253 for an example.

```
Enter adapter number =>

     Name             ID      Version   Priority  State
----------------  --------  --------  --------  -----
ssssssssssssssss  nnnnnnnn  nnnnnnnn  nnnnnnnn  sssssssss
-- Press Enter to continue --
```

Figure 6-19. Status Utility Process Summary Display

3) Resource Summary

The following screen shows the prompts and items displayed when the Resource Summary option is chosen from the main menu. The output line is repeated for each resource on the adapter. Valid resource types are:

semaphore
event
memory
timer
queue
mailbox
signal
vector
driver
hardware device

See Figure 6-46 on page 254 for an example.

```
Enter adapter number =>

     Name         Type
----------------  ----
ssssssssssssssss  sssssssss
 -- Press Enter to continue --
```

Figure 6-20. Status Utility Resource Summary Display

4) Memory

The following screen shows the prompts and items displayed when the Memory option is chosen from the main menu.

- The address of the card can be specified by the local card address or the memory name.
- By entering a B or W at the prompt, the data can be displayed two different ways: byte mode or word mode.
  - If the byte mode (B) is chosen, the three groups displayed are:
    - Address
    - Hexadecimal value of each byte
    - ASCII representation of each byte (if the character is not a printable character, a period is displayed)
  - If the word mode (W) is chosen, the two groups displayed are:
    - Address
    - Hexadecimal value of each word
- If the address was previously entered and a NULL value was entered at the memory address prompt, the Status utility continues to display the memory.
- The output line is repeated as necessary to display all data.

See Figure 6-47 on page 254 for an example.

```
Enter adapter number =>

Enter [Address|Name][Length][B|W] Or 0 to Return =>

  aaaaaaaa  hh hh hh hh hh hh hh hh-hh hh hh hh hh hh hh hh   cccccccccccccccc
OR
  aaaaaaaa  hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
```

Figure 6-21. Status Utility Memory Display

5) Process Details

The following screen shows the prompts and items displayed when the Process Details option is chosen from the main menu.

Valid process states are:

loaded
queued
blocked
driver
suspended
wait on PMRq
stopped
expired

Valid process types are:

normal
driver
subsystem
kernel

An *expired* process is one that has been *stopped* and unloaded from the adapter. If a process is in the stopped or expired state, the Process Details submenu also shows termination status information. A process can be terminated because of the following events:

Process Terminated by Software Event on page 234
Process Terminated by Processor Event on page 235
Process Terminated by Adapter Event on page 236

See Figure 6-48 on page 254 for an example.

```
Enter adapter number =>
Enter process name or ID =>

Name                  ssssssssssssssss
ID                    nnnnnnnn
Priority              nnnnnnnn
Entry point           nnnnnnnn
Stack pointer         nnnnnnnn
Param pointer         nnnnnnnn
State                 sssssssssss
Version               nnnnnnnn
Type                  sssssssssss
```

Figure 6-22. Status Utility Process Details Display

Process Terminated by Software Event: The following screen shows prompts and items displayed when the Process Details option is chosen from the main menu, the process is in a stopped or expired state, and the termination code is software.

See Figure 6-49 on page 255 for an example.

```
Enter adapter number =>
Enter process name or ID =>

Name               ssssssssssssssss
ID                 nnnnnnnn
Priority           nnnnnnnn
Entry point        nnnnnnnn
Stack pointer      nnnnnnnn
Param pointer      nnnnnnnn
State              sssssssssss
Version            nnnnnnnn
Type               sssssssssss

Termination Code   software
Requester Id       nnnnnnnn
Source Of Req      ssssssssssssssss
Error Code         nnnnnnnn
```

Figure 6-23. Process Details Display—Process Terminated by Software Event

The valid values for Source Of Req are shown in Table 6-10.

Table 6-10. Source of Request

Source of Request Value	Meaning
Local	Request came from a process on the local adapter.
Remote	Request came through a kernel mailbox command from either the local adapter or a peer unit.
System Unit Command	Request came from the system unit through a command (probably issued using ricload with the -U parameter).

Process Terminated by Processor Event: The following screen shows the prompts and items displayed when the Process Details option is chosen, the process is in a stopped or expired state, and the termination code is processor.

See Figure 6-50 on page 255 for an example.

```
Enter adapter number =>
Enter process name or ID =>

Name              ssssssssssssssss
ID                nnnnnnnn
Priority          nnnnnnnn
Entry point       nnnnnnnn
Stack pointer     nnnnnnnn
Param pointer     nnnnnnnn
State             sssssssssss
Version           nnnnnnnn
Type              sssssssssss

Termination Code  processor
Fault Type        ssssssssssssssss
SubType           ssssssssssssssss
Code Address      nnnnnnnn
```

Figure 6-24. Process Details Display—Process Terminated by Processor Event

Table 6-11 shows the Fault Type and SubType values.

Table 6-11. Termination Status Fault Types and Subtypes for a Processor Event

Fault Type	Fault Subtype	Notes
Parallel	Parallel faults occurred	
Trace	Instruction	
	Branch	
	Call	
	Return	
	PreReturn	
	Supervisor	
	Breakpoint	
Operation	Invalid Opcode	Operation Unaligned is 80960CA specific extension
	Unimplemented	
	Unaligned	
	Invalid Operand	
Arithmetic	Integer Overflow	
	Arithmetic Zero-Divide	
Constraint	Constraint Range	
	Privileged	
Protection	Length	
Type	Type Mismatch	
Reserved	Reserved	Reserved

Process Terminated by Adapter Event: The following screen shows the prompts and items displayed when the Process Details option is chosen from the main menu, the process is in a stopped or expired state, and the termination code is Adapter.

Valid values for Trap Type are: memory violation and processor.

See Figure 6-51 on page 256 for an example.

```
Enter adapter number =>
Enter process name or ID =>

Name                ssssssssssssssss
ID                  nnnnnnnn
Priority            nnnnnnnn
Entry point         nnnnnnnn
Stack pointer       nnnnnnnn
Param pointer       nnnnnnnn
State               sssssssssss
Version             nnnnnnnn
Type                sssssssssss

Termination Code    adapter
Trap Type           ssssssssssssssss
Memory Address      nnnnnnnn
Code Address        nnnnnnnn
```

Figure 6-25. Process Details Display—Process Terminated by Adapter Event

6) Process Resources

The following screen shows the prompts and items displayed when the Process resources option is chosen from the main menu. The display format is identical to the Resource summary on page 231, except that only the resources for the selected process' resources are displayed. The output line is repeated for each resource.

See Figure 6-52 on page 256 for an example.

```
Enter adapter number =>
Enter process name or ID =>

     Name           Handle   Type
----------------  --------  ----
ssssssssssssssss  nnnnnnnn  sssssssss
```

Figure 6-26. Status Utility Process Resources Display

7) Process Parameters

The following screen shows the prompts and items displayed when the Process parameters option is chosen from the main menu. The output line is repeated to display all parameters.

See Figure 6-53 on page 257 for an example.

```
Enter adapter number =>
Enter process name or ID =>


argv[nn] = "sssss"
```

Figure 6-27. Status Utility Process Parameters Display

8) Resource Details

The format of the Resource details display depends on the individual resource.

If a resource name without the resource type prefix is specified on the main menu, the following menu is displayed and you are asked to indicate the resource type.

```
0) Return to previous menu
1) Device Driver
2) Event
3) Mailbox
4) Memory
5) Queue
6) Semaphore
7) Signal
8) Timer
9) Hardware Device

Enter the resource type =>
```

Figure 6-28. Resource Details Submenu

The following sections explain each of the options on the Resource details submenu.

1) Device Driver (Resource Details Submenu) on page 240

2) Event (Resource Details Submenu) on page 241

3) Mailbox (Resource Details Submenu) on page 242

4) Memory (Resource Details Submenu) on page 243

5) Queue (Resource Details Submenu) on page 244

6) Semaphore (Resource Details Submenu) on page 245

7) Signal (Resource Details Submenu) on page 246

8) Timer (Resource Details Submenu) on page 247

9) Hardware Device (Resource Details Submenu) on page 248

Vector details are available only when specified by name or number. See *Vector Resource Details* on page 249 for an explanation of the displayed information. See Figure 6-63 on page 260 for an example.

1) Device Driver (Resource Details Submenu): The following screen shows the prompts and items displayed when a device driver is selected from the Resource details submenu. The entries in the Access list fields show all processes that have access to the resource.

See Figure 6-54 on page 257 for an example.

```
Enter adapter number =>
Enter resource name or handle =>

Resource type   driver
Driver name     ssssssssssssssss
Process name    ssssssssssssssss
Protection      sssssssss

Access list:

Proc No       Process Name                Handle
--------      ----------------            --------
nnnnnnnn      ssssssssssssssss            nnnnnnnn
```

Figure 6-29. Device Driver Detail Display

2) Event (Resource Details Submenu): The following screen shows the prompts and items displayed when an event is selected from the Resource details submenu. The entries in the Semaphores field show each semaphore in the event. The entries in the Access list fields show all processes that have access to the resource.

See Figure 6-55 on page 257 for an example.

```
Enter adapter number =>
Enter resource name or handle =>

Resource type   event
Name            ssssssssssssssss
Semaphores      ssssssssssssssss
                ssssssssssssssss

Access list:

Proc No     Process Name      Handle
--------    ----------------  --------
nnnnnnnn    ssssssssssssssss  nnnnnnnn
```

Figure 6-30. Event Detail Display

3) Mailbox (Resource Details Submenu): The following screen shows the prompts and items displayed when a mailbox is selected from the Resource details submenu.

- Valid mailbox types are: local, global, and remote.
- The Name field is the name of the memory associated with the mailbox. If no memory is associated with the mailbox, nothing is displayed in this field.
- The entries in the Access list fields show all processes that have access to the resource.
- If the mailbox is empty, the string "<empty>" is displayed on the Messages line. Otherwise, the first 16 bytes of each mailbox element are displayed in the standard memory-display format.

See Figure 6-56 on page 258 for an example.

```
Enter adapter number =>
Enter resource name or handle =>

Resource type   mailbox
Name            ssssssssssssssss
Type            ssssss
Receiver        ssssssssssssssss
Semaphore       nnnnnnnn

Access list:

Proc No     Process Name    Handle    Memory Name
--------    --------------  --------  ----------------
nnnnnnnn  ssssssssssssssss  nnnnnnnn  ssssssssssssssss

Messages:       sssssss
  aaaaaaaa  hh hh hh hh hh hh hh hh-hh hh hh hh hh hh hh hh    cccccccccccccccc
```

Figure 6-31. Mailbox Detail Display

4) Memory (Resource Details Submenu): The following screen shows the prompts and items displayed when a memory is selected from the Resource details submenu.

- Valid strings for the AIB DMA Access and Mchl Access fields are: R/W, R/O, W/O, or none.
- Valid strings for the Sharable field are: yes or no.
- The entries in the Access list fields show all processes that have access to the resource.

See Figure 6-57 on page 258 for an example.

```
Enter adapter number =>
Enter resource name or handle =>

Resource type   memory
Name            ssssssssssssssss
Address         nnnnnnnn
Size            nnnnnnnn
AIB DMA Access  sss
Mchl Access     sss
Sharable        sss

Access list:

Proc No     Process Name      Handle    Access
--------    ----------------  --------  ------
nnnnnnnn    ssssssssssssssss  nnnnnnnn   sss
```

Figure 6-32. Memory Detail Display

5) Queue (Resource Details Submenu): The following screen shows the prompts and items displayed when a queue is selected from the Resource details submenu.

- The entries in the Access list fields show all processes that have access to the resource.
- The Semaphore field is the handle of the semaphore associated with the queue.
- If the queue is empty, the string "<empty>" is displayed on the "Elements" line. Otherwise, the first 16 bytes of each queue element are displayed in the standard memory-display format.

See Figure 6-58 on page 259 for an example.

```
Enter adapter number =>
Enter resource name or handle =>

Resource type   queue
Name            ssssssssssssssss

Access list:

Proc No    Process Name      Handle    Semaphore
--------   ----------------  --------  --------
nnnnnnnn   ssssssssssssssss  nnnnnnnn  nnnnnnnn

Elements:       sssssss
  aaaaaaaa  hh hh hh hh hh hh hh hh-hh hh hh hh hh hh hh hh    cccccccccccccccc
```

Figure 6-33. Queue Detail Display

6) Semaphore (Resource Details Submenu): The following screen shows the prompts and items displayed when Semaphore is selected from the Resource details submenu. The entries in the Access list fields show all processes that have access to the resource.

See Figure 6-59 on page 259 for an example.

```
Enter adapter number =>
Enter resource name or handle =>

Resource type   semaphore
Name            ssssssssssssssss
Count           nnnnnnnn

Access list:

Proc No     Process Name     Handle
--------    --------------   --------
nnnnnnnn    ssssssssssssssss nnnnnnnn
```

Figure 6-34. Semaphore Detail Display

7) Signal (Resource Details Submenu): The following screen shows the prompts and items displayed when a signal is selected from the Resource details submenu.

- Valid signal options are: always, match, and sender.
- The entries in the Access list fields show all processes that have access to the resource.
- The Entry field is empty if the Option field is *sender*.
- The Key field is ignored unless the Option field is *match*.

See Figure 6-60 on page 259 for an example.

```
Enter adapter number =>
Enter resource name or handle =>

Resource type   signal
Name            ssssssssssssssss

Access list:

Proc No     Process Name     Handle    Entry      Key      Option
--------    ---------------  --------  --------   ------   --------
nnnnnnnn  ssssssssssssssss  nnnnnnnn  nnnnnnnn  nnnnnnnn  ssssssss
```

Figure 6-35. Signal Detail Display

8) Timer (Resource Details Submenu): The following screen shows the prompts and items displayed when a timer is selected from the Resource details submenu.

Valid timer states are: running, stopped, and expired.

See Figure 6-61 on page 260 for an example.

```
Enter adapter number =>
Enter resource name or handle =>

Resource type   timer
Name            ssssssssssssssss
Handle          nnnnnnnn
Handler         nnnnnnnn
State           sssss
Owner name      ssssssssssssssss
Owner no        nnnnnnnn
```

Figure 6-36. Timer Detail Display

9) Hardware Device (Resource Details Submenu): The following screen shows the prompts and items displayed when a hardware device is selected from the Resource details option of the main menu.

- The values for the Valid data field are: yes and no.
- The Owner name and Owner no fields may be blank if the device is not allocated.
- The Device data fields are displayed only if the valid data flag indicates that it is available.

See Figure 6-62 on page 260 for an example.

```
Enter adapter number =>
Enter resource name or handle =>

Resource type   hardware device
Name            ssssssssssssssss
Status          nn
Valid data      sss
Owner name      ssssssssssssssss
Owner no        nnnnnnnn

Device data:
  aaaaaaaa  hh hh hh hh hh hh hh hh-hh hh hh hh hh hh hh hh    cccccccccccccccc
```

Figure 6-37. Hardware Device Detail Display

Vector Resource Details: The following screen shows the prompts and items displayed when a vector resource number or name is specified on the main menu.

- The entries in the Access list fields show all processes that have access to the resource.
- Valid values for the Protection field are: enabled and disabled.
- Valid values for the Return Code field are: yes and no.

See Figure 6-63 on page 260 for an example.

```
Enter adapter number =>
Enter resource name or handle =>

Resource type   vector
Vector          nnnnnnnn

Access list:

Proc No  Process Name     Handle   Handler    Protection  Return Code
-------  ------------     ------   -------    ----------  ----------
nnnnnnnn ssssssssssssssss nnnnnnnn nnnnnnnnnn ssssssssss  ssssssssss
```

Figure 6-38. Vector Detail Display

9) Exception Conditions

The following screen shows the prompts and items displayed when the Exception conditions option is chosen from the main menu.

- The exception code is interpreted and a descriptive string is displayed if the exception is a predefined exception condition. Table 6-12 lists the recognized exceptions.
- The entries in the Exception data fields show the data for all exception conditions. However, the Exception data fields are not displayed if the exception code indicates that no exception-condition is present.

See Figure 6-64 on page 261 for an example.

```
Enter adapter number =>

Exception code = nnnnnnnn (ssssssssssssssss)

Exception data:
  aaaaaaaa  hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh

-- Press Enter to continue --
```

Figure 6-39. Status Utility Exception Conditions Display

Table 6-12. Recognized Exception Conditions

Fault Type	Exception
Processor Fault	Operation
Processor Fault	Arithmetic
Processor Fault	Constraint
Processor Fault	Type Mismatch
Adapter Fault	Watchdog Timeout
Adapter Fault	Parity
Adapter Fault	80960 Memory Protection Violation
Adapter Fault	System Bus Master Memory Protection Violation
Adapter Fault	AIB Memory Protection Violation
Adapter Fault	Async No More Resources
Adapter Fault	Invalid Interrupt
Adapter Fault	Processor
Adapter Fault	NMI Interrupt
Adapter Fault	PLX Interrupt
Adapter Error	Power On Self Test Failure
Software Error	Data Corruption
Software Error	Adapter POST Failure
Software Error	System Bus I/O Subsystem Failure
Software Error	SCB Subsystem Failure
Software Error	External Mailbox Failure

10) VPD Information

The following submenu shows the available selections to display Vital Product Data (VPD) information when the VPD information option is chosen from the main menu.

- For each selection, the resulting screen format is the same.
- Selection 2 is not displayed if the attached card is a PMC card.

```
 0) Previous Menu
 1) Base ROM VPD Information
 2) AIB ROM VPD Information

Enter item for display =>
```

Figure 6-40. Status Utility VPD Information Display

The following screen shows the VPD information contained in the ROM for Intel-based systems.

```
   Displayable Message   ssssssssssssssssssssssssssssss
   Adapter Type          nn
   Part Number           nnnnnnnnnnnn
   FRU Number            nnnnnnnnnnnn
   Serial Number         nnnnnnnn
   Manufacturer ID       nnnnnnnnnn
   EC Level              nnnnnnnnnnnn
   ROS Level and ID      n.n

-- Press Enter to continue --
```

Figure 6-41. Displayed VPD Information for Intel-based Systems

The following screen shows the VPD information contained in the ROM for RISC System/6000.

```
   Displayable Message   ssssssssssssssssssssssssssssss
   Adapter Type          nn
   Part Number           nnnnnnnnnnnn
   FRU Number            nnnnnnnnnnnn
   Serial Number         nnnnnnnn
   Manufacturer ID       nnnnnnnnnn
   EC Level              nnnnnnnnnnnn
   Device Driver Level   n.n
   Diagnostic Level      n.n
   Loadable Microcode    nn.nn
   ROS Level and ID      n.n

-- Press Enter to continue --
```

Figure 6-42. Displayed VPD Information for RISC System/6000

11) 80960 Registers

The following screen shows the prompts and items displayed when the 80960 registers option is chosen from the main menu. This option is available only if the Status utility is called on a dump file using the -F switch.

See Figure 6-67 on page 262 for an example.

```
Enter adapter number =>

g0  = nnnnnnnn          r0  = nnnnnnnn (pfp)
g1  = nnnnnnnn          r1  = nnnnnnnn (sp)
g2  = nnnnnnnn          r2  = nnnnnnnn (rip)
g3  = nnnnnnnn          r3  = nnnnnnnn
g4  = nnnnnnnn          r4  = nnnnnnnn
g5  = nnnnnnnn          r5  = nnnnnnnn
g6  = nnnnnnnn          r6  = nnnnnnnn
g7  = nnnnnnnn          r7  = nnnnnnnn
g8  = nnnnnnnn          r8  = nnnnnnnn
g9  = nnnnnnnn          r9  = nnnnnnnn
g10 = nnnnnnnn          r10 = nnnnnnnn
g11 = nnnnnnnn          r11 = nnnnnnnn
g12 = nnnnnnnn          r12 = nnnnnnnn
g13 = nnnnnnnn          r13 = nnnnnnnn
g14 = nnnnnnnn          r14 = nnnnnnnn
g15 = nnnnnnnn (fp)     r15 = nnnnnnnn

sf0 = nnnnnnnn (SSSS)
sf1 = nnnnnnnn (SSSS)
sf2 = nnnnnnnn (SSSS)
sf3 = nnnnnnnn (SSSS)
sf4 = nnnnnnnn (SSSS)

ip  = nnnnnnnn
ac  = nnnnnnnn
pc  = nnnnnnnn
tc  = nnnnnnnn

fp0 = nnnnnnnnnnnnnnnnnnnn    fp2 = nnnnnnnnnnnnnnnnnnnn
fp1 = nnnnnnnnnnnnnnnnnnnn    fp3 = nnnnnnnnnnnnnnnnnnnn

-- Press Enter to continue --
```

Figure 6-43. Status Utility 80960 Registers Display

- The following SF registers are displayed, depending on the adapter.

Adapters	SF Registers Displayed				
	sf0	sf1	sf2	sf3	sf4
ARTIC960 and ARTIC960 PCI	y	y	y	n	n
ARTIC960Rx and ARTIC960RxD	y	y	n	y	n

- The lines fp0–fp3 are displayed only on adapters with an 80960 processor that supports floating-point operations.

Examples of Interactive Messages

The following examples all assume that the adapter number has been passed on the command line as a default.

1) Configuration

This example shows an Rx card with one window.

```
Slot number                0xFF
Card ID                    0x801014
Bus Type                   PCI
Interface Chip             Rx
Data Cache HW              Not Present
Base I/O address           0x0000
Interrupt level            0xA
AIB ID                     0xFFFFFFFF (PMC Adapter Present)
Full window address        0xFDC00000
Total memory size          0x00400000 (4.0 MB)
Available memory           0x003B8000

Memory Region    Size                         Type
-------------    ----                         ----
0xA0000000       0x00400000 (4.0 MB)          Packet

-- Press Enter to continue --
```

Figure 6-44. Example Screen—Configuration

2) Process Summary

```
     Name            ID          Version      Priority     State
----------------  --------     --------     --------     -----
PRC_ric_kern.rel  0x00050000   0x01000001   0x00000000   blocked
PRC_RIC_Mbx_SS    0x01050001   0x01000001   0x00000002   blocked
PRC_ric_base.rel  0x01050002   0x01000001   0x00000028   driver
PRC_ric_mcio.rel  0x01050003   0x01000001   0x00000028   blocked
PRC_ric_scb.rel   0x01050004   0x01000001   0x00000001   blocked
PRC_PROC1.rel     0x02050005   0x00000000   0x00000028   queued
PRC_Alfonso       0x02050006   0x00000000   0x00000028   suspended
PRC_Mason         0x02050007   0x00000000   0x00000028   blocked

-- Press Enter to continue --
```

Figure 6-45. Example Screen—Process Summary

3) Resource Summary

```
     Name              Type
----------------       ----
QUE_QUEUE_A            queue
MEM_DATA_BUFFERS       memory
SEM_ProcessSync1       semaphore
SEM_ProcessSync2       semaphore
TIM_FeedMeNow          timer

-- Press Enter to continue --
```

Figure 6-46. Example Screen—Resource Summary

4) Memory

```
Enter [Address|Name][Length][B|W] Or 0 to Return => 22040000 2A B

  22040000  30 31 32 33 34 00 41 42-43 00 00 00 00 00 00 00    01234.ABC.......
  22040010  30 30 30 00 00 00 61 62-00 00 00 00 00 00 00 00    000...ab........
  22040020  00 00 00 00 01 02 03 04-05 06                        ..........

Enter [Address|Name][Length][B|W] Or 0 to Return => 22040000 8 W

  22040000  33323130 42410034 00000043 00000000
  22040010  00303030 62610000 00000000 00000000
```

Figure 6-47. Example Screen—Memory

5) Process Details

```
Enter process name or ID => PROCESS_A

Name                  PRC_PROCESS_A
ID                    0x01050007
Priority              0x28
Entry point           0x22061C60
Stack pointer         0x220661F0
Param pointer         0x22067134
State                 queued
Version               0x0
Type                  normal
```

Figure 6-48. Example Screen—Process Details

This example shows a process terminated by a software event. The process stopped normally with an error code of 0.

```
Enter process name or ID => 6

Name             PRC_stopproc.rel
ID               0x01050006
Priority         0x28
Entry point      0x2207FDC0
Stack pointer    0x2207E1E0
Param pointer    0x22080134
State            stopped
Version          0x0
Type             normal

Termination Code Software
Requester Id     0x01050006
Source Of Req    Local
Error Code       0x0
```

Figure 6-49. Example Screen—Process Terminated by Software Event

This example shows a process terminated by a processor event. The process was stopped because it tried to perform an unsupported operation.

```
Enter process name or ID => opfault.rel

Name             PRC_opfault.rel
ID               0x01050007
Priority         0x28
Entry point      0x22082020
Stack pointer    0x220811E0
Param pointer    0x22083144
State            stopped
Version          0x0
Type             normal

Termination Code Processor
Fault Type       Operation
Subtype          Invalid Opcode
Code Address     0x22083000
```

Figure 6-50. Example Screen—Process Terminated by Processor Event

This example shows a process terminated by an adapter event. The process was stopped because of an unsupported memory access.

```
Enter process name or ID => TPROC_3

Name             PRC_TPROC_3
ID               0x0105000B
Priority         0x28
Entry point      0x2208C000
Stack pointer    0x2208F1E0
Param pointer    0x22093134
State            stopped
Version          0x0
Type             normal

Termination Code Adapter
Trap Type        Processor
Memory Address   0x20002040
Code Address     0x2208E1E0
```

Figure 6-51. Example Screen—Process Terminated by Adapter Event

6) Process Resources

The process resources screen includes resources that the process created (owns) and opened.

```
Enter process name or ID => 6

     Name             Handle            Type
----------------   --------          ----
PRS\procab.rel     0x03040586        memory
PRC\procab.rel     0x0304058B        memory
PRD\procab.rel     0x03040589        memory
MBMJ\PAM05BJ       0x02040588        memory
STM\procab.rel     0x020A0582        timer
MBS\PAM            0x02070581        semaphore
MBX\PAM            0x0203058A        mailbox
MEM_buffa          0x02040583        memory
QSM\ErrMsg         0x02070580        semaphore
QUE_ErrMsg         0x02060584        queue
SEM_               0x0207057F        semaphore
```

Figure 6-52. Sample Screen—Process Resources

7) Process Parameters

```
Enter process name or ID => Mason

argv[0] = "Mason"
argv[1] = "Get"
argv[2] = "off"
argv[3] = "the"
argv[4] = "table"
argv[5] = "you"
argv[6] = "stupid"
argv[7] = "cat!"
```

Figure 6-53. Example Screen—Process Parameters

8) Resource Details

The following are example screens for resource details.

```
Enter resource name or handle =>0101057A

Resource type   driver
Driver name     SDD_PortDriver
Process name    PRC_DriverX
Protection      disabled

Access list:

Proc No         Process Name            Handle
--------        ----------------        --------
0x0005          PRC_Driverx             0x0101057A
0x0006          PRC_inproc.rel          0x01010569
0x0007          PRC_outproc.rel         0x0101055C
```

Figure 6-54. Example Screen—Device Driver Detail

```
Enter resource name or handle => 0102055F

Resource type   event
Name            EVN_EVENT_01
Semaphores      SEM_ProcessSync1
                SEM_ProcessSync2
                SEM_

Access list:

Proc No      Process Name      Handle
--------     ----------------  --------
0x0009       PRC_PamsProcess   0x0102055F
```

Figure 6-55. Example Screen—Event Detail

```
Enter resource name or handle => 0x01030557

Resource type   mailbox
Name            MBX_IncomingDataMbx
Type            local
Receiver        PRC_LineProcessor
Semaphore       0x01070527

Access list:

Proc No      Process Name        Handle         Memory Name
--------     ----------------    --------       ----------------
0x0011       PRC_LineProcessor   0x01030557     MBM_LineBufferData
0x0012       PRC_LineFeeder      0x0103055B     MBM_LineBufferData

Messages:

  20040000  30 31 32 33 34 00 41 42-43 00 00 00 00 00 00 00    01234.ABC.......
  20042050  30 30 30 00 00 00 61 62-00 00 00 00 00 00 00 00    000...ab........
  20040170  31 31 30 00 00 00 61 62-43 00 00 00 00 00 00 00    110...abC.......
```

Figure 6-56. Example Screen—Mailbox Detail

```
Enter resource name or handle => 0104055E

Resource type   memory
Name            MEM_DATA_BUFFERS
Address         0x20042000
Size            0x2000
AIB DMA Access  R/W
Mchl Access     R/W
Sharable        yes

Access list:

Proc No      Process Name         Handle        Access
--------     ---------------      --------      ------
0x001A       PRC_ProcMemHog       0x0104055E    R/W
0x001C       PRC_ProcessMonitor   0x01040517    R/O
```

Figure 6-57. Example Screen—Memory Detail

```
Enter resource name or handle => 01060572

Resource type   queue
Name            QUE_QUEUE_A

Access list:

Proc No     Process Name        Handle       Semaphore
--------    ----------------    --------     --------
0x0011      PRC_DonsProcess     0x01060572   0x01070571
0x0012      PRC_StevesProcess   0x01060561   0x01070560

Elements:       <empty>
```

Figure 6-58. Example Screen—Queue Detail

```
Enter resource name or handle => 01070560

Resource type   semaphore
Name            SEM_ProcessSync1
Count           0

Access list:

Proc No         Process Name        Handle
--------        ----------------    --------
0x0006          PRC_PROCA.rel       0x01070571
0x0007          PRC_PROCB.rel       0x01070560
```

Figure 6-59. Example Screen—Semaphore Detail

```
Enter resource name or handle => 0108057E

Resource type   signal
Name            BufferRcvdSig

Access list:

Proc No     Process Name      Handle      Entry      Key         Option
--------    ----------------  --------    --------   --------    --------
0x0005      MonitorAll        0x0108057E  0x2209B4F0 0x00000000  always
0x0006      MonitorSome       0x02080576  0x220617A4 0x00000001  match
```

Figure 6-60. Example Screen—Signal Detail

```
Enter resource name or handle => 010A0573

Resource type   timer
Name            TIM_FeedMeNow
Handle          0x010A055E
Handler         0x220841AC
State           running
Owner name      PRC_Mason
Owner no        0x0006
```

Figure 6-61. Example Screen—Timer Detail

```
Enter resource name or handle =>000C0022

Resource type   hardware device
Name            Strange Device
Status          0x000
Valid data      yes
Owner name      PRC_aib_proc
Owner no        0x000B

Device data:
  00040000  30 30 30 30 34 00 41 42-43 00 00 00 00 00 00 00   00004.ABC.......
  00040010  39 39                                              99
```

Figure 6-62. Example Screen—Hardware Device Detail

The following screen shows the prompts and items displayed when a vector resource number or name is specified on the main menu.

```
Enter resource name or handle => 64

Resource type   vector Vector         VEC_SWVect-64

Access list:

Proc No    Process Name  Handle      Handler     Protection  Return Code
-------    ------------  ------      -------     ----------  -----------
0x0005     PRC_TransData 0x0108057E  0x2209B4F0  enabled     yes
0x0006     PRC_MonitorErr 0x02080576 0x220617A4  disabled    yes
```

Figure 6-63. Example Screen—Vector Detail

9) Exception Conditions

```
Exception code = 0x24 (Adapter Fault: Watchdog Timeout)

Exception data:
  0x00000001  0x00000000  0x01050002  0x00000001
  0x2200CBA4  0x00000000  0x00000000  0x00000000
  0x00000000  0x00000000  0x00000000  0x00000000

-- Press Enter to continue --
```

Figure 6-64. Example Screen—Exception Conditions

10) VPD Information

```
   Displayable Message    ARTIC960 Co-Processor Adapter
   Adapter Type           00
   Part Number            0000091F7710
   FRU Number             0000061G2916
   Serial Number          12345678
   Manufacturer ID        1988000000
   EC Level               000000C33261
   ROS Level and ID       1.4



-- Press Enter to continue --
```

Figure 6-65. Example Screen—VPD Information for PS/2 Systems

```
   Displayable Message    ARTIC960 Co-Processor Adapter
   Adapter Type           00
   Part Number            0000091F7710
   FRU Number             0000061G2916
   Serial Number          12345678
   Manufacturer ID        1988000000
   EC Level               000000C33261
   Device Driver Level    1.0
   Diagnostic Level       1.0
   Loadable Microcode     01.01
   ROS Level and ID       1.4


-- Press Enter to continue --
```

Figure 6-66. Example Screen—VPD Information for RISC System/6000

11) 80960 Registers

```
Enter item for display =>11

g0  = 0x20003104       r0  = 0x2206B5D0 (pfp)
g1  = 0x00000000       r1  = 0x2206B660 (sp)
g2  = 0x00000000       r2  = 0x2200681C (rip)
g3  = 0x220672DC       r3  = 0x1FFA2010
g4  = 0x20003138       r4  = 0x00000000
g5  = 0x220672D8       r5  = 0x0000001C
g6  = 0x00000000       r6  = 0x00000001
g7  = 0x00000000       r7  = 0x00000001
g8  = 0x2206B3C0       r8  = 0x220672A0
g9  = 0x00000101       r9  = 0x00000024
g10 = 0x00000404       r10 = 0x00000020
g11 = 0x00000030       r11 = 0x00000020
g12 = 0x22068080       r12 = 0x00000000
g13 = 0x00000012       r13 = 0x00000000
g14 = 0x00000000       r14 = 0x00000000
g15 = 0x2206B610 (fp)  r15 = 0x00000000

sf0 = 0x00000FFF (IPND)
sf1 = 0x00000000 (IMSK)
sf2 = 0x00000000 (DMAC)

ip  = 0x2200681C
ac  = 0xD87F88FF
pc  = 0xFFFF6EFC
tc  = 0xF001FF81
```

Figure 6-67. Example Screen—80960 Registers

7 System Unit APIs

System unit application program interfaces (APIs) are provided to allow the developer to write programs that use the services of an ARTIC960 adapter. These APIs support only C programs.

Base API

The following interface routines are available to the application in the base API.

Routine	Page
RICOpen	265
RICClose	266
RICRead	267
RICWrite	269
RICReset	271
RICGetConfig	272
RICGetVersion	273
RICGetException	274

All API routines block until they are completed, unless otherwise noted. Refer to the *ARTIC960 Programmer's Guide* for additional information on system unit APIs.

RICOpen—Open an ARTIC960 Adapter

This routine is used to obtain a handle for use in accessing the ARTIC960 adapter.

Functional Prototype

```
RIC_ULONG      RICOpen (RIC_CARDNUM   CardNum,
                        RIC_HANDLE   *Handle,
                        RIC_ULONG     Reserved);
```

Parameters

CardNum Input. The logical card number to open for access.

Handle Output. Adapter device handle returned to the calling process. This handle is passed to all other services when referring to this adapter.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_CARD_NUMBER
RC_INVALID_RESERVED_PARM
RC_SU_OPEN_FAILED
```

Remarks

- An application must obtain a handle for each card it accesses directly through the base API services.
- The logical card numbers are assigned by the driver during installation. Refer to the *ARTIC960 Programmer's Guide* for information on the Micro Channel and PCI buses.
- In AIX:
  - The configuration manager scans the physical slots from low to high, and defines the consecutive logical card numbers starting at 0 for each supported card found. If an ARTIC960 adapter is added to a slot before an already defined ARTIC960 adapter, it is assigned the next consecutive logical number.
  - Handle is only valid to use within the process that opened it.
  - There is no thread support.
- The error `RC_SU_OPEN_FAILED` is returned if the device driver is not installed. `RC_INVALID_CARD_NUMBER` is returned if the card number is out of range (0–6 for OS/2 and Windows NT and 0–13 for AIX).

RICClose—Close an ARTIC960 Adapter

This routine is used to terminate access to an individual ARTIC960 adapter.

Functional Prototype

```
RIC_ULONG    RICClose (RIC_HANDLE   Handle,
                       RIC_ULONG    Reserved);
```

Parameters

Handle Input. The handle to be closed.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_SU_INVALID_HANDLE
```

Remarks

An application calls this routine to return a handle when it is no longer needed to access the adapter.

RICRead—Read from ARTIC960 Memory

This routine is used to read data from memory on an ARTIC960 adapter into system memory.

Functional Prototype

```
RIC_ULONG    RICRead (RIC_HANDLE    Handle,
                      RIC_PTR       SrcBuffer,
                      void         *DestBuffer,
                      RIC_ULONG     BufferLen,
                      RIC_ULONG     OptionWord);
```

Parameters

Handle Input. The handle for the ARTIC960 adapter.

SrcBuffer Input. The source memory buffer address on the adapter. This is a flat, 32-bit ARTIC960 address.

DestBuffer Input. The destination buffer address in system memory. This is a 32-bit logical address.

BufferLen Input. The length, in bytes, to be read.

OptionWord

Input. Reserved (must be 0).

Returns

```
RC_SUCCESS                           RC_NO_MORE_RES
RC_ADAPTER_EXCEPTION                 RC_RESET_ACTIVE
RC_DMA_TRANSFER_FAILED(AIX only)     RC_SCB_TRANSFER_FAILED
RC_DUMP_ACTIVE                       RC_SU_INVALID_HANDLE
RC_INVALID_ADDRESS                   RC_TIMEOUT
RC_INVALID_MEM_ACCESS                RC_SYSTEM_ERROR
RC_INVALID_OPTION                    RC_UNIT_NOT_FUNCTIONING
RC_INVALID_SIZE                      RC_WRN_PIPES_NOT_CONFIGURED
RC_NO_ADAPTER_RESPONSE
```

Remarks

- All references to ARTIC960 memory are flat addresses. There is no concept of paging, shared memory, or DMA visible to the user application.
- The memory-protection hardware on the ARTIC960 adapter reports all errors to the ARTIC960 processor. The system unit driver is not directly notified of access violations. Because of this, short RICRead calls may succeed, even though they cause access violations—whereas a long RICRead call to the same region may be rejected because of improper access rights. This is because the subsystems on the card verify proper access on all transfers requested by the system unit using SCB control elements.
- The return code `RC_WRN_PIPES_NOT_CONFIGURED` is a warning indicating the memory transfer was completed but the SCB subsystem is not configured.

- The return codes RC_DUMP_ACTIVE and RC_RESET_ACTIVE indicate a dump or reset was active when this call was made, or a dump or reset was done while the call was blocked.
- The return code RC_UNIT_NOT_FUNCTIONING occurs when the driver uses SCB control elements to move the data and the adapter does not respond within an internal driver timeout period.
- A buffer length of 0 is not valid. The maximum buffer size is limited to 64 KB.
- The IBM RISC System/6000 uses big-endian memory format, whereas the 80960 on the ARTIC960 adapter uses little-endian format across the PCI or MCA bus. It is up to the calling application to perform byte and word swapping where necessary. The RICRead and RICWrite functions do not steer the data for the application.

RICWrite—Write to ARTIC960 Memory

This routine is used to write data to memory on the ARTIC960 adapter.

Functional Prototype

```
RIC_ULONG    RICWrite (RIC_HANDLE   Handle,
                       void        *SrcBuffer,
                       RIC_PTR      DestBuffer,
                       RIC_ULONG    BufferLen,
                       RIC_ULONG    OptionWord);
```

Parameters

Handle Input. The handle for the ARTIC960 adapter.

SrcBuffer Input. The source buffer address in system memory. This is a 32-bit logical address.

DestBuffer Input. The destination buffer address on the adapter. This is a flat 32-bit ARTIC960 address.

BufferLen Input. The length, in bytes, to be written.

OptionWord
Input. Reserved (must be 0).

Returns

```
RC_SUCCESS                          RC_NO_MORE_RES
RC_ADAPTER_EXCEPTION                RC_RESET_ACTIVE
RC_DUMP_ACTIVE                      RC_SCB_TRANSFER_FAILED
RC_DMA_TRANSFER_FAILED(AIX only)    RC_SU_INVALID_HANDLE
RC_INVALID_ADDRESS                  RC_SYSTEM_ERROR
RC_INVALID_MEM_ACCESS               RC_TIMEOUT
RC_INVALID_OPTION                   RC_UNIT_NOT_FUNCTIONING
RC_INVALID_SIZE                     RC_WRN_PIPES_NOT_CONFIGURED
RC_NO_ADAPTER_RESPONSE
```

Remarks

- All references to ARTIC960 memory are flat addresses. There is no concept of paging, shared memory, or DMA visible to the user application.
- The memory-protection hardware on the ARTIC960 adapter reports all errors to the ARTIC960 processor. The system unit driver is not directly notified of access violations. Because of this, short RICRead calls may succeed, even though they cause access violations—whereas a long RICRead call to the same region may be rejected because of improper access rights. The reason is because the subsystems on the card verify proper access on all transfers requested by the system unit using SCB control elements.
- The return codes `RC_DUMP_ACTIVE` and `RC_RESET_ACTIVE` indicate a dump or reset was active when this call was made, or a dump or reset was done while the call was blocked.

- The return code `RC_INVALID_MEM_ACCESS` cannot be received in OS/2. If the driver detects an access violation, OS/2 terminates the process with a trap unless the application has an exception handler registered with OS/2.
- The return code `RC_UNIT_NOT_FUNCTIONING` occurs when the driver uses SCB control elements to move the data and the adapter does no respond within an internal driver timeout period.
- The return code `RC_WRN_PIPES_NOT_CONFIGURED` is a warning indicating the memory transfer was completed but the SCB subsystem is not configured.
- A buffer length of 0 is not valid. The maximum buffer size is limited to 64 KB.
- The IBM RISC System/6000 uses big-endian memory format, whereas the 80960 on the ARTIC960 adapter uses little-endian format across the PCI or MCA bus. It is up to the calling application to perform byte and word swapping where necessary. The RICRead and RICWrite functions do not steer the data for the application.

RICReset—Reset an ARTIC960 Adapter

This routine is used to reset an adapter.

Functional Prototype

```
RIC_ULONG      RICReset (RIC_HANDLE   Handle,
                         RIC_ULONG    Reserved);
```

Parameters

Handle Input. The handle for the ARTIC960 adapter.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_RESERVED_PARM
RC_NO_ADAPTER_RESPONSE
RC_RESET_FAILED
RC_SU_INVALID_HANDLE
RC_SYSTEM_ERROR
```

Remarks

This routine resets the adapter and aborts any pending RICRead, RICWrite, SendMbx, or ReceiveMbx commands for the adapter. In addition, the SCB configuration for the adapter is lost during the reset.

RICGetConfig—Get Configuration Information

This routine is used to obtain specific hardware configuration information that is otherwise unavailable at the application level.

Functional Prototype

```
RIC_ULONG      RICGetConfig (RIC_HANDLE    Handle,
                             RIC_ULONG     ConfigLen,
                             RIC_CONFIG   *ConfigData,
                             RIC_ULONG     Reserved);
```

Parameters

Handle Input. The handle for the ARTIC960 adapter.

ConfigLen Input. The length of the buffer provided for the returned configuration information. The length must be less than 64 KB for OS/2 and Windows NT.

ConfigData

Input. The address of a buffer in system unit memory to receive the configuration information. This is a 32-bit logical address.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                      RC_INVALID_RESERVED_PARM
RC_ADAPTER_EXCEPTION            RC_NO_ADAPTER_RESPONSE
RC_BUFFER_TOO_SMALL             RC_SU_INVALID_HANDLE
RC_INVALID_MEM_ACCESS           RC_SYSTEM_ERROR (AIX only)
RC_INVALID_SIZE
```

Remarks

- The following information is returned in the RIC_CONFIG structure:
  - Card and slot numbers
  - Window sizes and locations
  - Memory sizes
  - AIB ID

  The SlotNum field is not supported when using the ARTIC960 PCI, ARTIC960Hx, or ARTIC960Rx adapters. The value returned should not be used at this time.

  For more details on the information returned by this structure, see *RIC_CONFIG Structure* on page 290.
- When either `RC_ADAPTER_EXCEPTION` or `RC_NO_ADAPTER_RESPONSE` is returned, most of the configuration data is not valid. The partial data that is returned on these errors includes only the logical card number, slot number, and system bus base I/O address.
- The return code `RC_INVALID_MEM_ACCESS` cannot be received in OS/2. If the driver detects an access violation, OS/2 terminates the process with a trap unless the application has an exception handler registered with OS/2.

RICGetVersion—Get Version Number

This routine is used to obtain the version numbers of all of the installed ARTIC960 software. The structure returned includes major and minor version numbers for the device driver, library code, kernel, and base subsystems.

Functional Prototype

```
RIC_ULONG      RICGetVersion (RIC_HANDLE    Handle,
                              RIC_ULONG     VersionLen,
                              RIC_VERDATA  *VersionData,
                              RIC_ULONG     Reserved);
```

Parameters

Handle Input. The handle for the ARTIC960 adapter.

VersionLen
Input. The length of the buffer provided for the returned version information. (Cannot be greater than 64K–1 bytes.)

VersionData
Input. The address of a buffer in system unit memory to receive the version information. This is a 32-bit logical address.

Reserved Input. Reserved parameter (must be 0).

Returns

RC_SUCCESS	RC_INVALID_RESERVED_PARM
RC_BUFFER_TOO_SMALL	RC_INVALID_SIZE
RC_INVALID_MEM_ACCESS	RC_SU_INVALID_HANDLE

Remarks

- If the kernel or subsystems are not loaded or the adapter is inaccessible (reporting an exception, being reset, and so forth), this service returns 0 in the corresponding `RIC_VERDATA` field.

  For more details on the information returned by this structure, see *RIC_VERDATA Structure* on page 292.
- The return code `RC_INVALID_MEM_ACCESS` cannot be received in OS/2. If the driver detects an access violation, OS/2 terminates the process with a trap unless the application has an exception handler registered with OS/2.

RICGetException—Get Exception Status

This routine is used to query and wait for the ARTIC960 adapter's exception conditions.

Functional Prototype

```
RIC_ULONG      RICGetException (RIC_HANDLE       Handle,
                                RIC_ULONG        ExceptLen,
                                RIC_EXCEPT      *ExceptData,
                                RIC_TIMEOUT      Timeout,
                                RIC_ULONG        Reserved);
```

Parameters

Handle Input. The handle for the ARTIC960 adapter.

ExceptLen Input. This field specifies the length of the ExceptData buffer provided. The value must be at least 8 to allow the exception code and actual exception data length to be returned. It cannot be greater than 64K–1 bytes.

ExceptData
Input. The pointer to the buffer where the exception data should be returned.

Timeout Input. The timeout parameter specifies whether the call should block waiting for an exception condition to occur. A value of 0 indicates the call should return immediately. A value of –1 indicates the call should block until an exception occurs on the adapter. Any other value specifies the number of milliseconds to wait for an exception before timing out.

Reserved Input. Reserved parameter (must be 0).

Returns

RC_SUCCESS	RC_INVALID_TIMEOUT
RC_DUMP_ACTIVE	RC_NO_ADAPTER_RESPONSE
RC_BUFFER_TOO_SMALL	RC_NO_MORE_RES (OS/2 only)
RC_HANDLE_CLOSED	RC_RESET_ACTIVE
RC_INVALID_MEM_ACCESS	RC_SU_INVALID_HANDLE
RC_INVALID_RESERVED_PARM	RC_SYSTEM_ERROR
RC_INVALID_SIZE	RC_TIMEOUT

Remarks

- `RC_SUCCESS` indicates that an exception has occurred and the exception data is in the ExceptData field.
- `RC_DUMP_ACTIVE` and `RC_RESET_ACTIVE` are returned if a dump or reset is active when the RICGetException call is made.
- `RC_BUFFER_TOO_SMALL` indicates that an exception has occurred, but that the length of the buffer provided (specified in ExceptLen) is insufficient to return all of the exception information (partial data is returned).
- `RC_INVALID_MEM_ACCESS` cannot be received in OS/2. If the driver detects an access violation, OS/2 terminates the process with a trap unless the application has an exception handler registered with OS/2.

- `RC_NO_ADAPTER_RESPONSE` is returned if the adapter does not complete POST and cannot reliably report the failing exception condition.
- `RC_TIMEOUT` is immediately returned if the caller specifies a timeout of 0 and no exception condition is present.
- In AIX, ExceptData is word swapped for the caller because all exception data fields are defined as word length.

For more details on the information returned by this structure, see *RIC_EXCEPT Structure* on page 293.

Mailbox API

The programming interface for the mailbox routines is the same as the ARTIC960 kernel mailbox API, except that there may be slight differences in the implementations—such as additional error codes and different limits due to word sizes. These differences are noted within the function descriptions. The following are the mailbox routines.

Only remote mailboxes are supported (mailboxes between a system process and a card process). For system-process to system-process communications, the inter-process communication features of the operating system can be used.

Refer to the *ARTIC960 Programmer's Guide* for additional information on mailboxes.

CreateMbx—Create a Mailbox

This creates a mailbox and gives access to the requesting process.

Functional Prototype

```
RIC_ULONG      CreateMbx (char           * RIC_SUPTR MbxName,
                          char           * RIC_SUPTR MbxRxMemName,
                          RIC_ULONG        MsgUnitSize,
                          RIC_ULONG        MsgUnitCount,
                          RIC_ULONG        OptionWord,
                          RIC_MBXHANDLE  * RIC_SUPTR MbxHandle,
                          RIC_SEMHANDLE  * RIC_SUPTR SemHandle,
                          RIC_ULONG        Reserved);
```

Parameters

MbxName Input. A mailbox name to assign to the mailbox so other processes can get access to the same mailbox by name.

MbxRxMemName
Input. Optional storage-area name associated with this mailbox for receiving messages. A value of null means that there is no name associated with the memory, and memory cannot be shared.

MsgUnitSize
Input. The smallest message size that can be allocated. All messages are allocated in units of this size.

MsgUnitCount
Input. The maximum number of message units that can be allocated from this mailbox.

OptionWord
Input. Bit field to describe the options to be used to create the mailbox. The following constants should be ORed together to build the appropriate set of options.

- Type of mailbox to create

  The caller can create either a mailbox that accepts messages from other units (using `MBX_CREATE_GLOBAL`) or one that does not accept these messages (using `MBX_CREATE_LOCAL`).

  Because the system unit supports only remote mailboxes, the `MBX_CREATE_LOCAL` option is ignored.

- Mailbox buffer-pinning option (ignored in AIX)

  The caller can have the memory associated with mailbox buffers permanently pinned (using `MBX_PIN_MEMORY`). If this option is not selected, memory is pinned only for as long as absolutely necessary. This option applies only when memory is allocated by this CreateMbx call.

MbxHandle
Output. The mailbox handle returned to the requesting process. This handle is passed to all other mailbox services when referring to this mailbox.

SemHandle

The semaphore handle associated with the mailbox. This handle is passed to all other semaphore services when referring to this mailbox-associated semaphore. This semaphore is modified whenever a message is placed in the mailbox. In OS/2, it is cleared; in AIX, the *semval* variable is set to 0. For information on *semval*, see **/usr/include/sys/sem.h**.

OS/2 Output

The semaphore is allocated by the service and the semaphore handle is returned to the application to allow it to be used in OS/2 multiple semaphore waits.

AIX Input/Output

The semaphore must be created by the application and removed after CloseMbx. The application can then use the semaphore handle for a multiple wait call. For input, the user must initialize *semid* and *semnum* of the RIC_Semhandle (see page 279). Upon return, *semval* is initialized to 1, indicating an empty mailbox.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                  RC_NO_MBX_PROCESS
RC_DUP_RES_NAME             RC_NO_MORE_MBX
RC_INVALID_COUNT            RC_NO_MORE_MEM
RC_INVALID_NAME             RC_NO_MORE_RES
RC_INVALID_OPTION           RC_NO_MORE_SEM
RC_INVALID_RESERVED_PARM    RC_SYSTEM_ERROR
RC_INVALID_SIZE
```

Remarks

- Only the process that created the mailbox can receive messages from the mailbox.
- This service call allocates the memory requested by the user. This memory is used to keep the messages in the mailbox. If the memory name provided by the process is the same as that used on a previous CreateMbx or OpenMbx call, this service call gets access to the memory pool already created. Otherwise, the service call allocates the memory requested by the process. When memory is shared, the MsgUnitSize and MsgUnitCount parameters must each be equal to those passed when the memory was allocated. Otherwise, the `RC_INVALID_SIZE` or `RC_INVALID_COUNT` error is returned, depending on which parameter is not the same as the respective input parameter.
- OS/2 does not provide counting semaphores. In its implementation, the ReceiveMbx call sets the semaphore before blocking on it. Applications wanting to use the semaphore directly to wait on the arrival of a message must call the ReceiveMbx call with a no-wait timeout value before blocking on the semaphore. The semaphore is cleared by mailbox services when a message arrives.

- In AIX, the application is responsible for creating a semaphore and providing the returned information into the structure RIC_Semhandle (defined in **rictaixa.h**).

  ```
  typedef struct RIC_Semhandle
  {
     int semid ;
     int semnum ;
  } RIC_SEMHANDLE ;
  ```

  semid The semaphore identifier returned from semget system call

  semnum The semaphore number

- After CloseMbx is called, the application is responsible for removing the semaphore from the system. The application must not modify the variable *semval (*for information on *semval*, see **/usr/include/sys/sem.h**.), which is modified by the AIX Mailbox Daemon and has one of the following values.

  0 Messages in mailbox

  1 No messages in mailbox

- In AIX, MbxHandle is valid only within the process that obtained it. There is no thread support.

OpenMbx—Open a Mailbox

This opens a mailbox previously created by another process.

Functional Prototype

```
RIC_ULONG    OpenMbx (char           *RIC_SUPTR MbxName,
                      char           *RIC_SUPTR SendMbxMemName,
                      RIC_ULONG       MsgUnitSize,
                      RIC_ULONG       MsgUnitCount,
                      RIC_ULONG       OptionWord,
                      RIC_MBXHANDLE  *RIC_SUPTR MbxHandle,
                      RIC_ULONG      *RIC_SUPTR MbxType,
                      RIC_ULONG       Reserved);
```

Parameters

MbxName Input. The mailbox name used to create the mailbox.

SendMbxMemName
Input. Optional storage-area name associated with the mailbox for sending messages by this process. A value of NULL means that the memory cannot be shared. Refer to the *ARTIC960 Programmer's Guide* for information about mailbox memory options.

MsgUnitSize
Input. The smallest allocatable message size. All messages are allocated in units of this size. If the size is 0, `RC_INVALID_SIZE` is returned.

MsgUnitCount
Input. The maximum number of messages that can be allocated from this mailbox.

OptionWord
Input. Bit field to describe the options to be used to open the mailbox. The following constants should be ORed together to build the appropriate set of options.

- Search option for finding mailbox:

  `MBX_OPEN_SEARCH_GLOBAL`
  Other cards are searched if the mailbox does not exist on card.

  Because the system unit supports only remote mailboxes, the `MBX_OPEN_SEARCH_LOCAL` option (local cards are searched) is ignored.

- Mailbox buffer-pinning option (ignored in AIX)

  The caller can have the memory associated with mailbox buffers permanently pinned down with a parameter value of `MBX_PIN_MEMORY`. If this option is not selected, memory is pinned only for as long as absolutely necessary. This option applies only when memory is allocated by this OpenMbx call.

MbxHandle

Output. The mailbox handle returned to the requesting process. This handle is passed to all other mailbox services when referring to this mailbox.

MbxType Output. Type of mailbox that was opened. The MbxType field can return the following value:

`MBX_TYPE_REMOTE`
The mailbox is not local.

Because the system unit supports only remote mailboxes, the following options are ignored:

- `MBX_TYPE_LOCAL` (the mailbox is local but does not accept remote messages)
- `MBX_TYPE_GLOBAL` (the mailbox is local and accepts card messages)

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_DUP_RES_NAME
RC_INVALID_COUNT
RC_INVALID_NAME
RC_INVALID_OPTION
RC_INVALID_RESERVED_PARM
RC_INVALID_SIZE
RC_NAME_NOT_FOUND
RC_NO_MBX_PROCESS
RC_NO_MORE_MBX
RC_NO_MORE_MEM
RC_NO_MORE_REM_MBX
RC_NO_MORE_RES
RC_NO_MORE_RES_ON_REMOTE
RC_SYSTEM_ERROR
```

Remarks

If the memory name provided by the process is the same as that used on a previous CreateMbx or OpenMbx call, this service gets access to the already created memory. Otherwise, the service allocates the memory requested by the process. When memory is shared, the MsgUnitSize and MsgUnitCount parameters must each be less than or equal to those passed when the memory was allocated. Otherwise, `RC_INVALID_SIZE` or `RC_INVALID_COUNT` error is returned, depending on which parameter is not the same as the respective input parameter.

In AIX, MBXHandle is valid only within the process that obtained it. There is no thread support.

GetMbxBuffer—Get a Free Mailbox Buffer

This allocates a free mailbox buffer to the requesting process.

Functional Prototype

```
RIC_ULONG      GetMbxBuffer (RIC_MBXHANDLE  MbxHandle,
                             RIC_ULONG      Size,
                             void          *RIC_SUPTR *MsgPtr,
                             RIC_ULONG      Reserved);
```

Parameters

MbxHandle
: Input. Handle of the mailbox from which the process wants to get a message buffer.

Size
: Input. Message size in bytes. The size is rounded up to a multiple of the message unit size set by CreateMbx or OpenMbx. The size parameter must be in the range 0 < *Size* < 65503.

MsgPtr
: Output. Pointer to allocated mailbox buffer.

Reserved
: Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_HANDLE
RC_INVALID_RESERVED_PARM
RC_INVALID_SIZE
RC_NO_MBX_BUFFER
RC_NO_MBX_PROCESS
RC_NO_MBX_RECEIVER
```

Remarks

No more than 65503 bytes can be allocated with a single call to GetMbxBuffer.

FreeMbxBuffer—Free Mailbox Buffer

This returns a previously allocated mailbox buffer.

Functional Prototype

```
RIC_ULONG      FreeMbxBuffer (RIC_MBXHANDLE  MbxHandle,
                              void          * RIC_SUPTR MsgPtr,
                              RIC_ULONG       Reserved);
```

Parameters

MbxHandle
: Input. Handle of the mailbox where the process wants to free a message buffer.

MsgPtr
: Input. Pointer to allocated mailbox buffer.

Reserved
: Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_HANDLE
RC_INVALID_MBX_BUFFER_ADDR
RC_INVALID_RESERVED_PARM
RC_NO_MBX_PROCESS
RC_MBX_BUFFER_IN_QUEUE
```

Remarks

None

SendMbx—Send a Message

This puts a message into a mailbox.

Functional Prototype

```
RIC_ULONG      SendMbx (RIC_MBXHANDLE  MbxHandle,
                        void         * RIC_SUPTR MsgPtr,
                        RIC_ULONG      Size,
                        RIC_ULONG      OptionWord,
                        RIC_ULONG      Reserved);
```

Parameters

MbxHandle
Input. Handle of the mailbox to which the process wants to send the message.

MsgPtr Input. Pointer to the message buffer.

Size Input. Size of the message buffer. The size parameter must be in the range 0< *Size*< 65503. For ARTIC960 PCI co-processors, the size parameter must be in the range 0 < *Size* <16384.

OptionWord
Input. Bit field to describe how to send the message. Use the OR operation on the following constants to build the appropriate set of options.

`MBX_SEND_COPY`
Forces a copy of the message in the mailbox memory. This option applies only when the sender and receiver are sharing memory. Because the system unit supports only remote mailboxes, the `MBX_SEND_COPY` option is ignored.

`MBX_SEND_NO_COPY`
This is the default because the system unit supports only remote mailboxes.

`MBX_SEND_FREE_BUFFER`
Returns the buffer to the free pool.

`MBX_SEND_KEEP_BUFFER`
The buffer must be freed explicitly with the FreeMbxBuffer call. This is the default.

`MBX_SEND_LIFO`
Puts the message in the front of the message queue.

`MBX_SEND_FIFO`
The message is put in the back of the message queue. This is the default.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                    RC_NO_MORE_RES
RC_INVALID_HANDLE             RC_NO_RCV_BUFFER
RC_INVALID_MSG_BUFFER         RC_PIPES_NOT_CONFIGURED
RC_INVALID_OPTION             RC_SYSTEM_ERROR
RC_INVALID_RESERVED_PARM      RC_UNABLE_TO_ACCESS_UNIT
RC_INVALID_SIZE               RC_MBX_BUFFER_IN_QUEUE
RC_NO_MBX_PROCESS             RC_MC_TIMEOUT (AIX only)
RC_NO_MBX_RECEIVER
```

Remarks

If `MBX_SEND_FREE_BUFFER` is specified and the SendMbx service fails, the buffer is not freed. It must be explicitly freed by the sender using the FreeMbxBuffer service.

ReceiveMbx—Receive a Message

This reads or receives a message from a mailbox.

Functional Prototype

```
RIC_ULONG      ReceiveMbx (RIC_MBXHANDLE  MbxHandle,
                           RIC_ULONG      OptionWord,
                           RIC_TIMEOUT    Timeout,
                           void         * RIC_SUPTR * MsgPtr,
                           RIC_ULONG    * RIC_SUPTR Size,
                           RIC_ULONG      Reserved);
```

Parameters

MbxHandle
: Input. Handle of the mailbox from which the process wants to receive a message.

OptionWord
: Input. Option word for specifying receive options. The following constant can be used.

`MBX_RECEIVE_READ_MESSAGE`
: Return the pointer to the message but do not remove it from the mailbox message queue.

`MBX_RECEIVE_GET_MESSAGE`
: Return the pointer to the message and remove it from the mailbox message queue. This is the default.

Timeout
: Input. Optional timeout (in milliseconds) for waiting on a semaphore associated with this mailbox.

0 The process should not wait if no messages are available in the mailbox.
–1 There is no timeout. The process waits indefinitely for a message to arrive.

MsgPtr
: Output. Pointer to the received message buffer.

Size
: Output. Size of the received message buffer.

Reserved
: Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS                    RC_INVALID_TIMEOUT
RC_INVALID_HANDLE             RC_MBX_EMPTY
RC_INVALID_OPTION             RC_NO_MBX_PROCESS
RC_INVALID_RECEIVER           RC_NO_MORE_RES
RC_INVALID_RESERVED_PARM      RC_SYSTEM_ERROR
```

Remarks

- If the `MBX_RECEIVE_GET_MESSAGE` option is set in the OptionWord parameter, this call dequeues the first message buffer from the mailbox queue. The semaphore associated with the mailbox on the ARTIC960 adapter is decreased by 1.
- In OS/2 system-unit mailboxes, the semaphore is set if the dequeued message is the last one in the queue.
- In AIX system-unit mailboxes, the variable *semval* of the semaphore is set to 1 if the dequeued message is the last one in the queue. For information on *semval*, see **/usr/include/sys/sem.h**.
- If the `MBX_RECEIVE_READ_MESSAGE` option is set in the OptionWord parameter, the message is not dequeued from the message queue.

CloseMbx—Close a Mailbox

This releases the mailbox and deletes it if no other process has access to it.

Functional Prototype

```
RIC_ULONG     CloseMbx (RIC_MBXHANDLE  MbxHandle,
                        RIC_ULONG      Reserved);
```

Parameters

MbxHandle Input. Handle of the mailbox to close.

Reserved Input. Reserved parameter (must be 0).

Returns

```
RC_SUCCESS
RC_INVALID_HANDLE
RC_INVALID_RESERVED_PARM
RC_NO_MBX_PROCESS
RC_NO_MORE_RES
RC_PIPES_NOT_CONFIGURED
RC_SYSTEM_ERROR
RC_UNABLE_TO_ACCESS_UNIT
```

Remarks

- If the close is issued by a process while other processes still have access to the mailbox, the service simply removes access rights for the calling process.
- If the calling process is the only process using the memory pool associated with the mailbox, this memory pool is released by the mailbox process.
- In OS/2, if the mailbox to be closed was created by the calling process, the semaphore associated with the mailbox is released by the mailbox process.
- In AIX, the semaphore associated with the mailbox must be removed by the calling process after it calls CloseMbx.

A Structure Definition

This appendix contains structure definitions for `RIC_CONFIG`, `RIC_VERDATA`, and `RIC_EXCEPT`.

RIC_CONFIG Structure

The following is the structure definition for `RIC_CONFIG` (configuration information for the ARTIC960 adapter).

```
typedef struct RIC_Config
{
  RIC_ULONG            Reserved0;
  RIC_ULONG            AIBID;                  /* AIB ID              */
  RIC_ULONG            FullWindowLoc;          /* Physical address    */
  RIC_ULONG            FullWindowSize;         /* Size in bytes       */
  RIC_ULONG            TotalMemSize;           /* Size in bytes       */
  RIC_ULONG            Reserved1[9];
  RIC_ULONG            MCBaseIOAddr;           /* Base I/O address    */
  RIC_CARDNUM          CardNum;                /* Logical card number */
  RIC_ULONG            NumOfMemoryRegions;
  RIC_ADDRESS_RANGE    MemoryRegions[MAX_MEM_REGIONS];
  RIC_ULONG            NumOfIO_Regions;
  RIC_ADDRESS_RANGE    IO_Regions[MAX_IO_REGIONS];
  unsigned char        SlotNum;                /* Physical slot number */
  unsigned char        UnitID;                 /* SCB unit ID         */
} RIC_CONFIG;
```

Reserved0

Reserved0 contains information about the adapter card type. It indicates the bus type, the presence of data cache hardware, and the interface chip type. The following masks can be used

`RIC_CARD_TYPE` Indicates the bus type. Bus type values are:

`RIC_MCA`	Micro Channel
`RIC_PCI`	PCI (Peripheral Connect Interface)

`RIC_DCACHE` Indicates the presence of data cache hardware. Data cache hardware values are:

`0`	Data cache hardware is not present.
`1`	Data cache hardware is present.

`RIC_IF_CHIP`	Indicates the type of interface chip. Interface chip values are:	
	`RIC_MIAMI`	Miami (on an ARTIC960 MCA or ARTIC960 PCI adapter)
	`RIC_MP2P`	Miami PCI to PCI (on an ARTIC960Hx PCI adapter)
	`RIC_RP`	i960Rx (on an ARTIC960Rx PCI adapter)
	`RIC_RXD`	i960Rd (on an ARTIC960RxD PCI adapter)
`RIC_NO_P2P`	Indicates that peer-to-peer activity is not supported.	

Defined Macros

The following macros can be used to determine card information.

- isMCA
- isPCI
- isMIAMI
- isRP
- isMP2P
- isRXD

For example:

```
RIC_CONFIG   ConfigData;
isMCA(&ConfigData)     ;
```

RIC_VERDATA Structure

The following is the structure definition for `RIC_VERDATA` (version numbers of the installed ARTIC960 software).

```
typedef struct RIC_Version
{
   union
   {
      RIC_ULONG              CombinedVer;
      struct RIC_SeparateVer SeparateVer;
   } Driver;

   union
   {
      RIC_ULONG              CombinedVer;
      struct RIC_SeparateVer SeparateVer;
   } Lib;

   union
   {
      RIC_ULONG              CombinedVer;
      struct RIC_SeparateVer SeparateVer;
   } Kernel;

   union
   {
      RIC_ULONG              CombinedVer;
      struct RIC_SeparateVer SeparateVer;
   } BaseSS;

   union
   {
      RIC_ULONG              CombinedVer;
      struct RIC_SeparateVer SeparateVer;
   } MChanSS;

   union
   {
      RIC_ULONG              CombinedVer;
      struct RIC_SeparateVer SeparateVer;
   } SCBSS;
} RIC_VERDATA;
```

RIC_EXCEPT Structure

The following is the structure definition for `RIC_EXCEPT` (the exception conditions for the ARTIC960 adapters).

```
struct RIC_Except
{
  RIC_ULONG         ExceptionCode;
  RIC_ULONG         ExceptionDataSize;
  union
  {
     struct RIC_AsyncEvent    EventInfo;
     struct RIC_Invalid_Intr  InvIntr;
     struct RIC_Data_Corrupt  BadData;
     struct RIC_Kern_Init     KernIni;
     struct RIC_MBXErrInfo    MBXInfo;
     struct RIC_SCBErrInfo    SCBInfo;
     struct RIC_MCErrInfo     MCInfo;
     struct RIC_RPErrInfo     RPInfo;
     struct RIC_HxErrInfo     HxInfo;
  }ExceptionData;
};
```

B Message File

Driver, Mailbox Process, and Utility Messages

The following messages are displayed by the ARTIC960 tools, drivers, and processes. See *Mailbox Process Messages and Return Codes* on page 13 for a list of return codes for the OS/2 mailbox process.

RIC0001	**Unrecognized option: “xx”**
Explanation:	The option *xx* is not a valid command line option. This message is followed by help messages RIC0027–RIC0031.
Action:	Correct the command line and reissue the command.
Source:	Application Loader, Dump, Status, Configuration, Reset, OS/2 Driver, and Mailbox Process

RIC0002	**Invalid parameter: “*xxxxxxxx*”**
Explanation:	The parameter *xxxxxxxx* is invalid. Either a required parameter is missing or an optional parameter has been improperly specified.
Action:	Correct the parameter and reissue the command.
Source:	Application Loader, Dump, Status, Configuration, OS/2 driver, and Mailbox Process

RIC0003	**File “*yyyyyyyy*” not found**
Explanation:	File *yyyyyyyy* does not exist or is not in the specified directory.
Action:	Verify that the file exists and is in the proper directory.
Source:	Application Loader, Dump, Status, Configuration, and Mailbox Process

RIC0004	**Error accessing file “*yyyyyyyy*”**
Explanation:	An error was received when attempting to access file *yyyyyyyy*.
Action:	Verify that the file still exists and is accessible. If the file exists, make sure that no other applications are accessing the file or have a lock on it. For output files, verify that the destination file is write accessible and that the disk is not full.
Source:	Applicatio Loader, Dump, Status, Configuration, and Mailbox Process

RIC0005	**Invalid card number: *nn***
Explanation:	The specified logical card number is invalid. The card number is either nonnumeric or out of range.
Action:	Correct the card number and reissue the command.
Source:	Application Loader, Dump, Status, Configuration, Reset

RIC0006	**Insufficient storage**
Explanation:	There is not enough free storage to complete the request.
Action:	On a load operation, this indicates that there is not enough free memory available on the card. Either reduce the amount of memory required by the process, free up storage on the adapter, or install more memory on the adapter. During Mailbox Process initialization, this message indicates there is not enough system unit memory to allocate the threads memory pools. Reduce the values set for any of the following in the mailbox configuration parameter file: MAX_GLOBAL_MAILBOX MAX_REMOTE_MBX MAX_REMOTE_MAILBOX_OPEN MAX_REMOTE_MAILBOX_SEND MAX_REMOTE_MAILBOX_RCV MAX_NUM_OF_UNITS
Source:	Application Loader, Mailbox Process

RIC0007 **Invalid process name: "*xxxxxxxx*"**

Explanation: The process name *xxxxxxxx* is too long.
Action: Rename the process and retry the command.
Source: Application Loader

RIC0008 **Duplicate process name: "*xxxxxxxxx*"**

Explanation: The process name *xxxxxxxx* is already active on the adapter.
Action: Either specify a different process name, or unload the active process and retry the command.
Source: Application Loader

RIC0009 **Exception condition *xxxxxxxx* detected on card *nn***

Explanation: The adapter has detected exception condition *xxxxxxxx* (hex) on card *nn*.
Action: This message indicates that an unrecoverable exception has occurred on the adapter. Reset the adapter and retry the operation. If the problem persists, call support personnel.
Source: Application Loader, Configuration, Reset, OS/2 driver.

RIC0010 **No device response from card *nn***

Explanation: Adapter *nn* is not responding to commands.
Action: Check the state of the processes running on the adapter for severe error conditions. Reset the adapter and retry the operation. If the problem persists, call support personnel.
Source: Application Loader, Dump, Configuration, Reset, Status

RIC0011 **Dump of card *nn* in progress**

Explanation: A dump of card *nn* is currently in progress. This message is displayed during an immediate dump and after a triggered dump has been triggered by an error condition on the card.
Action: Wait for message indicating that the dump has been completed.
Source: Dump

RIC0012 **Dump of card *nn* complete**

Explanation: The dump of card *nn* is complete.
Action: Use the Status Utility to analyze the raw dump file. Reset the card to continue using it.
Source: Dump

RIC0013 **Dump trigger set for card *nn***

Explanation: A dump of card *nn* has been set up to trigger on an NMI error from the card.
Action: No action is necessary. This message is followed by a message indicating that a dump has started when the dump is triggered.
Source: Dump

RIC0014 **Triggered dump of card *nn* cancelled**

Explanation: The previously set up dump trigger for card *nn* has been canceled.
Action: To retrigger the card, call the dump utility again.
Source: Dump

RIC0015 **Triggered dump of card *nn* not pending**

Explanation: There is no untriggered dump of card *nn* pending that can be canceled.
Action: None.
Source: Dump

RIC0016 **Unexpected system error *nnnn***

Explanation: An operating system error condition has been received by the adapter firmware. The unexpected error code is *nnnn* (decimal).
Action: Consult the appropriate operating system reference to determine the meaning of the error code.
Source: Application Loader, Dump, Status, Configuration, OS/2 driver, and Mailbox Process

RIC0019 **Driver not installed**

Explanation: The driver is not installed and running in the system. This occurs when a utility or mailbox process attempts to access an ARTIC960 adapter and the device driver is not installed.
Action: Verify that the proper drivers are installed in the system and retry the operation.
Source: Application Loader, Dump, Status, Configuration, Reset, and Mailbox Process

RIC0020 **Licensed Materials — Property of RadiSys**
RadiSys ARTIC960 Adapter Support Version *n.nn.n*
(C) Copyright RadiSys Corporation yyyy, zzzz All rights reserved.
US Government Users Restricted Rights -
Use, duplication or disclosure restricted
by GSA ADP Schedule Contract with
RadiSys Corporation.
xxxxxxxx initializing

Explanation: The driver or process *xxxxxxxx* is installing. yyyy, zzzz are the copyright years.
Action: None. This message is normally followed by a message that states that the driver is installed and running.
Source: OS/2 driver

RIC0021 ***xxxxxxxxx* installed and running**

Explanation: The driver or process *xxxxxxxx* has installed successfully.
Action: None.
Source: OS/2 driver, and Mailbox Process

RIC0022 ***xxxxxxxx* successfully loaded from card *nn***

Explanation: The process *xxxxxxxx* was successfully unloaded from logical card *nn*.
Action: None.
Source: Application Loader

RIC0023 **Process *xxxxxxxx* not found on card *nn***

Explanation: The process *xxxxxxxx* was not found on logical card *nn* and could not be unloaded.
Action: Correct the process name and call the command again.
Source: Application Loader, Status

RIC0024 ***xxxxxxxx* successfully started on card *nn***

Explanation: The process *xxxxxxxx* was successfully started on logical card *nn*.
Action: None.
Source: Application Loader

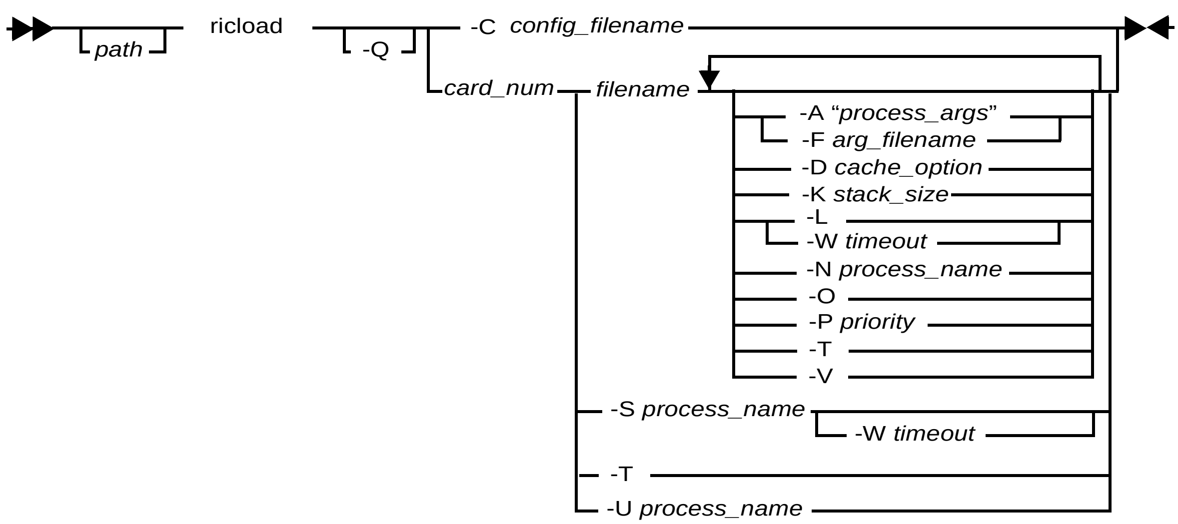
RIC0025 ***xxxxxxxx* already started on card *nn***

Explanation: The process *xxxxxxxx* was already running on logical card *nn*.
Action: Either stop and restart the process or let it run.
Source: Application Loader

RIC0026 **File format error in file "*yyyyyyyy*". Internal error *xxxxxxxx***

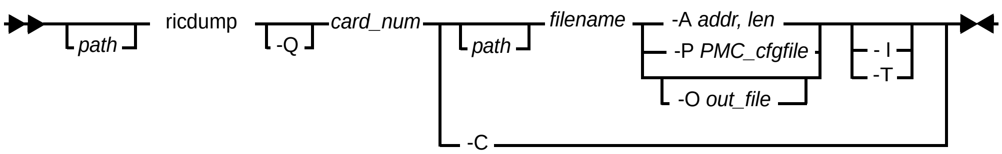
Explanation: The file *yyyyyyyy* is not in the proper format. The Application Loader returns this when a process file does not have the proper executable format. The Status utility returns this message when a dump file does not have the proper format. The error code *xx* is an internal error code that indicates the problem detected in the file.
Action: When reported by the Application Loader, recompile and relink the process in error with the proper options. When reported by the Status utility, the dump file is probably corrupted; the card must be dumped again.
Source: Application Loader, Status

RIC0027 **Correct syntax is:**



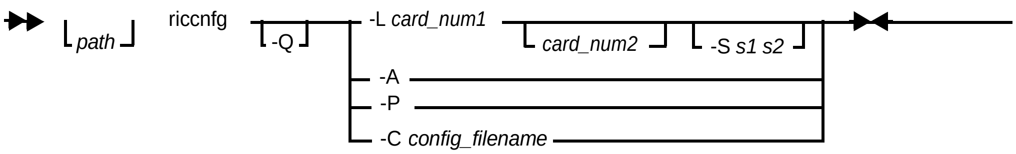
Explanation: Application Loader utility syntax help message.
Action: Select the proper parameters and call the Application Loader.
Source: Application Loader

RIC0028 **Correct syntax is:**



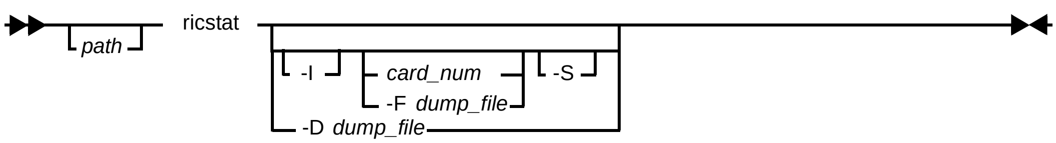
Explanation: Dump utility syntax help message.
Action: Select the proper parameters and call the Dump utility.
Source: Dump

RIC0029 **Correct syntax is:**



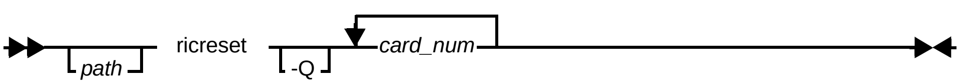
Explanation: SCB Configuration utility syntax help message.
Action: Select the proper parameters and call the SCB Configuration utility.
Source: Configuration

RIC0030 **Correct syntax is:**



Explanation: Status utility syntax help message.
Action: Select the proper parameters and call the Status utility.
Source: Status

RIC0031 **Correct syntax is:**



Explanation: Reset utility syntax help message.
Action: Select the proper parameters and call the Reset utility.
Source: Reset

RIC0032 **Reset of card *nn* in progress**

Explanation: A reset of card *nn* is in progress.
Action: None.
Source: Reset

RIC0033	**Reset of card *nn* complete**	

RIC0033 **Reset of card *nn* complete**

Explanation: Card *nn* has been reset successfully.
Action: None.
Source: Reset

RIC0034 **Reset of card *nn* failed**

Explanation: Card *nn* failed to reset.
Action: Run diagnostics to determine the cause of the failure.
Source: Reset

RIC0035 **Invalid microcode load**

Explanation: The adapter kernel is not loaded.
Action: Make sure that the kernel is properly loaded before attempting to load another process.
Source: Application Loader

RIC0036 **Peer communications between cards *xx and yy* successfully configured**

Explanation: The SCB delivery pipe was successfully configured.
Action: None
Source: Configuration

RIC0037 **Microcode error. Internal error *xxxx***

Explanation: The adapter kernel unexpectedly returned an error. *xxxx* is an internal error code.
Action: Verify that the kernel is properly loaded and there is enough memory available to satisfy Application Loader requests. *xxxx* is an internal kernel error code that generally maps to a kernel return code. If the problem persists, call support personnel.
Source: Application Loader, Configuration

RIC0038 **Error accessing card *nn*. Internal error *nnnn***

Explanation: An unexpected error was returned by the device driver while accessing card *nn. xxxx* is an internal error code that generally maps to a device driver return code.
Action: Call support personnel.
Source: Application Loader, Dump, Status, Configuration, and Reset

RIC0039 ***xxxxxxxx* not installed, no adapters found**

Explanation: The driver *xxxxxxxx* did not install because no ARTIC960 adapters were found.
Action: Verify that an adapter is installed before attempting to install the driver. If the problem persists, call support personnel.
Source: OS/2 Driver

RIC0040 **Dump on card *nn* already active**

Explanation: An attempt to call the dump utility on card *nn* failed because dump was already active for that adapter.
Action: Wait until the dump of the card has completed.
Source: Dump

RIC0041	**Peer communications not configurable with current hardware options**
Explanation:	The peer adapters could not be configured to communicate on a peer-to-peer basis due to the configuration of the adapter. Either the adapter full memory window is not present, or it is in a location that is inaccessible to the other peer adapter. This error can only be received in PS/2 systems.
Action:	Use the Reference Diskette to configure the location of the adapter memory window to allow the two adapters to communicate. In address constrained environments, it may be necessary to move an adapter from a 16-bit slot to a 32-bit slot to enable the necessary configuration.
Source:	Configuration

RIC0042	**WARNING: Process mismatch**
Explanation:	The file to be loaded was compiled for a processor type that is different from the one on the ARTIC960 adapter.
Action:	Recompile the file for the appropriate processor type.
Source:	Application Loader

RIC0043	**Peer communications pipe size out of range**
Explanation:	The peer adapters could not be configured to communicate on a peer-to-peer basis because the specified pipe size was too small.
Action:	Increase the pipe size to the minimum size.
Source:	Configuration

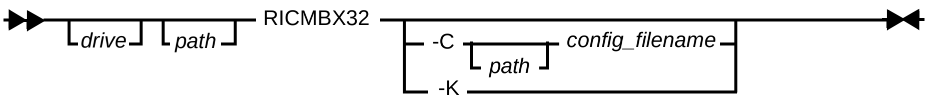
RIC0044	**Process failed to initialize**
Explanation:	The process was loaded using the –W option of the Application Loader, and it failed to issue the kernel service CompleteInit function call in the specified time period.
Action:	Correct the initialization error in the process.
Source:	Application Loader

RIC0045	**Process failed to initialize correctly. Error *xxxxxxxx***
Explanation:	The process was loaded using the –W option of the Application Loader, and it passed a non-zero error code on the kernel service CompleteInit function call. *xxxxxxxx* contains the error code.
Action:	Correct the initialization error in the process.
Source:	Application Loader

RIC0046	**Cards *xx* and *yy* are already configured**
Explanation:	The SCB pipes between units are already configured.
Action:	Accept the configuration as defined or reset the adapter and reconfigure.
Source:	Configuration

RIC0047	**Configuration failed between *xx* and *yy*.**
Explanation:	The SCB pipe between units *xx* and *yy* is already configured.
Action:	Verify the unit is not out of memory; if not, contact support personnel.
Source:	Configuration

RIC0048 **Correct syntax is:**



Explanation: Mailbox process syntax help message.

Action: Select the proper parameters and call the mailbox process.

Source: Mailbox Process

RIC0049 **Unable to install interrupt handler for card *nn***

Explanation: The driver could not allocate the interrupt level for card nn. The driver allocates interrupt levels with the share option. Therefore, another device has already allocated this interrupt level exclusively or more than four cards tried to share the interrupt level.

Action: For micro channel, change the interrupt level for card nn using the reference diskette. For PCI, this message indicates that a driver loaded prior to the ARTIC960 driver is claiming an interrupt as non-shared. Install an updated driver that claims the interrupt as shared for this other device.

Source: OS/2 Driver

RIC0050 **Resource *xxxxxxxx* already in use**

Explanation: The process is unable to create *xxxxxxxx* because it is already being used by another person.

Action: Terminate any other process using this resource.

Source: Mailbox Process

RIC0051 ***xxxxxxxx* already started on system unit**

Explanation: The process *xxxxxxxx* was already running on the host machine.

Action: Either stop and restart the process, or let it run.

Source: Mailbox Process

RIC0052 **Unable to set System Clock on card *nn*.**

Explanation: The system clock could not be set on card *nn*

Action: Load the base device driver on the card.

Source: Application Loader

RIC0053 **System Clock successfully started on card *nn*.**

Explanation: The system clock was successfully started on card *nn*.

Action: None

Source: Application Loader

RIC0054 **Entry Point** = ***0xnnnnnnnn***
Code = ***0xnnnnnnnn***
Data = ***0xnnnnnnnn***
BSS = ***0xnnnnnnnn***
Stack = ***0xnnnnnnnn***
Parameters = ***0xnnnnnnnn***

Explanation: Additional information about the task being loaded. Values are all in hexadecimal.
Action: None
Source: Application Loader

RIC0055 **Timeout trying to configure with card *nn*.**

Explanation: There was a timeout waiting for a response from card *nn*.
Action: Reset the adapter and reconfigure. Also, make sure all of the necessary subsystems are loaded on the card before attempting to configure the SCB pipes.
Source: Configuration

RIC0056 ***nnn* percent complete.**

Explanation: *nnn* Percent complete of the dump.
Action: None
Source: Dump

RIC0057 ***xxxxxxxx* successfully loaded on card *nn***
Process Name = **"*yyyyyyyy*"**
Process ID = ***0xnnnnnnnn***

Explanation: The file *xxxxxxxx* was successfully loaded on logical card *nn*. The process name is *yyyyyyyy* and the process ID is *0xnnnnnnnn* (hex).
Action: None
Source: Application Loader

RIC0059 **Peer communications between card *nn* and system unit successfully configured**

Explanation: Peer communications between card *nn* and the system unit were successfully configured.
Action: None
Source: Configuration

RIC0060 **Card *nn* and system unit area already configured**

Explanation: Communications between card *nn* and the system unit area already configured.
Action: None
Source: Configuration

RIC0061 **Configuration failed between card *nn* and system unit**

Explanation: Configuration between card *nn* and the system unit failed.
Action: Reset the adapter and reconfigure.
Source: Configuration

RIC0062	**Mailbox process successfully terminated.**
Explanation:	The Mailbox process was successfully terminated.
Action:	None
Source:	Mailbox Process

RIC0063	**Mailbox process not running.**
Explanation:	The Mailbox process was not found and could not be terminated.
Action:	None
Source:	Mailbox Process

RIC0064	**ROM error *0xnnnnnnnn* detected on card *nn*.**
Explanation:	The adapter has detected ROM error *0xnnnnnnnn* (hex) on card *nn*.
Action:	This message indicates that an unrecoverable exception has occurred on the adapter. Reset the adapter and retry the operation. If the problem persists, call support personnel.
Source:	Application Loader, Configuration, Reset, OS/2 Driver

RIC0065	**Symbol *xxxxxxxx* is undefined.**
Explanation:	The linker failed to understand the external symbol *xxxxxxxx*
Action:	Define symbol then recompile and link.
Source:	Application Loader

RIC0066	***xxxxxxxx* Interrupt nesting disabled**
Explanation:	Interrupt nesting disabled in the driver through the –N command line switch.
Action:	None
Source:	OS/2 Driver

RIC0067	**Pipe configuration failed between card *nn* and system unit.**
Explanation:	The configuration between card *nn* and the system unit failed.
Action:	Reset the adapter and reconfigure. Also ensure that all of the necessary subsystems are loaded on the card before attempting to configure the card.
Source:	Configuration, Application Loader, Reset.

RIC0068	**One or more of the required subsystems was not found for card *nn*.**
Explanation:	The card could not be configured because a required system was not found.
Action:	Reset the adapter and load the necessary subsystems on the card before attempting to configure the card.
Source:	Reset, Application Loader, Configuration.

RIC0069	***xxxxxxxx* SCB transfers disabled**
Explanation:	Device driver data transfers through SCB are disabled. All transfers are done through programmed I/O. This driver option is usually only configured for a development or debug environment.
Action:	To enable device driver SCB transfers, remove the –S option from the device driver CONFIG.SYS entry.
Source:	OS/2 Driver

RIC0070 ***xxxxxxxx* timeouts disabled**

Explanation: Device driver timeouts for SCB transfers and commands to the card are disabled. This driver option is usually only configured for a development or debug environment.

Action: To enable device driver timeouts, remove the –T option from the device driver CONFIG.SYS entry.

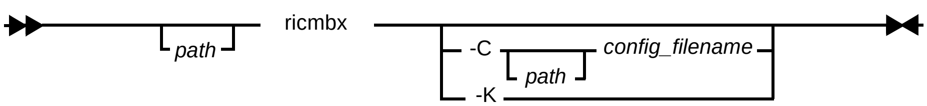
Source: OS/2 Driver

RIC0071 **Down-level ROM version on card %1.**

Explanation: The version of ROM on the adapter is down level and cannot be supported by the device driver.

Action: Update the ROM code on the adapter to a valid level.

RIC0072 **Correct syntax is:**



Explanation: Mailbox process syntax help message.

Action: Select the proper parameters and call the mailbox process.

Source: Mailbox process.

RIC0073 **Timeout during mailbox initialization.**

Explanation: Initialization of the mailbox process failed.

Action: Restart the process.

Source: Mailbox Process

RIC0075 **Only 4 –A options can be specified.**

Explanation: The ricdump utility only accepts four –A options at one time.

Action: Retry the command with four or fewer –A options.

Source: Dump

RIC0076 **User must have root authority to execute ricmbx.**

Explanation: ricmbx requires root authority for execution.

Action: Login with root authority, and reissue the command.

Source: Mailbox Process

RIC0079 **Unable to register hardware for card *nn***

Explanation: The driver was unable to register hardware information with the operating system. Conflicting settings and/or unsupported hardware options may be the cause of the problem.

Action: Verify adapter configuration and check that the operating system is at the required install level.

Source: Novell Driver

RIC0080 **Warning: Unsupported option: *xxxxxxxx***

Explanation: The parameter *xxxxxxxx* is not supported.

Action: No action is needed because the parameter *xxxxxxxx* is ignored.

Source: Configuration, Dump, Application Loader

RIC0081 **Calibrating ARTIC 960/RP Timers using card *nn***

Explanation: Informational message notifying the user that the device driver is calculating the local bus speed constant using the ARTIC 960/RP card displayed in the message.

Action: None

Source: OS/2 Driver

RIC0082 **Unsupported option *xxxxxxxx* for this hardware.**

Explanation: This option *xxxxxxxx* is not supported with the current hardware.

Action: Reissue the command without option *xxxxxxxx*.

Source: Dump

RIC0083 **Dump process not followed correctly.**

Explanation: One must first initiate a regular dump of the card before a dump of the PMC regions can be dumped.

Action: Reissue the command dumping the card first and then the PMC regions.

Source: Dump

RIC0084 **Dump of PMC on card *xxxxxxxx* in progress.**

Explanation: The PMC dump of card *xxxxxxxx* is currently in progress.

Action: Wait for a message indicating that the PMC dump has completed.

Source: Dump

RIC0085 **Dump of PMC on card *xxxxxxxx* complete.**

Explanation: The PMC dump of card *xxxxxxxx* is complete.

Action: Use a binary editor to analyze the raw dump file. Reset the card to continue using it.

Source: Dump

RIC0086 **The format of the configuration file is incorrect.**

Explanation: The configuration specified has too many entries or the syntax of the entries is incorrect.

Action: Reduce the number of entries in the configuration file or correct the syntax of the entries in the configuration file and reissue the command.

Source: Dump

RIC0087

Explanation: The format specified is incorrect.

Action: Correct the format and reissue the command.

Source: Dump

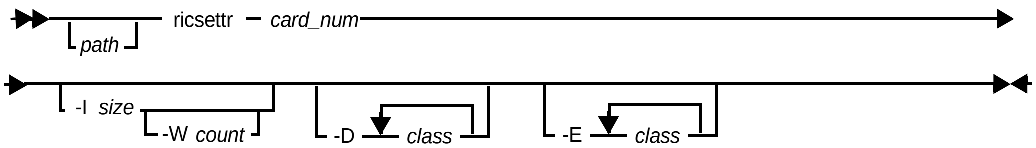
RIC0100–RIC0299

Explanation: These messages are used in the status utility.

Action: None

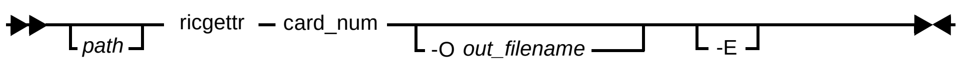
Source: Status

RIC0300 **Correct syntax is:**



Explanation: Set Trace utility syntax help message.
Action: Select the proper parameters and call the Set Trace.
Source: Set Trace

RIC0301 **Correct syntax is:**



Explanation: Get Trace utility syntax help message.
Action: Select the proper parameters and call the Get Trace.
Source: Get Trace

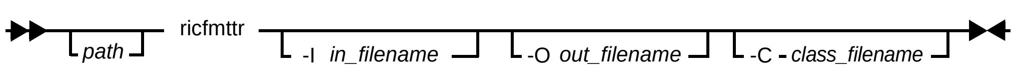
RIC0302 **Trace buffer successfully fetched from card *nn* in file ssssssss**

Explanation: The trace buffer was successfully read from card number *nn* and written to a file name *ssssssss*.
Action: None
Source: Get Trace

RIC0303 **Run ricfmttr to format and view the trace**

Explanation: After a successful Get Trace, this message is displayed to instruct the user to run the Format Utility to analyze the results of the trace.
Action: None
Source: Get Trace

RIC0304 **Correct syntax is:**



Explanation: Format Trace utility syntax help message.
Action: Select the proper parameters and call the Format Trace.
Source: Format Trace

RIC0305 **Trace uninitialized on card *nn***

Explanation: Get Trace failed to enable and/or disable a service class because the trace buffer was not previously initialized on card number *nn*.
Action: Include –I on the ricgettr command line.
Source: Get Trace

RIC0306 **The trace buffer is empty - no trace logged**

Explanation: The trace file is empty.
Action: Run the card application to be traced, run Get Trace and rerun Format Trace.
Source: Format Trace

RIC0307 – RIC0322

Explanation: These messages are used to format the trace buffer.
Action: None
Source: Format Trace

RIC0323 **Trace input file successfully formatted**

Explanation: The Trace Formatter successfully formatted the input trace file.
Action: None
Source: Format Trace

RIC0324 **Invalid Service Class *xxx*: Valid Class Range <0 - 255>**

Explanation: The service class specified *xxx* must be in the range 0 to 255.
Action: Select a valid service class and reenter command.
Source: Set Trace

RIC0325

Explanation: This message is used to format the trace buffer.
Action: None.
Source: Format Trace

RIC0326 **Trace successfully set on card *nn***

Explanation: The Set Trace command was successfully executed on card number *nn*.
Action: None

RIC0350–RIC0399

Explanation: These messages are used for the ROM Update Utility.
Action: None
Source: ROM Update

RIC0400–RIC0460

Explanation: These messages are used for the RICDiag utility.
Action: None
Source: RICDiag

C Return, Error, and Exit Codes

This appendix contains a listing of the codes used by programs and applications in the ARTIC environment. Return codes are returned by the various routines and services provided by the ARTIC960 APIs. These codes are listed in alphabetic and numeric order. The numeric listing includes a description of the exception condition.

The terminal error codes for the adapter, returned by the kernel, and the exit codes, returned by the system utilities, are listed in numeric order only.

Return Codes (Listed Alphabetically)

Return Code	VALUE
RC_ADAPTER_EXCEPTION	0x00010001
RC_ALREADY_INITIALIZED	0x00010209
RC_BAD_QUEUE_ELEMENT	0x80011601
RC_BAD_CONFIG_PARAM	0x00011201
RC_BUFFER_TOO_SMALL	0x00010019
RC_CALL_TERMINATED	0x00010105
RC_CANT_STOP_SHARING	0x00010303
RC_CLOSE_ENTRY_FAILURE	0x00010A04
RC_CMD_NOT_DELIVERED	0x00010E02
RC_DD_RC_OUT_OF_RANGE	0x00010A06
RC_DEPENDENT_EVENTS	0x00010403
RC_DEVICE_DRIVER	0x00010206
RC_DMA_TRANSFER_FAILED	0x00010021
RC_DUMP_ACTIVE	0x00010002
RC_DUMP_NOT_ACTIVE	0x00010009
RC_DUP_ASYNC_EVENT	0x00010D01
RC_DUP_RES_HANDLES	0x00010503
RC_DUP_RES_NAME	0x00010101
RC_ELEMENT_NOT_FOUND	0x00010802
RC_ENTITY_ALREADY_REGISTERED	0x00010010
RC_ENTITY_NOT_FOUND	0x00011102
RC_ENTITY_NOT_REGISTERED	0x00010011
RC_HANDLE_CLOSED	0x0001000A
RC_HOOK_ALREADY_REGISTERED	0x00010F01
RC_HOOK_NOT_REGISTERED	0x00010F02
RC_HW_ALREADY_ALLOCATED	0x00010C01
RC_HW_NOT_ALLOCATED	0x00010C02
RC_INVALID_ADDRESS	0x0001001A
RC_INVALID_ALIGNMENT	0x00010307
RC_INVALID_BASEPTR	0x00010302
RC_INVALID_CALL	0x00010104
RC_INVALID_CALLER_POSITION	0x00011004
RC_INVALID_CARD_NUMBER	0x0001000D
RC_INVALID_CMD_DEST	0x00010E01
RC_INVALID_COMMAND	0x00010E03
RC_INVALID_COUNT	0x00010014
RC_INVALID_ENTITY_NUMBER	0x00010012
RC_INVALID_EVN_MASK	0x00010501
RC_INVALID_FUNCTION_CODE	0x00010108
RC_INVALID_HANDLE	0x00010020
RC_INVALID_HOOK	0x00010F03
RC_INVALID_MBX_BUFFER_ADDR	0x00010905
RC_INVALID_MEM_ACCESS	0x0001001B
RC_INVALID_MSG_BUFFER	0x00010908
RC_INVALID_NAME	0x00010015
RC_INVALID_NUM_RES	0x00011202

Return Code	VALUE
RC_INVALID_OPTION	0x00010016
RC_INVALID_PIN	0x00010B04
RC_INVALID_PRIORITY	0x0001020A
RC_INVALID_PROCEDURE_ID	0x00011003
RC_INVALID_PROCESSID	0x00010106
RC_INVALID_RECEIVER	0x00010903
RC_INVALID_RESERVED_PARM	0x0001001C
RC_INVALID_SEM_COUNT	0x00010402
RC_INVALID_SEMHANDLE	0x00010910
RC_INVALID_SERVICECLASS	0x00011002
RC_INVALID_SIZE	0x0001001D
RC_INVALID_SUBALLOC_ADDR	0x00010304
RC_INVALID_TICKS	0x80011502
RC_INVALID_TIMEOUT	0x0001001E
RC_INVALID_TIMER	0x80011501
RC_INVALID_UNIT_NUMBER	0x0001000F
RC_INVALID_VECTOR	0x00010B01
RC_INVOKE_ENTRY_FAILURE	0x00010A05
RC_MBX_BUFFER_IN_QUEUE	0x0001090F
RC_MBX_EMPTY	0x00010906
RC_MC_BUS_FAULT	0x0001130F
RC_MC_CHAINING_EX_ERR	0x00011309
RC_MC_CARD_SEL_FDBACK_ERR	0x00011303
RC_MC_CHCK_ERR	0x00011302
RC_MC_DATA_PARITY_ERR	0x00011301
RC_MC_EXCEPTION_ERR	0x00011306
RC_MC_INVALID_COMBINATION	0x00011308
RC_MC_LOCAL_BUS_PARITY_ERR	0x00011305
RC_MC_LOSS_OF_CHANNEL_ERR	0x00011304
RC_MC_MASTER_ABORT	0x0001130E
RC_MC_MEM_FAULT	0x00011310
RC_MC_POSTSTAT_EX_ERR	0x0001130A
RC_MC_SERR	0x0001130D
RC_MC_TARGET_ABORT	0x0001130C
RC_MC_TIMEOUT	0x00011307
RC_MEM_SHARING_ERROR	0x00010301
RC_MOVE_ASYNC_ALREADY_REG	0x80011402
RC_MOVE_ASYNC_HANDLER_NOT_REG	0x80011401
RC_MSG_BUFFER_NOT_FREED	0x00010902
RC_NAME_NOT_FOUND	0x00010103
RC_NEW_SEM_COUNT	0x00010401
RC_NO_ADAPTER_RESPONSE	0x00010003
RC_NO_BASE_DEVICE_DRIVER	0x00010701
RC_NO_ELEMENTS	0x0001000B
RC_NO_FLOAT_SUPPORT	0x00010208
RC_NO_MBX_BUFFER	0x00010907
RC_NO_MBX_PROCESS	0x00010909
RC_NO_MBX_RECEIVER	0x00010901

Return Code	VALUE
RC_NO_MORE_DEV	0x00010A02
RC_NO_MORE_ENTITIES	0x00010013
RC_NO_MORE_EVNS	0x00010502
RC_NO_MORE_HOOKS	0x00010F04
RC_NO_MORE_MBX	0x00010904
RC_NO_MORE_MEM	0x00010306
RC_NO_MORE_PROC	0x00010207
RC_NO_MORE_QUEUES	0x00010803
RC_NO_MORE_REM_MBX	0x0001090D
RC_NO_MORE_RES	0x0001001F
RC_NO_MORE_RES_ON_REMOTE	0x0001090A
RC_NO_MORE_SEM	0x00010404
RC_NO_MORE_SIGS	0x00010602
RC_NO_MORE_TIMERS	0x00010704
RC_NO_RCV_BUFFER	0x0001090B
RC_NO_RES_ACCESS	0x00010102
RC_NO_SUCH_SIG_ID	0x00010601
RC_NOT_DD_OR_SS	0x00010A01
RC_NOT_REGISTERED	0x00010D02
RC_OPEN_ENTRY_FAILURE	0x00010A03
RC_OWNER_CLOSED_SEM	0x00010406
RC_PCI_BAD_REGISTER_NUMBER	0x00011403
RC_PCI_DEVICE_NOT_FOUND	0x00011404
RC_PCI_INVALID_COMMAND	0x00011402
RC_PCI_NO_BIOS	0x00011401
RC_PERF_TIMER_NOT_ENABLED	0x00010707
RC_PERMANENT_PROCESS	0x00010204
RC_PIPE_FULL	0x0001000C
RC_PIPES_NOT_CONFIGURED	0x00010017
RC_PROCESSES_WAITING_ON_SEM	0x00010408
RC_PROCESS_ALREADY_STARTED	0x00010203
RC_PROCESS_NOT_LOADED	0x00010202
RC_PROCESS_NOT_STARTED	0x00010201
RC_PROCESS_STOPPED	0x00010205
RC_QUEUE_EMPTY	0x00010801
RC_REMOTE_CFG_NOT_EST	0x0001090E
RC_RESET_ACTIVE	0x00010004
RC_RESET_FAILED	0x00010005
RC_SCB_INIT_ERROR	0x00011101
RC_SCB_TRANSFER_FAILED	0x00010006
RC_SEM_ALREADY_OWNED	0x00010407
RC_SEM_NOT_OWNED	0x00010409
RC_SU_INVALID_HANDLE	0x00000006 (OS/2) 0x00000009 (AIX)
RC_SU_OPEN_FAILED	0x0000006E (OS/2) 0x00000013 (AIX)
RC_SUCCESS	0x00000000
RC_SYSTEM_ERROR	0x00010007
RC_TIMEOUT	0x00010018

Return Code	VALUE
RC_TIMER_IS_ACTIVE	0x00010702
RC_TIMER_IS_INACTIVE	0x00010703
RC_TIMER_OVERFLOWED	0x00010706
RC_TOD_NOT_ENABLED	0x00010705
RC_TRACE_NOT_INITIALIZED	0x00011001
RC_UNABLE_TO_ACCESS_UNIT	0x0001090C
RC_UNABLE_TO_CONVERT_ADDRESS	0x0001130B
RC_UNIT_NOT_FUNCTIONING	0x0001000E
RC_UNSUPPORTED_FUNCTION	0x00010107
RC_VECTOR_NOT_ALLOCATED	0x00010B03
RC_VECTOR_NOT_AVAILABLE	0x00010B02
RC_WRN_PIPES_NOT_CONFIGURED	0x00010008

Return Codes (Listed Numerically)

See *Mailbox Process Messages and Return Codes* on page 13 for mailbox process return codes.

Return Code	Description
0x00000000	RC_SUCCESS No error occurred.
0x00000006	RC_SU_INVALID_HANDLE In OS/2, an invalid handle was passed to the API call.
0x00000009	RC_SU_INVALID_HANDLE In AIX, an invalid handle was passed to the API call.
0x00000013	RC_SU_OPEN_FAILED In AIX, this error indicates the driver is not installed.
0x0000006E	RC_SU_OPEN_FAILED In OS/2, this error indicates the driver is not installed.
0x00010001	RC_ADAPTER_EXCEPTION A terminal adapter exception condition has been detected on the adapter.
0x00010002	RC_DUMP_ACTIVE The command was aborted by a dump of the adapter or the request or command cannot be issued because a dump is active.
0x00010003	RC_NO_ADAPTER_RESPONSE This error indicates a severe adapter error. This code is returned when the adapter fails to pass the power-on self test at power on or after a reset.
0x00010004	RC_RESET_ACTIVE A reset is currently active on the destination unit.
0x00010005	RC_RESET_FAILED The card failed to reset properly. This error usually indicates defective hardware. This error may also be returned because of either user-specified timeouts or internal driver timeouts during API calls.
0x00010006	RC_SCB_TRANSFER_FAILED An error occurred when trying to transfer data using a subsystem control block.
0x00010007	RC_SYSTEM_ERROR An unexpected system error occurred. Under AIX, more information about the error condition can be found in *errno*.
0x00010008	RC_WRN_PIPES_NOT_CONFIGURED The operation completed successfully even though there is no subsystem control block (SCB) pipe configured to communicate with the adapter.
0x00010009	RC_DUMP_NOT_ACTIVE A dump command was called without first activating the dump.
0x0001000A	RC_HANDLE_CLOSED Another thread within the process closed the process' handle, which forces any threads using that handle to abort with this error. The SCB entity is also deregistered.
0x0001000B	RC_NO_ELEMENTS This error is returned on a dequeue SCB call when no elements are available to be dequeued.
0x0001000C	RC_PIPE_FULL • The element cannot be enqueued at this time because the destination pipe is full. • The SCB pipe was full when attempting to enqueue a control element.
0x0001000D	RC_INVALID_CARD_NUMBER • The requesting card is not one of the cards specified in the move system bus operation. • The logical card number is out of range or invalid. • The requested operation is not supported on this card in this environment.

Return Code	Description
0x0001000E	RC_UNIT_NOT_FUNCTIONING • The peer unit involved in the operation is not functioning. A timeout error occurred accessing the unit or waiting for a response from the unit. • A timeout occurred when trying to send or receive an SCB element to the unit.
0x0001000F	RC_INVALID_UNIT_NUMBER • The unit number is beyond the range of acceptable unit numbers. • An invalid unit number was passed.
0x00010010	RC_ENTITY_ALREADY_REGISTERED The entity is already registered.
0x00010011	RC_ENTITY_NOT_REGISTERED The entity number passed by the caller is invalid. The entity number has not been registered.
0x00010012	RC_INVALID_ENTITY_NUMBER Entity zero is reserved by the system for the system management entity.
0x00010013	RC_NO_MORE_ENTITIES The number of entities registering has exceeded the maximum (8).
0x00010014	RC_INVALID_COUNT • The count parameter is out of range. • The mailbox message count is incompatible with the previously created mailbox.
0x00010015	RC_INVALID_NAME The name used to create or open a resource exceeds the maximum size.
0x00010016	RC_INVALID_OPTION • An invalid user option was selected, possibly through an OptionWord parameter. • An invalid option was passed on the call.
0x00010017	RC_PIPES_NOT_CONFIGURED • SCB pipes are not configured for this unit (after a reset). • The SCB pipes to the destination unit are no longer configured.
0x00010018	RC_TIMEOUT • The semaphore wait timed out before the process was awakened. This may occur during an explicit call to RequestSem or implicitly through another call that waits on a semaphore for the process. • The operation timed out before it could complete successfully.
0x00010019	RC_BUFFER_TOO_SMALL • The buffer provided by the caller is too small. The buffer will be filled up to its size. • The supplied memory buffer is not large enough to receive the entire buffer of the data requested.
0x0001001A	RC_INVALID_ADDRESS • The adapter address is out of range. • An invalid adapter address was specified. The invalid address can be either a bad memory or I/O address.
0x0001001B	RC_INVALID_MEM_ACCESS • The memory access on the address passed by the user is not appropriate for the action to be taken. The user should check system bus as well as 80960 access. • The application does not have proper access to the supplied memory buffer or the driver was unable to pin the physical memory to perform the necessary DMA request. Note that in 16-bit OS/2, applications will not receive this return code. Instead, 16-bit OS/2 terminates the process with a trap. In 32-bit OS/2, threads have the ability to get control through an exception handler when the driver reports this error.
0x0001001C	RC_INVALID_RESERVED_PARM A non-zero reserved parameter was passed. Reserved parameters must be zero.

Return Code	Description
0x0001001D	RC_INVALID_SIZE • Size of request exceeds amount of memory allocated or size is 0. • Mailbox message unit size is incompatible with previously created mailbox. • Size specified for a system bus operation exceeds maximum allowed. • The size of a passed parameter was invalid (out of range).
0x0001001E	RC_INVALID_TIMEOUT The timeout value given must be between 0 and 0xFFFF or –1.
0x0001001F	RC_NO_MORE_RES • Either no more of the resource is available for allocation, or not enough internal kernel control blocks are available to handle the allocation. If the latter is true, increasing the maximum value for the resource type removes this constraint. • All available internal Mailbox Process resources have been allocated.
0x00010020	RC_INVALID_HANDLE • An invalid resource handle was passed to a resource service. The user can use only handles returned by the Create and Open services. In addition, implicit semaphore handles returned by CreateQueue and CreateMbx cannot be passed directly to ReleaseSem or RequestSem. They can be passed only to WaitEvent. To wait on a single implicit semaphore, use GetQueue or ReceiveMbx. • An invalid semaphore handle or an invalid lock was passed to the API call.
0x00010021	RC_DMA_TRANSFER_FAILED RICRead or RICWrite attempted to obtain direct memory access to the data and a failure was reported by the operating system. This is an AIX-only return code.
0x00010101	RC_DUP_RES_NAME The same name cannot be used to create two resources of the same type. Resources of different types can have identical names.
0x00010102	RC_NO_RES_ACCESS • The requester does not have access to the resource. • Global mailboxes of the same name exist on two or more units.
0x00010103	RC_NAME_NOT_FOUND • The open resource name does not match any previously created resources. If a mailbox name was specified using the global search option, this message indicates that a global mailbox matching the resource name was not found on a remote unit. This could be because the mailbox was never created, because SCB pipes for the remote unit are not configured, or because the remote unit is not functioning. • The requested name does not exist or could not be found within the specified domain. The domain is limited to the SCB pipes configured. The query may have failed due to a timeout waiting for the SCB pipes to change to a not-full state.
0x00010104	RC_INVALID_CALL The called service is not available from the caller's environment, for example, calling a blocking service in an interrupt handler.
0x00010105	RC_CALL_TERMINATED The subsystem that was called has been stopped. This error occurs when a process was executing as an extension of the caller's process and is stopped.
0x00010106	RC_INVALID_PROCESSID The process ID parameter specified was invalid.
0x00010107	RC_UNSUPPORTED_FUNCTION The function number used for the calling SVC call is invalid.

Return Code	Description
0x00010108	RC_INVALID_FUNCTION_CODE The function number passed to QueryCallAddress is out of range. This may also be returned if a service is called directly using InvokeDev, and an invalid function number is passed.
0x00010201	RC_PROCESS_NOT_STARTED The process being stopped is not started as yet.
0x00010202	RC_PROCESS_NOT_LOADED Only a previously loaded process can be started or unloaded.
0x00010203	RC_PROCESS_ALREADY_STARTED The process has already been started.
0x00010204	RC_PERMANENT_PROCESS The process has declared itself as permanent and cannot be stopped or unloaded.
0x00010205	RC_PROCESS_STOPPED The process is already stopped.
0x00010206	RC_DEVICE_DRIVER Only a device driver/subsystem or the kernel can stop a device driver/subsystem.
0x00010207	RC_NO_MORE_PROC No more process management resources are available to create a new process.
0x00010208	RC_NO_FLOAT_SUPPORT The adapter does not support floating point.
0x00010209	RC_ALREADY_INITIALIZED Process has already called issued a CompleteInit.
0x0001020A	RC_INVALID_PRIORITY The process is trying to use a reserved or out of range priority.
0x00010301	RC_MEM_SHARING_ERROR The memory cannot be opened because it was not made sharable by the creating process.
0x00010302	RC_INVALID_BASEPTR The memory base pointer is invalid.
0x00010303	RC_CANT_STOP_SHARING The memory protection on the allocated memory cannot be made non-sharable because multiple processes have access to the memory.
0x00010304	RC_INVALID_SUBALLOC_ADDR The suballocation block cannot be freed because the suballocation block pointer is invalid.
0x00010306	RC_NO_MORE_MEM There is no more memory or not enough contiguous memory to complete the allocation request.
0x00010307	RC_INVALID_ALIGNMENT The process is trying to allocate memory on a boundary that is not possible.
0x00010401	RC_NEW_SEM_COUNT When SetSemCount is called for a semaphore that has processes waiting on it, the processes are awakened with this return code.
0x00010402	RC_INVALID_SEM_COUNT An invalid semaphore count was passed to SetSemCount.
0x00010403	RC_DEPENDENT_EVENTS The semaphore could not be closed because events still exist that depend on the semaphore. Close the events before attempting to close the semaphore.

Return Code	Description
0x00010404	RC_NO_MORE_SEM No more semaphores can be allocated. All available semaphores have been allocated.
0x00010406	RC_OWNER_CLOSED_SEM The process that owned a mutex semaphore closed it, or a process was stopped while it owned the mutex semaphore. The code and data serialized by the mutual exclusion semaphore may be in an state that cannot be determined.
0x00010407	RC_SEM_ALREADY_OWNED The process requesting the mutual exclusion semaphore already owns that semaphore.
0x00010408	RC_PROCESSES_WAITING_ON_SEM Returned when calling SetSemCount. This is a warning to the process that other processes were waiting on this semaphore.
0x00010409	RC_SEM_NOT_OWNED The semaphore is not owned by the process trying to release it.
0x00010501	RC_INVALID_EVN_MASK Invalid wait mask passed to WaitEvent.
0x00010502	RC_NO_MORE_EVNS All available events have been created.
0x00010503	RC_DUP_RES_HANDLES Duplicate semaphore handles were passed to CreateEvent.
0x00010601	RC_NO_SUCH_SIG_ID There was no process to receive the signal.
0x00010602	RC_NO_MORE_SIGS All signal resources are allocated.
0x00010701	RC_NO_BASE_DEVICE_DRIVER The service failed because the base subsystem or device driver is not installed.
0x00010702	RC_TIMER_IS_ACTIVE The TimeOfDay or Performance timers cannot be started because it is active.
0x00010703	RC_TIMER_IS_INACTIVE The time-of-day or performance timer cannot be stopped because it is inactive.
0x00010704	RC_NO_MORE_TIMERS All the timers have been allocated.
0x00010705	RC_TOD_NOT_ENABLED The time of day timer was not enabled using the TIME_OF_DAY parameter in the kernel configuration file.
0x00010706	RC_TIMER_OVERFLOWED The performance timer has already expired.
0x00010707	RC_PERF_TIMER_NOT_ENABLED The performance timer was not enabled using the PERFORMANCE_TIMER parameter in the kernel configuration file.
0x00010801	RC_QUEUE_EMPTY The queue was empty and no elements were added before the timeout expired on the call to GetQueue.
0x00010802	RC_ELEMENT_NOT_FOUND SearchQueue did not find the element in the queue.
0x00010803	RC_NO_MORE_QUEUES All queues are allocated.

Return Code	Description
0x00010901	RC_NO_MBX_RECEIVER No receiver is present for the mailbox. The mailbox has been closed.
0x00010902	RC_MSG_BUFFER_NOT_FREED • The message buffer was not returned to the pool even though the buffer return option was set in SendMbx. • Sender and receiver are sharing memory, and copy option was not used. Receiver should free buffer when finished with the message.
0x00010903	RC_INVALID_RECEIVER Only the creating process can receive messages from a mailbox.
0x00010904	RC_NO_MORE_MBX All available mailboxes have been allocated.
0x00010905	RC_INVALID_MBX_BUFFER_ADDR • The message buffer pointer was invalid. • An invalid mailbox buffer pointer was passed to FreeMbxBuffer.
0x00010906	RC_MBX_EMPTY There are no messages in the mailbox.
0x00010907	RC_NO_MBX_BUFFER • There is not enough memory left in the mailbox pool to allocate the buffer. • There are no more available mailbox buffers in the pool.
0x00010908	RC_INVALID_MSG_BUFFER The message is not in the message pool associated with the open of this mailbox or the message has been freed.
0x00010909	RC_NO_MBX_PROCESS The mailbox process is not loaded.
0x0001090A	RC_NO_MORE_RES_ON_REMOTE • A RC_NO_MORE_RES error was received from the remote unit on a remote mailbox operation. • During an open mailbox, the remote unit did not have enough available internal Mailbox Process resources to satisfy the request.
0x0001090B	RC_NO_RCV_BUFFER The destination mailbox has no receive buffers to accept the message.
0x0001090C	RC_UNABLE_TO_ACCESS_UNIT This unit is unable to perform the requested operation with the peer unit. Possible reasons are adapter exception, dump active, reset active, peer unit not functioning.
0x0001090D	RC_NO_MORE_REM_MBX All of the remote mailboxes have been allocated.
0x0001090E	RC_REMOTE_CFG_NOT_EST A global search for the named mailbox cannot be made because the remote configuration has not been established. This could be because the Configuration Utility has not successfully established system unit <-> adapter SCB pipes, because the system bus I/O Subsystem has not been installed successfully on the adapter, or because the SCB Subsystem has not been installed successfully on the adapter.
0x0001090F	RC_MBX_BUFFER_IN_QUEUE The buffer is queued currently to a mailbox and has not been received by the mailbox creator.
0x00010910	RC_INVALID_SEMHANDLE Cannot access the semaphore handle.

Return Code	Description
0x00010A01	RC_NOT_DD_OR_SS This process is not a device driver or subsystem, but is attempting to use a service restricted to device drivers and subsystems.
0x00010A02	RC_NO_MORE_DEV No more device drivers/subsystems can be created.
0x00010A03	RC_OPEN_ENTRY_FAILURE The open entry routine failed for the subsystem or device driver.
0x00010A04	RC_CLOSE_ENTRY_FAILURE The close entry routine failed for the subsystem or device driver.
0x00010A05	RC_INVOKE_ENTRY_FAILURE The call entry routine failed for the subsystem or device driver.
0x00010A06	RC_DD_RC_OUT_OF_RANGE A subsystem or device driver has returned a value out of the range specified for use by subsystems and device drivers. The acceptable range is 0XFFFF0000 through 0XFFFFFFFF.
0x00010B01	RC_INVALID_VECTOR The process is trying to allocate a vector greater than 255.
0x00010B02	RC_VECTOR_NOT_AVAILABLE The requested vector number is not available.
0x00010B03	RC_VECTOR_NOT_ALLOCATED The requester is trying to return or set a vector that was never allocated.
0x00010B04	RC_INVALID_PIN The valid range of external interrupt pin numbers is from 0 to 7.
0x00010C01	RC_HW_ALREADY_ALLOCATED The requested hardware name is already allocated.
0x00010C02	RC_HW_NOT_ALLOCATED The requester is returning a hardware resource that was not previously allocated.
0x00010D01	RC_DUP_ASYNC_EVENT A process can register an async handler for an event only once. If a process wants to change the address of its async handler, then it should de-register the async handler before re-registering it.
0x00010D02	RC_NOT_REGISTERED A process is trying to deregister an asynchronous event for which it is not registered.
0x00010E01	RC_INVALID_CMD_DEST The destination process ID for the command is invalid.
0x00010E02	RC_CMD_NOT_DELIVERED The command could not be delivered to the destination process.
0x00010E03	RC_INVALID_COMMAND The command is invalid.
0x00010F01	RC_HOOK_ALREADY_REGISTERED The hook has already been registered by the calling process.
0x00010F02	RC_HOOK_NOT_REGISTERED The process is trying to deregister a hook that it has not registered.
0x00010F03	RC_INVALID_HOOK The process is trying to register an invalid hook.
0x00010F04	RC_NO_MORE_HOOKS All available hooks are already registered.

Return Code	Description
0x00011001	RC_TRACE_NOT_INITIALIZED A call to EnableTrace or DisableTrace was made without a successful call to InitTrace. LogTrace specified a service class that was not enabled.
0x00011002	RC_INVALID_SERVICECLASS The range of valid service classes is from 0 to 255.
0x00011003	RC_INVALID_PROCEDURE_ID The Procedure ID specified is not valid for the service class.
0x00011004	RC_INVALID_CALLER_POSITION The caller position is not within the valid range of values.
0x00011101	RC_SCB_INIT_ERROR The reply to an Initialize SCB Pipe command is responding with an error element.
0x00011102	RC_ENTITY_NOT_FOUND The named entity was not found on the remote unit.
0x00011201	RC_BAD_CONFIG_PARAM Invalid parameter passed to kernel through configuration file.
0x00011202	RC_INVALID_NUM_RES The configuration parameters passed were such that the required number of resources exceeded the kernel's limit.
0x00011301	RC_MC_DATA_PARITY_ERR A system bus data parity error was returned on a Micro Channel operation.
0x00011302	RC_MC_CHCK_ERR A channel check was returned on a system bus operation.
0x00011303	RC_MC_CARD_SEL_FDBACK_ERR A card selected feedback error was returned on a system bus .
0x00011304	RC_MC_LOSS_OF_CHANNEL_ERR A loss of channel error was returned on a system bus operation.
0x00011305	RC_MC_LOCAL_BUS_PARITY_ERR A local bus parity error was returned on a system bus operation.
0x00011306	RC_MC_EXCEPTION_ERR An exception error was returned on a system bus operation.
0x00011307	RC_MC_TIMEOUT A timeout occurred on a system bus operation or waiting for DMA resources.
0x00011308	RC_MC_INVALID_COMBINATION An invalid combination error was returned on a system bus operation.
0x00011309	RC_MC_CHAINING_EX_ERR A list-chaining exception error was returned on a system bus operation.
0x0001130A	RC_MC_POSTSTAT_EX_ERR A posted status exception error was returned on a system bus operation.
0x0001130B	RC_UNABLE_TO_CONVERT_ADDRESS The system bus address does not correspond to a local card address.
0x0001130C	RC_MC_TARGET_ABORT A target abort error was returned on a system bus operation on the ARTIC960 PCI card.
0x0001130D	RC_MC_SERR A SERR# error was returned on a system bus operation on the ARTIC960 PCI card.
0x0001130E	RC_MC_MASTER_ABORT A master abort error was returned on a system bus operation on the ARTIC960Rx PCI card.

Return Code	Description
0x0001130F	RC_MC_BUS_FAULT A bus fault error was returned on a system bus operation on the ARTIC960Rx PCI card.
0x0001310	RC_MC_MEM_FAULT A memory fault error was returned on a system bus operation on the ARTIC960Rx PCI card.
0x00011401	RC_PCI_NO_BIOS PCI driver not installed or card does not have a local PCI bus.
0x00011402	RC_PCI_INVALID_COMMAND An invalid IOCTL number was issued to the PCI driver. This happens only when the driver library services are not being used.
0x00011403	RC_PCI_BAD_REGISTER_NUMBER An invalid configuration register number was specified.
0x00011404	RC_PCI_DEVICE_NOT_FOUND The PCI device is not present.
0x80011401	RC_MOVE_ASYNC_HANDLER_NOT_REG The service called requires an async handler to be registered.
0x80011402	RC_MOVE_ASYNC_ALREADY_REG The subsystem name is already registered as a move async handler.
0x80011501	RC_INVALID_TIMER A bad timer number was given to the base subsystem.
0x80011502	RC_INVALID_TICKS The base subsystem attempted to start a hardware timer with zero ticks.
0x80011601	RC_BAD_QUEUE_ELEMENT An internal link list is invalid or corrupted.

Kernel Terminal Error Codes

Error Code	Description
0x0020	TERMERR_MC_IO_FAIL System bus IO subsystem failure.
0x0021	TERMERR_SCB_FAIL SCB subsystem failure.
0x0022	TERMERR_EXTMAIL_FAIL External mailbox failure.
0x0023	TERMERR_INVALID_INTR Hardware interrupt occurred. No second-level handler was installed.
0x0024	TERMERR_WATCHDOG Watchdog timeout.
0x0025	TERMERR_PARITY A parity error has occurred. It is one of the following: multiple-bit ECC error, AIB bus read parity error with 80960 master, and local bus parity for ARTIC960 32-bit Memory Controller Chip, system bus Interface Chip, and CFE Local Bus/AIB Interface Chip.
0x0026	TERMERR_MEM_PROCESSOR Memory-protection violation with 80960 master occurred at interrupt time.
0x0027	TERMERR_MEM_MICROCHANNEL Memory-protection violation with system bus master.
0x0028	TERMERR_MEM_AIB Memory-protection violation with AIB master.
0x0029	TERMERR_ASYNC_NO_MORE_RES No more async event resources could be allocated because the internal pools are exhausted. The event cannot be processed.
0x002A	TERMERR_PROCESSOR Program has attempted to perform an illegal operation on an architecture-defined data type or a typed data structure.
0x002B	TERMERR_DATA_CORRUPTION The kernel found its internal data structures corrupted.
0x002C	TERMERR_KERNEL_INIT Kernel initialization error.
0x002D	TERMERR_NMI_INTERRUPT An NMI interrupt occurred on an ARTIC960Rx adapter.
0x002E	TERMERR_PLX_INTERRUPT PLX caused an error on an ARTIC960Hx adapter.
0x1001	TERMERR_NO_MORE_MEM There is not enough memory left in the internal pools to perform the operation.
0x1002	TERMERR_MC_ERR An error occurred on a system bus operation.
0x1003	TERMERR_NO_MORE_SEM There is no semaphore available to perform the operation.
0x1004	TERMERR_NO_MORE_QUEUES There is no queue available to perform the operation.
0x1005	TERMERR_NO_MORE_TIMERS There is no timer available to perform the operation.

Error Code	Description
0x1006	TERMERR_DATA_PARITY A data parity error was returned on a system bus operation.
0x1007	TERMERR_CHCK A channel check error was returned on a system bus operation.
0x1008	TERMERR_CARD_SEL_FDBACK A data card selected feedback error was returned on a system bus operation.
0x1009	TERMERR_LOSS_OF_CHANNEL A loss of channel error was returned on a system bus operation.
0x100A	TERMERR_LOCAL_BUS_PARITY A local bus parity error was returned on a system bus operation.
0x100B	TERMERR_EXCEPTION A local exception error was returned on a system bus operation.
0x100C	TERMERR_TIMEOUT A timeout error was returned on a system bus operation.
0x100D	TERMERR_PIPE_ACCESS A system bus error was returned while trying to enqueue an SCB element. Note: This error can occur in RISC systems if the secondary arbitration level is not configured. See *ARTIC960 Support for AIX* on page 10.
0x100E	TERMERR_PIPE_TIMEOUT A system bus timeout error occurred while trying to enqueue an SCB element.
0x100F	TERMERR_INVOKING_RIC_MCIO An error occurred trying to open or call the system bus Subsystem.
0x1010	TERMERR_INVOKING_RIC_SCB An error occurred trying to open or call the system bus Subsystem.

Refer to the *ARTIC960 Programmer's Guide* for more information about terminal errors.

Exit Codes for System Unit Utilities

The following exit codes are listed by decimal value.

Exit Code	Description
0	RC_UTIL_SUCCESS The utility command executed successfully.
1	RC_UTIL_INVALID_CARD_NUMBER The specified logical card number is invalid. The card number is either non-numeric or out of range.
2	RC_UTIL_RESET_FAILED The card failed to reset due to an exception condition detected on the card.
3	RC_UTIL_ACCESS_ERROR An unexpected error was returned by the device driver while accessing the card.
4	RC_UTIL_NO_ADAPTER_RESPONSE The adapter is not responding to commands.
5	RC_UTIL_NOT_INSTALLED The driver is not installed and running in the system. This occurs when a utility or mailbox process attempts to access an adapter and the device driver is not installed.
6	RC_UTIL_ADAPTER_EXCEPTION The adapter has detected an exception condition.
7	RC_UTIL_ALREADY_STARTED The process was already running on the adapter.
8	RC_UTIL_DUP_RES_NAME A process with the same name has already been loaded on the adapter.
9	RC_UTIL_FILE_ACCESS An error was received when attempting to access a file.
10	RC_UTIL_FILE_FORMAT A file is not in the proper format. The Application Loader returns this message when a process file does not have the proper executable format. The status utility returns this message when a dump file does not have the proper format. The trace formatter returns this message when the input trace file is not in the proper format.
11	RC_UTIL_FILE_NOT_FOUND A file does not exist or is not in the specified directory. Under AIX, it may indicate a file permissions problem.
12	RC_UTIL_INVALID_CMDLINE_OPTION An option is not a valid command line option.
13	RC_UTIL_INVALID_CMDLINE_PARM A parameter is invalid. Either a required parameter is missing or a optional parameter has been improperly specified.
14	RC_UTIL_INVALID_MICROCODE The RadiSys ARTIC960 kernel is not loaded.
15	RC_UTIL_INVALID_NAME The process name is too long.
16	RC_UTIL_MICROCODE_ERROR The kernel unexpectedly returned an error.
17	RC_UTIL_NAME_NOT_FOUND The process was not found on the adapter and could not be unloaded.

Exit Code	Description
18	RC_UTIL_NOT_PENDING There is no triggered dump pending on the adapter that can be canceled.
19	RC_UTIL_NO_MORE_MEM There is not enough free storage to complete the request.
20	RC_UTIL_PIPE_ALREADY_CONF The SCB pipes between units are already configured.
21	RC_UTIL_PIPE_CONF_FAILED Configuration failed between the adapter and the system unit.
22	RC_UTIL_PIPE_SIZE_OUT_OF_RANGE The peer adapters could not be configured to communicate on a peer-to-peer basis because the specified pipe size was too small.
23	RC_UTIL_PIPE_UNCONF The peer adapters could not be configured to communicate on a peer-to-peer basis because of the configuration of the adapter. Either the adapter full memory window is not present, or it is in a location that is inaccessible to the other peer adapter. This error can be received only in PS/2 systems.
24	RC_UTIL_PROC_DID_NOT_INIT The process was loaded using the –W option of the Application Loader and it failed to issue the kernel CompleteInit() call in the specified time period.
25	RC_UTIL_PROC_INIT_ERROR The process was loaded using the –W option of the Application Loader, and it passed a non-zero error code on the kernel CompleteInit() call.
26	RC_UTIL_PROC_MISMATCH The file to be loaded was compiled for a processor type that is different from the adapter type.
27	RC_UTIL_SYSTEM_ERROR An operating system error condition has been received by the software.
28	RC_UTIL_UNIT_NOT_FUNCTIONING The peer adapters could not be configured to communicate on a peer-to-peer basis because of the configuration of the adapter. Either the adapter full memory window is not present, or it is in a location that is inaccessible to the other peer adapter.
29	RC_UTIL_WRNHELP_GIVEN Appropriate syntax diagram is displayed for the selected utility.
30	RC_UTIL_RESOURCE_BUSY The process is unable to create the resource because it is already being used by another process.
31	RC_UTIL_TIMESET_ERROR There was a timeout waiting for a response from the adapter.
32	RC_UTIL_SNGL_PIPE_ALRDY_CONF Peer communications between the adapter and the system unit were successfully configured.
33	RC_UTIL_NOT_RUNNING The mailbox process was not found or could not be terminated.
34	RC_UTIL_SNGLPIPE_CONF_FAILED Configuration failed between the adapter and the system unit.
35	RC_UTIL_SUBSYSTEM_NOT_FOUND The specified subsystem was not found.

Exit Code	Description
36	RC_UTIL_FILL_ROM_FAILED Fill ROM failed during the ROM update process on the adapter.
37	RC_UTIL_ERASE_ROM_FAILED Erase ROM failed during the ROM update process on the adapter.
38	RC_UTIL_WRITE_ROM_FAILED Write ROM failed during the ROM update process on the adapter.
39	RC_UTIL_CHECKSUM_FAILED Checksum procedure failed during the ROM update process on the adapter.
40	RC_UTIL_DATA_COMPARE_FAILED ROM Update on the adapter failed. After the new image was written to the ROM, a comparison was done with the ROM image supplied. This comparison failed.
41	RC_UTIL_INVALID_VPD_DATA Invalid data was detected in the VPD data file.
42	RC_UTIL_INVALID_VPD_FILE Invalid VPD file format. The VPD file specified does not conform to the required format.
43	RC_UTIL_INVALID_SERIAL_NUMBER The serial number specified is invalid.
44	RC_UTIL_AIB_VPD_NOT_FOUND VPD information not found in the file specified.
45	RC_UTIL_AIB_NOT_INSTALLED AIB option is not installed. An attempt was made to update a card that is not installed.
46	RC_UTIL_INVALID_MFG_ID_NUMBER The manufacturer ID specified is invalid.
47	RC_UTIL_BASE_VPD_NOT_FOUND VPD information not found in the file.
48	RC_UTIL_UNSUPPORTED_OPTION • The option listed is not supported. • The option is not supported in this environment.
49	RC_UTIL_INVALID_ROM_FILE ROM image file specified for ROM update is not valid for the specified card.
50	RC_UTIL_ROM_FILE_WARNING The specified ROM image file cannot be positively identified for the specified card.
51	RC_UTIL_PROTECT_ROM_SECTOR One of the sectors of the flash is write protected and cannot be updated by the ROM update utility.
52	RC_UTIL_NO_ROM_FOR_PMC The PMC card does not have ROM. Cannot update the PMC ROM.
53	RC_UTIL_UNSUPPORTED_OPT_HARDWARE The option listed is not supported on the current hardware.
54	RC_UTIL_DUMP_PROCESS_ERROR A regular dump on the card was not initiated before the PMC dump was requested.
55	RC_UTIL_DUMP_CONFIG_ERROR The config file specified for the PMC dump has too many entries.
56	RC_UTIL_PARM_SYNTAX_ERROR The format of the parameter is incorrect.
58	RC_UTIL_NO_MORE_ROM The image is too large for the ROM size.

Exit Code	Description
59	RC_UTIL_OEM_ROM The image is non-RadiSys.

Glossary

A

AAL: ATM Adaptation Layer — Enhances the services provided by the ATM Layer to support functions required by the next higher level.

B

BIB: Backward indicator bit

C

calling processes:
Processes that open a signal with a NULL EntryPoint. See *receiving process*.

counting semaphore:
Semaphore used for synchronizing processes, such as synchronizing a producer-consumer pair of processes.

D

DMA: Direct memory access

E

explicit semaphore:
Semaphore that is decremented before control returns to the process.

H

HAL: Hardware abstraction layer

HPFS: High performance file system

I

ICE: 80960 interactive computing environment

implicit semaphore:
Semaphores that are decremented when the process calls the appropriate resource services, such as removing a queue element or mailbox message.

M

MVDM: Multiple virtual DOS machines

MP Safe: Multiprocessing safe

mutex: Mutual exclusion semaphores used for serializing access to code or data structures.

O

OSS: On-card STREAMS subsystem

R

RDT: Resource Descriptor Table

receiving processes:
Processes that open a signal with a non-NULL EntryPoint. See *calling processes*.

ricmbx: The mailbox process for AIX that is a daemon process that works in conjunction with the device driver to handle remote mailbox processing.

RICMBX32.EXE:
The mailbox process for OS/2 that is a detached process that works with the physical device driver to handle remote mailbox processing.

ROM: Read only memory

S

SCB: Subsystem Control Block

SAL: STREAMS Access Library

semval: AIX variable. For information on semval, see **/usr/include/sys/sem.h**.

SMP: Symmetric multiprocessing

system executables:
A collective term for the kernel and related subsystems that must be loaded onto the adapter before any application processes are loaded.

Index

D

E

F

G

H

I

K

L

M

N

O

P

S

W