IBM ARTIC Support for Windows NT and Windows 98 User's Guide Second Edition

IBM ARTIC Support for Windows NT and Windows 98 User's Guide

Second Edition

Note:

Before using this information and the product it supports, read the general information under Appendix E, “Notices and Trademarks” on page E-1.

Second Edition (March 1999)

| This edition replaces and makes obsolete the previous edition for Version 1 Release 1 of this book. This edition applies to the IBM
| ARTIC Support for Windows as follows.

| • ARTIC Support for Windows NT, Version 1.4.0 or later
| • ARTIC Support for Windows 98, Version 1.0.0 or later

| *ARTIC Support for Windows* applies to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. You may address your comments to the following address:

IBM Corporation
Department GDN
1798 N.W. 40th Street
Boca Raton, Florida 33431
U.S.A.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Preface

This edition replaces and makes obsolete the previous edition for Version 1 Release 1 of this book. This edition applies to the IBM ARTIC Support for Windows as follows.

- ARTIC Support for Windows NT, Version 1.4.0 or later
- ARTIC Support for Windows 98, Version 1.0.0 or later

Terms Used Throughout This Book

The following terms are used throughout this book.

Windows
Refers to any of the Microsoft® Windows® products:

- Windows NT Workstation 4.0
- Windows NT Server 4.0
- Windows 98

Windows NT
Refers to either of the following.

- Windows NT Workstation 4.0
- Windows NT Server 4.0

ARTIC Windows
Refers to the following ARTIC support on the IBM ARTIC adapters.

- IBM ARTIC Support for Windows NT
- IBM ARTIC Support for Windows 98

co-processor adapter
Refers to any of the co-processor adapters that are supported by ARTIC Windows.

IBM ARTIC Support for Windows NT	IBM ARTIC Support for Windows 98
• IBM ARTIC X.25 PCI • IBM ARTIC X.25 ISA • IBM ARTIC X.25 MCA • IBM ARTIC Dual Port • IBM ARTIC Multiport • IBM ARTIC Multiport/2 • IBM ARTIC Multiport 8-Port 232 • IBM ARTIC186 8-Port Adapter • IBM ARTIC186 8-Port PCI Adapter • IBM ARTIC Multiport Model II • IBM ARTIC PortMaster Adapter/A • IBM ARTIC186 Model II ISA/PCI Adapter	• IBM ARTIC X.25 PCI • IBM ARTIC X.25 ISA • IBM ARTIC Dual Port • IBM ARTIC Multiport • IBM ARTIC Multiport 8-Port 232 • IBM ARTIC186 8-Port Adapter • IBM ARTIC186 8-Port PCI Adapter • IBM ARTIC Multiport Model II • IBM ARTIC186 Model II ISA/PCI Adapter

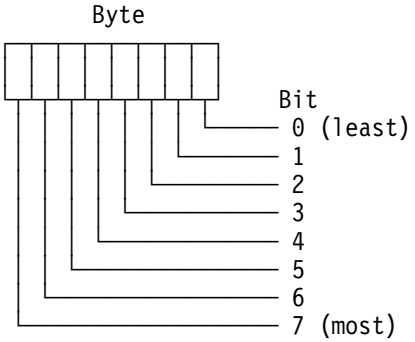
Win32 API
Refers to the *Microsoft Win32 Software Development Kit.*

DDK API
Refers to the *Microsoft Windows NT Device Driver Kit.*

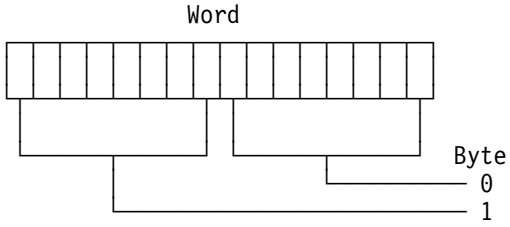
Conventions

The following conventions are used in this guide:

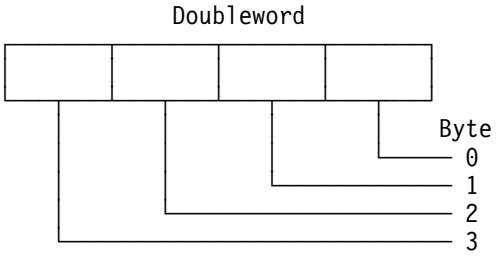
- All numbers are considered to be in decimal format unless they are immediately preceded by **0x** or immediately followed by an **h** (or **H**), in which case they are hexadecimal numbers.
- All counts in this document are assumed to start at zero.
- All bit numbering in this document conforms to the industry standard of the highest order bit having the highest bit number.
- A **byte** is 8 contiguous bits and must be considered as a single object; the bits are numbered 0 to 7. Bit number 0 is the least significant bit.



- A **word** is 16 contiguous bits and must be considered as a single object; the bits are numbered 0 to 15. Bit number 0 is the least significant bit and is in byte 0, which is the low byte.



- A **doubleword** is 32 contiguous bits and must be considered as a single object; the bits are numbered 0 to 31. Bit number 0 is the least significant bit and is in byte 0, which is the low byte.



How This Guide Is Organized

This guide is divided into the following sections:

- Chapter 1, “General Product Information” provides an overview of the ARTIC Windows support capabilities, hardware requirements, and software requirements.
- Chapter 2, “Registry Description for Windows NT” describes how the ARTIC Support for Windows NT uses the Windows NT Registry to store operational parameters and adapter detection information
- Chapter 3, “Registry Description for Windows 98” describes how the ARTIC Support for Windows 98 uses the Windows 98 Registry to store operational parameters and adapter detection information.
- Chapter 4, “Device Driver Services” describes the device driver functions and the IOCTL() system functions supported by the device driver.
- Chapter 5, “Application Loader Utility” describes how the Application Loader utility loads the Realtime Control Microcode and applications to the co-processor adapter.
- Chapter 6, “Online Dump Utility” provides instructions on how to use the Online Dump Utility to obtain a dump of the co-processor adapter’s on-board memory and registers for debugging programs.
- Chapter 7, “Dump Formatter Utility” provides instructions for formatting a dump file for viewing and printing.
- Chapter 8, “Display Utility” provides instructions for displaying co-processor adapter data structures.
- Chapter 9, “C Language Interface Routines” provides a programming interface for system unit programs to the ARTIC Windows support device driver and any adapter installed.
- Appendix A, “Output File Format for Dump Formatter Utility” contains samples of Dump Formatter Utility output files.
- Appendix B, “C Language Support and Include Files” provides information about the device driver and the C Language Interface Include files.
- Appendix C, “Return Codes” describes the return codes for the device driver, Application Loader Utility, and Online Dump Utility.
- Appendix D, “Messages” provides Application Loader Utility information and error messages, Online Dump Utility information and error messages, and Dump Formatter Utility error messages.
- Appendix E, “Notices and Trademarks” contains notices and trademarks.

Reference Publications

Windows Programming Books

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

- *Microsoft Win32 Software Development Kit*
- *Microsoft Windows NT Device Driver Kit*
- *Microsoft Windows 98 Device Driver Kit*

Other Publications

For co-processor adapter tasks, you may need one or more of the following:

- *IBM Macro Assembler/2 1.0*
- *Microsoft Macro Assembler 5.1*
- *Microsoft C 6.0 Optimizing Compiler*

Related Publications

The following are part of the ARTIC library:

- *ARTIC C Language Support User's Guide, Version 1.03.01, Volume II - Co-Processor Adapter*

  This guide provides the C interface routines and the methods of compiling and linking C tasks for the adapter.

- *IBM Realtime Interface Co-Processor Extended Services User's Guide*

  This guide explains the installation and loading of software, event management services, intertask communications services, asynchronous and synchronous communications support; provides information necessary for Realtime Interface Co-Processor Extended Services to interface with adapters; and describes the functions and capabilities of Realtime Interface Co-Processor Extended Services.

- *ARTIC Firmware Technical Reference, Volume I - System Interfaces and Functions*

  This book provides detailed information on the program interfaces to the Realtime Control Microcode for the family of Realtime Interface Co-Processor adapters. It is intended for hardware and software designers who need to understand the design and operating characteristics of the control microcode.

The following books provide both introductory and reference information, and are intended for hardware designers who need to understand the design and operating characteristics of the co-processor adapter. (See also the *ARTIC Firmware Technical Reference*.)

- *ARTIC X.25 Interface Co-Processor Technical Reference*
- *ARTIC X.25 Interface Co-Processor PCI Technical Reference*
- *ARTIC X.25 Interface Co-Processor/2 Technical Reference*
- *ARTIC Dual Port Hardware Technical Reference*
- *ARTIC Multiport Hardware Technical Reference*
- *ARTIC Multiport/2 Hardware Technical Reference*
- *ARTIC Multiport Model II Hardware Technical Reference*
- *ARTIC Portmaster Hardware Technical Reference*

You can obtain these publications from the ARTIC World Wide Web (Web) site:

http://wwprodsoln.bocaraton.ibm.com/artic/pubs.html

Chapter 1. General Product Information

This chapter provides an overview of the ARTIC Windows support and lists the minimum hardware and software requirements.

Overview

Each of the ARTIC Windows support products is a package of program services that provide an interface between applications running under Windows and tasks running on the co-processor adapter. Each product consists of the following components:

- A device driver that allows Windows applications to interface with tasks executing on a co-processor adapter
- An Application Loader utility to load task files to the co-processor adapter
- An Online Dump utility to dump adapter memory and hardware context
- A Dump Formatter utility to produce a dump report for analysis
- A Display utility to display co-processor adapter data structures.
- C language interface routines, which provide a programming interface for system unit applications to the device driver and any installed co-processor adapters
- An include file for development of Windows applications that use the device driver
- An include file for development of Windows applications that use the C language routines to interface with the device driver

The ARTIC Windows support can be used with both IBM and non-IBM products. IBM does not exercise any control over the hardware or the software of non-IBM products. The user is responsible for determining if the non-IBM products are compatible with the ARTIC Windows support. IBM does not assume any responsibility for selection of any non-IBM products, nor does it provide any information on the products, or their performance, price, or maintenance.

Hardware Requirements

The minimum hardware requirements for the ARTIC Windows support is a system unit that supports Windows and any of the target co-processor adapters shown in Table 1-1.

Table 1-1. Supported Co-Processor Adapters

IBM ARTIC Support for Windows NT	IBM ARTIC Support for Windows 98
• IBM ARTIC X.25 PCI • IBM ARTIC X.25 ISA • IBM ARTIC X.25 MCA • IBM ARTIC Dual Port • IBM ARTIC Multiport • IBM ARTIC Multiport/2 • IBM ARTIC Multiport 8-Port 232 • IBM ARTIC186 8-Port Adapter • IBM ARTIC186 8-Port PCI Adapter • IBM ARTIC Multiport Model II • IBM ARTIC PortMaster Adapter/A • IBM ARTIC186 Model II ISA/PCI Adapter	• IBM ARTIC X.25 PCI • IBM ARTIC X.25 ISA • IBM ARTIC Dual Port • IBM ARTIC Multiport • IBM ARTIC Multiport 8-Port 232 • IBM ARTIC186 8-Port Adapter • IBM ARTIC186 8-Port PCI Adapter • IBM ARTIC Multiport Model II • IBM ARTIC186 Model II ISA/PCI Adapter

Software Requirements

The minimum software requirements for the ARTIC Windows support are:

- Any of the following Microsoft Windows products.
  - Windows NT Workstation 4.0
  - Windows NT Server 4.0
  - Windows 98
- IBM Realtime Control Microcode, which is included in either of the ARTIC Windows support packages, or it can also be downloaded separately from the ARTIC Web site.

  `http://wwprodsoln.bocaraton.ibm.com/artic/file_rep.html`

Chapter 2. Registry Description for Windows NT

The ARTIC Support for Windows NT product uses the Windows NT Registry to store all operational parameters. Some sections in the Windows NT Registry are permanent across machine boots (hive) and others are dynamically updated when the ARTIC Support for Windows NT product is started, depending on co-processor adapters installed in the system. Use the following keys after you start ARTIC Support for Windows NT. (All Windows NT Registry paths listed are relative to the HKEY_LOCAL_MACHINE key.)

- HARDWARE\RESOURCEMAP\ARTIC\ARTIC

  This section provides information on hardware resources held by each co-processor adapter. This section is dynamically built when the ARTIC Windows NT is started, can only be browsed, and contains volatile information subject to change across system boots.

- SOFTWARE\IBM\ARTIC

  This section provides information on the level of software that is installed, as well as the path where the software is installed.

- SYSTEM\CurrentControlSet\Services\ARTIC\Parameters

  This section provides information on the status of each co-processor adapter installed in the system. Refer to section “Co-Processor Adapter Status for Windows NT” on page 2-3 for details.

- SYSTEM\CurrentControlSet\Services\ARTIC\pci

  This section contains one subkey per PCI (peripheral component interconnect) co-processor adapter installed in the system. Each subkey is encoded using the PCI numbering convention bus-device-function. The subkey holds co-processor adapter parameter definitions explained in section “Adapter Parameter Definitions” on page 2-2.

- SYSTEM\CurrentControlSet\Services\ARTIC\isa

  This section contains one subkey per ISA (Industry Standard Architecture) co-processor adapter in the system. The subkeys are defined by the user. Each subkey is encoded using a user-defined convention. Refer to section “Defining ISA Co-Processor Adapters for Windows NT” on page 2-4 for details. The subkey holds co-processor adapter parameter definitions explained in “Adapter Parameter Definitions” on page 2-2.

- SYSTEM\CurrentControlSet\Services\ARTIC\mca

  This section contains one subkey per MCA (Micro Channel architecture) co-processor adapter installed in the system. Each subkey is encoded using the physical slot number that can be found on the back panel of the system. The subkey holds co-processor adapter parameter definitions explained in “Adapter Parameter Definitions” on page 2-2.

Adapter Parameter Definitions

This section defines parameters configurable per co-processor adapter. You can either leave the predefined default values unchanged or change some (or all) of them according to your requirements. These parameter values are located in the Windows NT Registry under each co-processor adapter's definition identified by all subkeys present under the following:

- SYSTEM\CurrentControlSet\Services\ARTIC\pci key for PCI co-processor adapters
- SYSTEM\CurrentControlSet\Services\ARTIC\mca key for MCA co-processor adapters
- SYSTEM\CurrentControlSet\Services\ARTIC\isa key for ISA co-processor adapters

For any change to take effect, the ARTIC Support for Windows NT must be stopped and then restarted.

MAXTASK

Range: 0x00 - 0xF8
Description: This is the highest task number that can be loaded on a given co-processor adapter. Task 0 is reserved for the Realtime Control Microcode. Select this value carefully to avoid reserving unneeded space in the Realtime Control Microcode's data area.

MAXPRI

Range: 0x01 - 0xFF
Description: This is the highest priority level that you can assign to a task loaded on this co-processor adapter. Select this value carefully to avoid reserving unneeded space in the Realtime Control Microcode's data area.

MAXQUEUE

Range: 0x00 - 0xFE
Description: This is the highest queue number available for the application tasks executing on the co-processor adapter. Select this value carefully to avoid reserving unneeded space in the Realtime Control Microcode's data area.

MAXTIME

Range: 0x00 - 0xFE
Description: This is the highest timer number reserved for application tasks executing on the given co-processor adapter.
Select this value carefully to avoid reserving unneeded space in the Realtime Control Microcode's data area.

The default system parameters are:

MAXTASK Set to default value of 0x10 (decimal 16).
MAXPRI Set to default value of 0x10 (decimal 16).
MAXQUEUE Set to default value of 0x50 (decimal 80).
MAXTIME Set to default value of 0x32 (decimal 50).

Co-Processor Adapter Status for Windows NT

This section describes the steps for checking a specific co-processor adapter status within the system. It also describes how to correlate a co-processor adapter type with its assigned logical adapter number.

In the SYSTEM\CurrentControlSet\Services\ARTIC\Parameters section of the Windows NT Registry, sixteen values are defined. Each value name is formed from the same base Logical Card, plus a specific number (0 to 15) describing the logical co-processor adapter number owner (Logical Card 0, Logical Card 1, .. , Logical Card 15). The data for each of these values gives the status of each co-processor adapter:

- 0 : Co-processor adapter undefined. This logical co-processor adapter number is unused in the system.
- 1 : Co-processor adapter defined. This logical co-processor adapter number is in use by the system but not operational. This can result from a failed initialization of the co-processor adapter or simply because the co-processor adapter has been removed from the system.
- 2 : Co-processor adapter available. This is the fully operational state of a co-processor adapter. The co-processor adapter can be referenced using the logical co-processor adapter number contained in the value name. Using the logical co-processor adapter number, the co-processor adapter type can be retrieved by browsing the Windows NT Registry *AdapterName* value under the pci/mca/isa subkeys (based on the SYSTEM\CurrentControlSet\Services\ARTIC key) for the matching *CardNumber* value.

Auto-detection of ISA Co-Processor Adapters for Windows NT

The ARTIC Support for Windows NT provides the auto-detection function to detect ISA co-processor adapters. This detection is performed by reading and writing to all the base I/O addresses that are valid for the ARTIC ISA adapters. If the device driver can read and write to a base I/O address, it assumes that this is an IBM ARTIC ISA adapter. This sometimes causes an invalid entry to be written to the registry for an IBM ARTIC ISA adapter.

The auto-detection feature for the ISA co-processor adapters is enabled by default during the installation. It can be manually disabled by setting the following value in the registry to zero.

```
HKLM\SYSTEM\CurrentControlSet\Services\ARTIC\Parameters\"ISA Autodetect"
```

If the ISA auto-detection feature is disabled, the ISA co-processor adapters may need to be defined manually in the registry as described in “Defining ISA Co-Processor Adapters for Windows NT” on page 2-4.

Any invalid entries in the registry should be deleted using REGEDIT.exe or REGEDT32.exe after the auto-detection feature is disabled.

Defining ISA Co-Processor Adapters for Windows NT

This section describes the steps to add an IBM ARTIC ISA adapter definition after a new ISA co-processor adapter is added in the system.

ISA co-processor adapters must be defined within the Windows NT Registry in order for the ARTIC Support for Windows NT to recognize and drive them. Therefore, you need to manually add entries in the Windows NT Registry under the ISA section of the ARTIC service. Each ISA co-processor adapter must have a unique subkey created under the SYSTEM\CurrentControlSet\Services\ARTIC\isa key. (For example, name this key after the I/O address for the ISA co-processor adapter.) Here are the steps:

1. Create the subkey under the SYSTEM\CurrentControlSet\Services\ARTIC\isa key.
2. Create two values for the newly created subkey as follows:

   ```
   IoAddress  : REG_DWORD : <base I/O address>
   MemAddress : REG_DWORD : <Shared Storage Window address>
   ```

3. For changes to take effect, stop and then re-start the ARTIC Support for Windows NT as follows.
   a. To stop ARTIC Windows NT, enter at the command prompt the command:

      net stop artic

   b. To restart ARTIC Windows NT, enter at the command prompt the command:

      net start artic

      For example, if your ISA co-processor adapter's I/O address is 0x2a0 and you want to set the Shared Storage Window to 0xd6000, create the following subkey:

      ```
      SYSTEM\CurrentControlSet\Services\ARTIC\isa\0x2a0
      ```

      and then create the following two values under that subkey:

      ```
      IoAddress  : REG_DWORD : 0x2a0
      MemAddress : REG_DWORD : 0xd6000
      ```

Chapter 3. Registry Description for Windows 98

The ARTIC Support for Windows 98 product uses the Windows 98 Registry to store all operational parameters. Some sections in the Windows 98 Registry are permanent across machine boots (hive) and others are dynamically updated when the ARTIC Support for Windows 98 product is started, depending on the co-processor adapters installed in the system. Use the following keys after you start the ARTIC Support for Windows 98. (All Windows 98 Registry paths listed are relative to the HKEY_LOCAL_MACHINE key.)

- SOFTWARE\IBM\ARTIC

  This section provides information on the level of software installed, as well as the path where the software is installed.

- SYSTEM\CurrentControlSet\Services\Class\ARTIC\Parameters

  This section provides information on the status of each co-processor adapter installed in the system. Refer to section "Co-Processor Adapter Status for Windows 98" on page 3-2 for details.

- SYSTEM\CurrentControlSet\Services\Class\ARTIC

  This section contains one subkey per co-processor adapter installed in the system. Each subkey represents the logical card number of the co-processor adapter installed and ranges from 0000 through 0000F. The subkey holds co-processor adapter parameter definitions explained in "Adapter Parameter Definitions."

Adapter Parameter Definitions

This section defines parameters configurable per co-processor adapter. You can either leave the predefined default values unchanged or change some (or all) of them according to your requirements. These parameter values are located in the Windows 98 Registry under each co-processor adapter's definition identified by all subkeys present under the following:

- SYSTEM\CurrentControlSet\Services\Class\ARTIC*xxxx*
  (where *xxxx* is the co-processor adapter's logical card number)

For any change to take effect, Windows 98 must be shut down and restarted.

MAXTASK

Range: 0x00 - 0xF8
Description: This is the highest task number that can be loaded on a given co-processor adapter. Task 0 is reserved for the Realtime Control Microcode. Select this value carefully to avoid reserving unneeded space in the Realtime Control Microcode's data area.

MAXPRI

Range: 0x01 - 0xFF
Description: This is the highest priority level that you can assign to a task loaded on this co-processor adapter. Select this value carefully to avoid reserving unneeded space in the Realtime Control Microcode's data area.

MAXQUEUE

Range: 0x00 - 0xFE
Description: This is the highest queue number available for the application tasks executing on the co-processor adapter. Select this value carefully to avoid reserving unneeded space in the Realtime Control Microcode's data area.

MAXTIME

Range: 0x00 - 0xFE
Description: This is the highest timer number reserved for application tasks executing on the given co-processor adapter.
Select this value carefully to avoid reserving unneeded space in the Realtime Control Microcode's data area.

The default system parameters are:

MAXTASK Set to default value of 0x10 (decimal 16).
MAXPRI Set to default value of 0x10 (decimal 16).
MAXQUEUE Set to default value of 0x50 (decimal 80).
MAXTIME Set to default value of 0x32 (decimal 50).

Co-Processor Adapter Status for Windows 98

This section describes the steps for checking a specific co-processor adapter status within the system. It also describes how to correlate a co-processor adapter type with its assigned logical adapter number.

In the SYSTEM\CurrentControlSet\Services\Class\ARTIC\Parameters section of the Windows 98 Registry, sixteen values are defined. Each value name is formed from the same base Logical Card, plus a specific number (0 to 15) describing the logical co-processor adapter number owner (Logical Card 0, Logical Card 1, .. , Logical Card 15). The data for each of these values gives the status of each co-processor adapter:

- 0 : Co-processor adapter undefined. This logical co-processor adapter number is unused in the system.
- 1 : Co-processor adapter defined. This logical co-processor adapter number is in use by the system but not operational. This can result from a failed initialization of the co-processor adapter or simply because the co-processor adapter has been removed from the system.

- 2 : Co-processor adapter available. This is the fully operational state of a co-processor adapter. The co-processor adapter can be referenced using the logical co-processor adapter number contained in the value name.

SYSTEM\CurrentControlSet\Services\Class\ARTIC\xxxx defines the logical card number.

Installing and Configuring ARTIC Adapters for Windows 98

ARTIC ISA adapters should be installed and configured in the system before installing the ARTIC PCI adapters.

Installing and Configuring an ARTIC ISA Adapter

1. Install the ARTIC ISA adapter in the system and then boot the system.
2. Configure the adapter by selecting the **Add New Hardware** menu from the Control Panel.
3. When prompted, select the **artic98.inf** file from the ARTIC install directory.
4. Select the ARTIC adapter you have installed from the Models Window.

   If the resources assigned are not correct, change them using the **Device Manager**.

Installing and Configuring an ARTIC PCI Adapter

1. Install the ARTIC PCI adapter in the system and then boot the system.
2. If the **Configuration Manager** prompts for an information file, select the **artic98.inf** file in the ARTIC install directory.

The ARTIC PCI adapter should be operational when Windows 98 completes.

Chapter 4. Device Driver Services

The ARTIC Windows support device driver has implemented the following list of services accessible either from user mode applications or kernel mode drivers through the device driver calls ***DeviceIoControl*** and ***IoCallDriver***, respectively.

Note: Each indented service uses the service immediately preceding it.

- Get Buffer Addresses
- Get Parameters
- Get Primary Status
- Get Version
- Interrupt Deregister
  - Interrupt Semaphore Deregister
- Interrupt Register
  - Interrupt Semaphore Register
- Interrupt Wait
- Issue Command
- Read Memory
- Reset
- Special Events Deregister
  - Special Events Semaphore Deregister
- Special Events Register
  - Special Events Semaphore Register
- Special Events Wait
- Write Memory

The device driver supports the following list of Windows services in order for a user mode application or kernel mode driver to establish control, communicate, and release communication with the device driver. This list is not exclusive, but it represents the most commonly used services for communicating with device drivers in Windows NT or Windows 98.

CreateFile	Opens access to the device driver from a user mode application
IoGetDeviceObjectPointer	Opens access to the device driver from a kernel mode driver
DeviceIoControl	Sends an IOCTL request to the device driver from a user mode application
IoCallDriver	Sends an IOCTL request to the device driver from a kernel mode driver
CloseHandle	Closes access to the device driver from a user mode application
ObDereferenceObject	Closes access to the device driver from a kernel mode driver

Commands Synchronization

There are two types of requests. One type of request involves physical I/O operations with the co-processor adapter (adapter I/O requests) and the other type of request does not. Co-processor adapter I/O requests are not overlapped in the device driver for the same co-processor adapter. However, and to take full advantage of symmetric multiprocessors (SMP) machines, I/Os for different co-processor adapters run *concurrently* among these co-processor adapters

All co-processor adapter I/O operations are performed asynchronously. Queueing is performed on a per co-processor adapter basis.

Requests not requiring any co-processor adapter I/O are completed in the corresponding device driver's IOCTL (input/output control code) Dispatch routine. Control over the IOCTL Request Packet (IRP) is returned immediately to the caller. The **Interrupt Wait** and **Special Events Wait** functions block a user mode application and queue a kernel mode driver's request by holding the IRP as long as the condition is not met (or timeout occurs). Since a kernel mode driver cannot be blocked, the kernel mode driver controls whether to wait or not until its allocated I/O event is satisfied. All queued I/O requests can be canceled; the device driver internally releases any resource held by a terminating user mode application or kernel mode driver.

From User Mode

The caller can use the Windows I/O overlap feature, but the device driver serializes access to the hardware resource and thus will *not* perform parallel I/O operations for the same co-processor adapter. (However, concurrent I/O operations are possible for two different co-processor adapters). If the overlap feature is not used, the **DeviceIoControl** function returns to the caller when the operation is complete and the final return code is filled in. At the user level, synchronization is handled by the Windows I/O manager and the caller is blocked until the request is complete.

From Kernel Mode

The kernel mode driver performing an IOCTL to the device driver is responsible for waiting for the event it has created and set using the **IoBuildDeviceIoControlRequest** function. The **IoCallDriver** function returns *STATUS_PENDING* if the request necessitates co-processor adapter I/O operation, thus returning *before* the IOCTL is actually completed. The event is later released by the Windows I/O manager on completion of the IOCTL by all intermediate drivers involved. Another alternative to waiting for completion is to set an I/O completion routine which is called when the IRP request processing is completed by the device driver. (See **IoSetCompletionRoutine()** in the DDK API.)

Note: Different implementations are possible but are beyond the scope of this document.

Function and I/O Control Codes

When a function has a *synchronous* return, it always returns control to the caller from the device driver's dispatch routine. For a *synchronous* or *asynchronous* function, the processing involved determines if the IRP remains pending or is completed when the caller regains control.

Command	I/O Type
ICAGETBUFADDRS	sync
ICAGETPARMS	sync
ICAGETPRIMSTAT	async
ICAGETVER	sync
ICAINTDEREG	sync
ICAINTREG	sync
ICAINTWAIT	sync/async
ICAISSUECMD	async
ICAREADMEM	async
ICARESET	async
ICASEDEREG	sync
ICASEREG	sync
ICASEWAIT	sync/async
ICAWRITEMEM	async

Open/Attach Device Driver

Purpose

The device driver is defined in the I/O file system with the device special file ***\DosDevices\icaricio***. This single file is used to open the device driver, regardless of which co-processor adapter is being accessed. Windows defines the **CreateFile** function to open the device driver from a user mode application and the **IoGetDeviceObjectPointer** function to attach to the device driver from the kernel mode driver. A single call to the function allows the caller to further access all of the co-processor adapters recognized by the device driver.

From user mode

```
HANDLE    filehandle;               // device driver file handle

filehandle = CreateFile  (ICA186_DRIVERNAME,
                          (GENERIC_READ | GENERIC_WRITE),
                          (FILE_SHARE_READ | FILE_SHARE_WRITE),
                          0,
                          NULL,
                          OPEN_EXISTING,
                          FILE_ATTRIBUTE_NORMAL,
                          NULL);
```

The handle is then used to perform I/O requests to the device driver using the **DeviceIoControl** function.

From kernel mode

```
NTSTATUS        ReturnCode;
PFILE_OBJECT    fileo;               // File object
PDEVICE_OBJECT  deviceo;             // Device object

ReturnCode = IoGetDeviceObjectPointer ( "\Device\icaricio",
                                        FILE_READ_DATA,
                                        &fileo,
                                        &deviceo);
```

The device object pointer is then used to perform I/O requests to the device driver using the **IoCallDriver** function. The kernel mode driver has to provide the file object in the I/O stack of each and every IRP it builds for the device driver. This file object represents the client environment accessing the device driver and is used to identify resources in use by this client.

The use of the **FsContext2** parameter in the FileObject structure is reserved by the device driver and should not be used nor altered by any high- or intermediate-level drivers.

The device driver can be opened multiple times by the same process, thread, or high-level driver. However, event registration routines (defined in “Special Events Register” on page 4-27) are honored based on each “attachment” (handle for user mode opens, file object for kernel mode opens) between the process, thread, or high-level driver and the device driver. Since the device driver can be accessed simultaneously from user mode and kernel mode, the *file object* allocated by the I/O manager during the “attachment” becomes the *caller identification*.

If a child process is launched from a main process, and also communicates with the device driver by inheriting the handle owned by this main process, it is seen by the device driver as the same process and, therefore, cannot register events already registered by the main process. To do so, it must perform its own **CreateFile** function to get a new file object to access the device driver.

Sharing the handle

It is the application's responsibility to ensure that a child process does not close a file handle while the parent process still accesses the device driver using the same handle. To avoid this problem, the child process should create its own access to the device driver without inheriting the parent's device driver access handle.

Note: In both user modes and kernel modes, the file object pointer is passed to the device driver in the IRP.

Close/Detach Device Driver

Purpose The device driver access is closed from a user mode application with the Windows **CloseHandle** function and detached from a kernel mode driver with the Windows **ObDereferenceObject** function. A single call to the function further prevents the caller from accessing all of the co-processors recognized by the device driver.

From user mode

```
HANDLE  filehandle;  // device driver file handle from CreateFile()

CloseHandle (filehandle);
```

From kernel mode

```
PFILE_OBJECT fileo;  // File object from IoGetDeviceObjectPointer()

ObDereferenceObject (fileo);
```

Perform IOCTLs to Device Driver

Purpose Any service requested from the device driver is carried through an IOCTL Request Packet (IRP):

- From a user mode application with the Windows **DeviceIoControl** function. The major IOCTL control code generated is automatically *IRP_MJ_DEVICE_CONTROL*.
- From a kernel mode driver with the Windows **IoCallDriver** function. The major IOCTL control code *must* be set to *IRP_MJ_INTERNAL_DEVICE_CONTROL* by the kernel mode driver while building the IRP. (See **IoBuildDeviceIoControlRequest()** in either the *Microsoft Windows NT Device Driver Kit* book or the *Microsoft Windows 98 Device Driver Kit* book.)

The device driver differentiates a request coming from a user mode application or a kernel mode driver by the major IOCTL control code it receives.

From user mode

The following example shows the use of the **Reset** function from a user mode application to reset logical co-processor adapter 1.

```
HANDLE  filehandle;  // Device driver file handle from CreateFile()
ULONG   nbytesout;
BOOLEAN status;
struct ICARESET_PARMS     reset_s;

reset_s.coprocnum = 1;

status = DeviceIoControl
          (filehandle,                    // File handle
           ICARESET,                      // Command
           &reset_s,                      // InBuffer
           sizeof(struct ICARESET_PARMS)  // InBufferSize
           &reset_s,                      // OutBuffer
           sizeof(struct ICARESET_PARMS)  // OutBufferSize
           &nbytesout,                    // Mandatory
           NULL);                         // NULL if non-overlapping
```

If the OVERLAPPED feature is not used, the calling thread waits until I/O completion.

The IOCTL completion code is set inside the IOCTL Control block's **retcode** parameter for each IOCTL, if the Boolean value from the **DeviceIoControl** function is set to TRUE. The completion code is one of the E_ICA_xxxxxxx codes defined in Appendix C, "Return Codes" on page C-1. If the Boolean value is set to FALSE, the **GetLastError** function indicates the IOCTL return code from the list of STATUS_xxxxx system status codes and the IOCTL completion code (**retcode** parameter in the IOCTL control block) contains an undetermined value.

From kernel mode

```
NTSTATUS                ReturnCode;
PDEVICE_OBJECT          deviceo;   // from IoGetDeviceObjectPointer()
KEVENT                  event;
IO_STATUS_BLOCK         iostatus;
struct ICARESET_PARMS   reset_s;
PIRP                    Piorequest;

reset_s.coprocnum = 1;

   1) allocates IRP and save pointer in Piorequest
   2) initialize I/O stack's file object parameter
   3) send the IRP to the next lower driver

Piorequest = IoBuildDeviceIoControlRequest
              (ICARESET,                          // Command
               deviceo,                           // Device object
               &reset_s,                          // InBuffer
               sizeof(struct ICARESET_PARMS),     // InBufferSize
               &reset_s,                          // OutBuffer
               sizeof(struct ICARESET_PARMS),     // OutBufferSize
               TRUE,                              // Always TRUE
               &event,
               &iostatus);

ReturnCode = IoCallDriver (deviceo, Piorequest);
```

Note

This is an example of one method to initialize the IRP to be sent to the device driver. If another method is to be used, ensure that the same functions are performed.

The caller is responsible for checking the return code for an indication of asynchronous operation and can wait until IOCTL completion.

The **IoCallDriver** function returned value must be checked to determine if the operation is still pending (STATUS_PENDING) or completed at that time. If it is pending, a wait for the passed event object is required before checking the IOCTL completion code located in the **retcode** parameter of each IOCTL control block.

The caller *must* initialize the file object pointer describing the connection with the device driver it obtained from a successful **IoGetDeviceObjectPointer** function inside the I/O stack location it allocates for the IRP recipient.

The following legend applies to parameters in the **DeviceIoControl** and **IoBuildDeviceIoControlRequest** functions. (Refer to either the *Microsoft Windows NT Device Driver Kit* book or the *Microsoft Windows 98 Device Driver Kit* book for more information.)

Command IOCTL control code for the operation. Mnemonic name details can be found in "Function and I/O Control Codes" on page 4-3.

InBuffer Input buffer pointer to be passed to the device driver. In *all* cases, this is the input IOCTL control block pointer containing IOCTL parameters.

InBufferSize Size of the **InBuffer** block (in bytes).

OutBuffer Output buffer pointer to be passed to the device driver. In *all* cases, this is the IOCTL control block pointer receiving the completion code and can be either the same block pointer as the one passed in the **InBuffer** parameter or another location having at least the same size as the **InBufferSize** specified.

OutBufferSize Size of the **OutBuffer** block (in bytes).

This legend applies to IOCTL descriptions:

Format IOCTL control block containing input and output parameters associated with the request to be performed. The IOCTL control block memory pointer must be set in the **InBuffer** parameter.

Parameters Each input parameter contained in the IOCTL control block is explained in that section.

Returns Each output parameter contained in the IOCTL control block is explained in that section.

retcode This is the IOCTL completion code. A value of 0 means that the IOCTL operation has been successfully performed. The return code parameter is valid only when the IOCTL is complete at all stages (in layered IRPs). From a user mode application, the **DeviceIoControl** function must return the TRUE value. From a kernel mode driver, the **IoCallDriver** function must return a STATUS_SUCCESS value, or the event is signaled and the IRP IoStatus parameter is set to STATUS_SUCCESS.

Important Programming Note

Application threads should not share the device driver handle if they do not synchronize their access to the device driver. If another thread uses the device driver handle to perform another IOCTL while the first thread's IOCTL is pending, the first thread's IOCTL exits with the E_ICA_INTR return code and the second thread's IOCTL is performed. Another way to do this is to have each thread acquiring its own handle to the device driver. This behavior, dictated by the Windows NT I/O manager, is particularly disturbing for threads waiting on tasks interrupts or special events. For these particular IOCTLs, the user should use the semaphore registration feature that alleviates the device driver handle sharing restriction which was previously described. See “Interrupt Register” on page 4-18 and “Special Events Register” on page 4-27 functions for more information.

Get Buffer Addresses

Purpose Gets the address and length of a task's input, output, and secondary status buffers.

Command ICAGETBUFADDRS

Format

```
typedef struct
{
   OUT ULONG      retcode;             // IOCTL completion code
   IN UCHAR       coprocnum;           // Co-processor adapter number
   IN UCHAR       tasknum;             // Task number
   UCHAR          align[2];            // Alignment
   OUT ICABUFFER  ib;                  // Input buffer information
   OUT ICABUFFER  ob;                  // Output buffer information
   OUT ICABUFFER  ssb;                 // SS buffer information
   IN ULONG       reserved;            // Reserved parameter (must be 0)
} ICAGETBUFADDRS_PARMS, * PICAGETBUFADDRS_PARMS;
```

Parameters

coprocnum The logical number of the co-processor adapter.

tasknum The task number on the co-processor adapter.

Returns

ib, ob, and ssb The addresses of three structures to receive the address and length of the input structure, the output structure, and the secondary status buffers. Each structure has the following format:

```
typedef struct
{
  OUT USHORT   length;    // Length of buffer
  OUT USHORT   offset;    // Offset of buffer address
  OUT UCHAR    page;      // Page of buffer address
  OUT UCHAR    align[3];  // Alignment
} ICABUFFER;
```

where the parameters are defined as follows:

ib

ib.length The length of the task's input buffer.
ib.offset The page offset of the task's input buffer.
ib.page The page number of the task's input buffer.

ob

ob.length The length of the task's output buffer.
ob.offset The page offset of the task's output buffer.
ob.page The page number of the task's output buffer.

ssb

ssb.length The length of the task's secondary status buffer.
ssb.offset The page offset of the task's secondary status buffer.
ssb.page The page number of the task's secondary status buffer.

retcode The return code set by the device driver:

E_ICA_BAD_OPEN_HANDLE
E_ICA_INVALID_REQUEST
E_ICA_INVALID_COPROC
E_ICA_INVALID_TASK_STATUS
E_ICA_INVALID_TASK

Remarks The **Get Buffer Addresses** function returns the addresses in page:offset format only.

The **Get Buffer Address** function performs the same function as the C Language Interface routines **IcaInBuf**, **IcaOutBuf**, and **IcaSecStatBuf**.

Note: By convention, input buffer is "input" to the system unit, and output buffer is "output" from the system unit to the co-processor adapter.

Related Topics None

Get Parameters

Purpose Obtains the configuration parameter information for a co-processor adapter.

Command ICAGETPARMS

Format

```
typedef struct
{
   OUT ULONG   retcode;         // IOCTL completion code
   IN UCHAR    coprocnum;       // Co-processor adapter number
   UCHAR       align[3];        // Alignment
   IN ULONG    reserved;        // Reserved parameter (must be 0)
   OUT ICAPARM cfgparms;        // Configuration parameters
} ICAGETPARMS_PARMS, * PICAGETPARMS_PARMS;

typedef struct
{
   OUT ULONG    io_addr;      // Address of I/O ports
   OUT ULONG    maxtask;      // Maximum task number
   OUT ULONG    maxpri;       // Maximum task priorities
   OUT ULONG    maxqueue;     // Maximum queues
   OUT ULONG    maxtime;      // Maximum timers
   OUT ULONG    int_level;    // Adapter interrupt level
   OUT ULONG    ssw_size;     // Shared storage window size
   OUT ULONG    adapter_type; // Adapter type
   OUT ULONG    adapter_mem;  // Memory installed
} ICAPARMS;
```

Parameters

coprocnum The logical number of the co-processor adapter.

Returns

cfgparms Structure set by the device driver with the configuration parameters for the specified co-processor adapter, where the parameters are defined as follows:

io_addr The base I/O address of the co-processor adapter's I/O ports. These ports are used by the device driver for controlling the co-processor adapter.

Note: See Chapter 2, "Registry Description for Windows NT" for more information about the **maxtask, maxpri, maxqueue** and **maxtime** parameters that follow.

maxtask The highest task number that can be loaded on the co-processor adapter.

maxpri The highest value of a task's priority. The highest priority level is 1, whereas the lowest priority level has the maximum value.

maxqueue The highest queue number that can be allocated on the co-processor adapter.

maxtime The highest timer number that can be allocated on the co-processor adapter.

int_level The interrupt level on which the co-processor adapter interacts with the system unit.

ssw_size A code indicating the size of the shared storage window.

The following table indicates what size window each value represents:

Size Code	Window Size (in KB)
0	8
1	16
2	32
3	64

adapter_type The type of co-processor adapter installed. The following values are defined.

Type	Value	Name
PCI	IBM_X25PCI_ADAPTER IBM_8PORT_PCI_ADAPTER IBM_MM2ISA_PCI_ADAPTER	IBM ARTIC X.25 PCI IBM ARTIC186 8-Port PCI IBM ARTIC186 Model II ISA/PCI
ISA	IBM_X25ISA_ADAPTER IBM_DLPISA_ADAPTER IBM_MM2ISA_ADAPTER IBM_M8PISA_ADAPTER	IBM ARTIC X.25 ISA IBM ARTIC Dual Port IBM ARTIC Multiport Model II IBM ARTIC Multiport IBM ARTIC Multiport 8-Port 232 IBM ARTIC186 8-Port Adapter
MCA	IBM_X25MCA_ADAPTER IBM_PMAMCA_ADAPTER IBM_MP2MCA_ADAPTER	IBM ARTIC X.25 MCA IBM ARTIC PortMaster Adapter/A IBM ARTIC Multiport/2

adapter_mem Size (in bytes) of the co-processor adapter's memory accessible by the system unit. (The actual physical amount of memory mounted on the co-processor adapter might be greater than the value defined here.) The following values are defined:

- 0x80000
  512KB
- 0xf0000
  960KB
- 0x100000
  1MB
- 0x200000
  2MB

retcode The return code set by the device driver:

E_ICA_BAD_OPEN_HANDLE
E_ICA_INVALID_REQUEST
E_ICA_INVALID_COPROC

Remarks Some of the parameters returned by the **Get Parameters** function (**maxtask**, **maxpri**, **maxtime**, and **maxqueue**) can be defined in the registry. The device driver uses the parameters or defaults when loading the Realtime Control Microcode on a co-processor adapter.

The **Get Parameters** function performs the same function as the C Language Interface routine **IcaGetParms**.

Related Topics None

Get Primary Status

Purpose Gets the primary status byte for a task.

Command ICAGETPRIMSTAT

Format

```
typedef struct
{
   OUT ULONG      retcode;         // IOCTL completion code
   IN UCHAR       coprocnum;       // Co-processor adapter number
   IN UCHAR       tasknum;         // Task number
   OUT UCHAR      psb;             // Primary Status byte
   UCHAR          align;           // Alignment
   IN ULONG       reserved;        // Reserved parameter (must be 0)
} ICAGETPRIMSTAT_PARMS, * PICAGETPRIMSTAT_PARMS;
```

Parameters

coprocnum The logical number of the co-processor adapter.

tasknum The task number on the co-processor adapter.

Returns

psb The value of the task's primary status byte set by the device driver.

retcode The return code set by the device driver:

E_ICA_BAD_OPEN_HANDLE
E_ICA_INVALID_REQUEST
E_ICA_INVALID_COPROC
E_ICA_INVALID_TASK

Remarks See the *ARTIC Firmware Technical Reference* for the definition of the bits in the primary status byte.

The **Get Primary Status** function performs the same function as the C Language Interface routine **IcaGetPrimStat**.

Related Topics None

Get Version

Purpose Gets the release level of this version of the device driver.

Command ICAGETVER

Format

```
typedef struct
{
   OUT ULONG    retcode;             // IOCTL completion code
   OUT UCHAR    minvc;               // Minor version code
   OUT UCHAR    majvc;               // Major version code
   UCHAR        align[2];            // Alignment
   IN ULONG     reserved;            // Reserved parameter (must be 0)
} ICAGETVER_PARMS, * PICAGETVERPARMS;
```

Parameters

None

Returns

minvc The minor version code of the device driver.

majvc The major version code of the device driver.

retcode The return code set by the device driver:

E_ICA_BAD_OPEN_HANDLE

Remarks The **Get Version** function performs the same function as the C Language Interface routine **IcaGetVer**.

Related Topics None

Interrupt Deregister

Purpose Cancels the application's request to be notified of a specific task interrupt.

Command ICAINTDEREG

Format

```
typedef struct
{
   OUT ULONG   retcode;         // IOCTL completion code
   IN UCHAR    coprocnum;       // Co-processor adapter number
   IN UCHAR    tasknum;         // Task number
   UCHAR       align[1];        // Alignment
   IN UCHAR    semset;          // Semaphore presence in semhandle
   IN PVOID    semhandle;       // Semaphore handle/object
} ICAINTDEREG_PARMS, * PICAINTDEREG_PARMS;
```

Parameters

coprocnum The logical number of the co-processor adapter.

tasknum The task number on the co-processor adapter.

semset Flag indicating the deregistration of a semaphore. The semaphore handle/object is set in the **semhandle** parameter.

- If the flag value is 0, the **semhandle** parameter is ignored and the active wait is deregistered.
- If the flag value is 1, the **semhandle** parameter is used to deregister the semaphore.

Note: To avoid deadlocks in a user thread waiting for the semaphore to be deregistered, the semaphore is released prior to being deregistered.

semhandle The semaphore handle or object to deregister. This parameter must contain the same value specified during the corresponding ICAINTREG IOCTL.

Returns

retcode The return code set by the device driver:

E_ICA_BAD_OPEN_HANDLE
E_ICA_INVALID_REQUEST
E_ICA_INVALID_COPROC
E_ICA_NOT_REG
E_ICA_INVALID_TASK
E_ICA_BAD_SEMAPHORE

Remarks The **Interrupt Deregister** function cancels a previous application's request to be notified of a specific task interrupt. Processes should cancel all requests for task interrupt notification prior to terminating. However, if there is any outstanding Interrupt Wait for that client, the Interrupt Wait is terminated with the return code *E_ICA_INTR*.

When used without specifying a semaphore, the **Interrupt Deregister** function performs the same function as the C Language Interface routine **IcaIntDereg**. Otherwise, the **Interrupt Deregister** function performs the same function as the C Language Interface routine **IcaDevRemSemaphore**.

Related Topics Interrupt Register, Interrupt Wait

Interrupt Register

Purpose Registers an application with the device driver for notification of a specific task interrupt.

Command ICAINTREG

Format

```
typedef struct
{
   OUT ULONG   retcode;         // IOCTL completion code
   IN UCHAR    coprocnum;       // Co-processor adapter number
   IN UCHAR    tasknum;         // Task number
   UCHAR       align[1];        // Alignment
   IN UCHAR    semset;          // Semaphore presence in semhandle
   IN PVOID    semhandle;       // Semaphore handle/object
} ICAINTREG_PARMS, * PICAINTREG_PARMS;
```

Parameters

coprocnum The logical number of the co-processor adapter.

tasknum The task number on the co-processor adapter.

semset The flag indicating the registration of a semaphore to release when a specific task interrupt occurs. The semaphore handle/object is set in the **semhandle** parameter.

- If the flag value is 0, the **semhandle** parameter is ignored and an ICAINTWAIT has to be used to perform a wait for the specific task interrupt.
- If the flag value is 1, the **semhandle** parameter is used to register the semaphore. The caller can then wait for the semaphore handle/object using, for example, the **WaitForSingleObject** function (WIN32 API) or the **KeWaitForSingleObject** function (DDK API).

semhandle The semaphore handle or object to register and later release when the specified task's interrupt occurs.

- For user mode applications, this parameter must be a semaphore handle, usually acquired using the **CreateSemaphore** function Win32 API.
- For kernel mode drivers, this parameter must be a pointer on a dispatcher object of type semaphore, usually initialized using the **KeInitializeSemaphore** function (DDK API).

Returns

retcode The return code set by the device driver:

E_ICA_BAD_OPEN_HANDLE
E_ICA_INVALID_REQUEST
E_ICA_INVALID_COPROC
E_ICA_ALREADY_REG
E_ICA_RESOURCE_SHORTAGE
E_ICA_BAD_SEMAPHORE
E_ICA_INVALID_TASK

Remarks The **Interrupt Register** function allows applications to be notified of task interrupts with either the **Interrupt Wait** function or the DDK API wait function. An application must first register using this function before being notified of task interrupts. Note that for the **Interrupt Register** function, the error task (0xFE) is always a valid task number and does not result in the E_ICA_INVALID_TASK return code being returned.

When used without specifying a semaphore, the **Interrupt Register** function performs the same function as the C Language Interface routine **IcaIntReg**. Otherwise, the **Interrupt Register** function performs the same function as the C Language Interface routine **IcaDevRegSemaphore**.

Semaphores are released *each time* a task interrupt occurs. Therefore, counting semaphores should be initialized with a sufficient maximum count. This is a different behavior from the one described in the **Interrupt Wait** function.

Related Topics Interrupt Deregister, Interrupt Wait

Interrupt Wait

Purpose Waits until a specific task on a co-processor adapter interrupts the system unit.

Command ICAINTWAIT

Format

```
typedef struct
{
   OUT ULONG   retcode;         // IOCTL completion code
   IN UCHAR    coprocnum;       // Co-processor adapter number
   IN UCHAR    tasknum;         // Task number
   UCHAR       align[2];        // Alignment
   IN LONG     timeout;         // Timeout
   IN ULONG    reserved;        // Reserved parameter (must be 0)
} ICAINTWAIT_PARMS, * PICAINTWAIT_PARMS;
```

Parameters

coprocnum The logical number of the co-processor adapter.

tasknum The task number on the co-processor adapter.

timeout The time in milliseconds to wait for a task interrupt. If it is 0, the call returns immediately, indicating whether or not a previous task interrupt has occurred. If none has occurred, the return code is set to *E_ICA_TIMEOUT*. If a timeout value of -1 is specified, the wait blocks either indefinitely or until the Interrupt Wait is deregistered.

Returns

retcode The return code set by the device driver:

E_ICA_BAD_OPEN_HANDLE
E_ICA_INVALID_REQUEST
E_ICA_INVALID_COPROC
E_ICA_TIMEOUT
E_ICA_NOT_REG
E_ICA_INVALID_TASK
E_ICA_IN_USE
E_ICA_INTR

Remarks The **Interrupt Wait** function returns immediately with no error if an application previously registered for that task interrupt and the interrupt occurred prior to this call. If the interrupt did not occur previously, this call blocks the user mode application until the specified task interrupts or the time specified in the **timeout** parameter has expired. The kernel mode driver must use its I/O event if it wants to perform a synchronous wait. If multiple interrupts by the same task occur prior to the Interrupt Wait call, the application is notified only once. An application must first register with the Interrupt Register before being notified of task interrupts. Note that for the **Interrupt Wait** function, the error task (0xFE) is always a valid task number and does not result in the E_ICA_INVALID_TASK return code being returned.

The **Interrupt Wait** function should only be used for non-semaphore registrations performed using the **Interrupt Registration** function.

The **Interrupt Wait** function performs the same function as the C Language Interface routine **IcaIntWait**.

Related Topics Interrupt Register

Issue Command

Purpose Issues a command to a task with an option to copy parameter data from an application buffer to the task's output buffer before issuing the command.

Command ICAISSUECMD

Format

```
typedef struct
{
   OUT ULONG   retcode;     // IOCTL completion code
   IN UCHAR    coprocnum;   // Co-processor adapter number
   IN UCHAR    tasknum;     // Task number
   IN USHORT   length;      // Length of parameter buffer
   IN UCHAR    cmdcode;     // Command Code
   UCHAR       align[3];    // Alignment
   IN ULONG    timeout;     // Timeout
   IN PUCHAR   prms;        // Pointer to parameters
   IN ULONG    reserved;    // Reserved parameter (must be 0)
} ICAISSUECMD_PARMS, * PICAISSUECMD_PARMS;
```

Parameters

coprocnum The logical number of the co-processor adapter.

tasknum The task number on the co-processor adapter.

length The number of bytes to be copied to the task's output buffer. A value of 0 indicates that nothing should be written to the task's output buffer.

cmdcode The command code to put in the task's Buffer Control Block (BCB).

timeout The number of milliseconds to wait for the Realtime Control Microcode to respond to the command.

Note: If a value of 0 is specified, the device driver gives Realtime Control Microcode the minimum amount of time to respond to the command, which is 100 microseconds. In heavy traffic conditions, this might not be sufficient and could lead to timeout. Therefore, set a value greater than 0 for this parameter.

prms A pointer to the application buffer containing the data to be written to the task's output buffer. The **prms** parameter is ignored if the **length** parameter is 0.

- For user mode applications, this parameter must be a pageable or non-pageable user space virtual address.
- For kernel mode drivers, this parameter must be a non-pageable or locked kernel space virtual address.

Returns

retcode The return code set by the device driver:

E_ICA_BAD_OPEN_HANDLE
E_ICA_INVALID_REQUEST
E_ICA_RESOURCE_SHORTAGE
E_ICA_INVALID_COPROC

E_ICA_INVALID_TASK_STATUS[1]
E_ICA_TIMEOUT
E_ICA_BAD_PCSELECT
E_ICA_CMD_REJECTED
E_ICA_OB_SIZE
E_ICA_INVALID_TASK
E_ICA_INVALID_FORMAT

Remarks The **Issue Command** function issues a command to a task. The caller has the option of copying parameter information into the task's output buffer before the command is issued.

The timeout value applies only to the time to wait for the acknowledgement from RCM after the command is issued to the task. It does not apply to wait for the task's output buffer to be free if it currently is in the BUSY state.

If the task's output buffer is in the BUSY state, the command is rejected with return code E_ICA_INVALID_TASK_STATUS. It is the responsibility of the application to ensure that any parameter data to be copied to the task's output buffer is in Intel x86 format.

The **Issue Command** function performs the same function as the C Language Interface routine **IcaIssueCmd**.

Related Topics None

[1] A primary status of 0x00 is allowed if the Realtime Control Microcode is not yet loaded, so applications can send commands to ROS.

Read Memory

Purpose Reads from a co-processor adapter's memory into an application buffer.

Command ICAREADMEM

Format

```
typedef struct
{
   OUT ULONG   retcode;         // IOCTL completion code
   IN UCHAR    coprocnum;       // Co-processor adapter number
   IN UCHAR    addr_format;     // Address format
   IN USHORT   segpage;         // Segment/Page
   IN USHORT   offset;          // Offset
   UCHAR       align[2];        // Alignment
   IN ULONG    length;          // Length
   IN PUCHAR   dest;            // Destination buffer pointer
   IN ULONG    reserved;        // Reserved parameter (must be 0)
} ICAREADMEM_PARMS, * PICAREADMEM_PARMS;
```

Parameters

coprocnum The logical number of the co-processor adapter.

addr_format The control parameter determining the address format:

Value	Address Interpretation
0x00	The **segpage** parameter is a segment in co-processor adapter memory, and the **offset** is an offset within that segment.
0xFF	The **segpage** parameter is a page in co-processor adapter memory, and the **offset** is an offset within that page.
0x01	The **segpage** and **offset** parameters are a 32-bit physical address in co-processor adapter memory. The least significant 16-bits are in the **offset** parameter.

Note: All addressing formats are converted to page:offset formats by the device driver prior to accessing co-processor adapter memory. Accesses across pages are handled and physical memory boundaries are checked. Any invalid memory access is indicated without effectively attempting to access the memory. When accessing the upper 1MB on a 2MB IBM ARTIC Portmaster Adapter/A, page:offset format must be used because the segment:offset format can refer only to addresses in the 0–1MB range.

segpage The segment or page of the co-processor adapter memory address.

offset The offset of the co-processor adapter memory address. The interpretation of this parameter is determined by the **addr_format** parameter.

length The number of bytes to be read from the co-processor adapter memory.

dest A pointer to the data buffer where the co-processor adapter memory is to be copied.

- For user mode applications, this parameter must be a pageable or non-pageable user space virtual address.
- For kernel mode drivers, this parameter must be a non-pageable or locked kernel space virtual address.

Returns

retcode The return code set by the device driver:

E_ICA_BAD_OPEN_HANDLE
E_ICA_INVALID_REQUEST
E_ICA_RESOURCE_SHORTAGE
E_ICA_INVALID_COPROC
E_ICA_INVALID_PAGE
E_ICA_INVALID_OFFSET
E_ICA_INVALID_FORMAT

Remarks The **Read Memory** function reads from the co-processor adapter memory into a system unit application's buffer. The address in co-processor adapter memory can be specified either as a segment and offset or as a page and offset.

It is the responsibility of the application to recognize that any data read from the co-processor adapter memory with this call are in Intel x86 format.

The **Read Memory** function performs the same function as the C Language Interface routine **IcaReadMem**.

Related Topics Write Memory

Reset

Purpose Issues a hardware reset to the co-processor adapter.

Command ICARESET

Format

```
typedef struct
{
   OUT ULONG   retcode;           // IOCTL completion code
   IN UCHAR    coprocnum;         // Co-processor adapter number
   UCHAR       align[3];          // Alignment
   IN ULONG    reserved;          // Reserved parameter (must be 0)
} ICARESET_PARMS, * PICARESET_PARMS;
```

Parameters

coprocnum The logical number of the co-processor adapter to be reset.

Returns

retcode The return code set by the device driver:

E_ICA_BAD_OPEN_HANDLE
E_ICA_INVALID_REQUEST
E_ICA_INVALID_COPROC
E_ICA_RESET_FAILED

Remarks The **Reset** function issues a hardware reset to the co-processor adapter. The Realtime Control Microcode (RCM) and all other tasks are unloaded, and the co-processor adapter performs a power-on self test (POST).

A reset of the co-processor adapter may also be performed when loading the Realtime Control Microcode to the co-processor adapter by using the **-reset** application loader option.

RCM parameters (MaxTask, MaxPri, MaxQueue and MaxTime) are refreshed during reset with values read from the registry for the co-processor adapter being reset.

The **Reset** function performs the same function as the C Language Interface routine **IcaReset**.

Related Topics None

Special Events Deregister

Purpose Cancels an application's request to be notified of the Realtime Control Microcode's receipt of an Initialize command.

Command ICASEDEREG

Format

```
typedef struct
{
   OUT ULONG     retcode;          // IOCTL completion code
   IN UCHAR      coprocnum;        // Co-processor adapter number
   UCHAR         align[2];         // Alignment
   IN UCHAR      semset;           // Semaphore presence in semhandle
   IN PVOID      semhandle;        // Semaphore handle/object
} ICASEDEREG_PARMS;
```

Parameters

coprocnum The logical number of the co-processor adapter.

semset The flag indicating the deregistration of a semaphore. The semaphore handle/object is set in the **semhandle** parameter.

- If the flag value is 0, the **semhandle** parameter is ignored and the active wait is deregistered.
- If the flag value is 1, the **semhandle** parameter is used to deregister the semaphore.

Note: To avoid deadlocks in a user thread waiting for the semaphore to be deregistered, the semaphore is released prior to being deregistered.

semhandle The semaphore handle or object to deregister. This parameter must contain the same value specified during the corresponding **Special Events Register** function.

Returns

retcode The return code set by the device driver:

E_ICA_BAD_OPEN_HANDLE
E_ICA_INVALID_REQUEST
E_ICA_INVALID_COPROC
E_ICA_NOT_REG
E_ICA_BAD_SEMAPHORE

Remarks The **Special Events Deregister** function cancels a previous application's request to be notified of the Realtime Control Microcode's receipt of an Initialize command. Processes should cancel all requests for such notification prior to terminating.

When used without specifying a semaphore, the **Special Event Deregister** function performs the same function as the C Language Interface routine **IcaSeDereg**. Otherwise, the **Special Event Deregister** function performs the same function as the C Language Interface routine **IcaDevRemNotify**.

Related Topics Special Event Register

Special Events Register

Purpose Registers an application with the device driver for notification of the Realtime Control Microcode's receipt of an Initialize command.

Command ICASEREG

Format

```
typedef struct
{
   OUT ULONG     retcode;          // IOCTL completion code
   IN UCHAR      coprocnum;        // Co-processor adapter number
   IN UCHAR      ctrlflag;         // Control flag
   UCHAR         align[1]          // Alignment
   IN UCHAR      semset;           // Semaphore presence in semhandle
   IN PVOID      semhandle;        // Semaphore handle/object
} ICASEREG_PARMS, * PICASEREG_PARMS;
```

Parameters

coprocnum The logical number of the co-processor adapter.

ctrlflag Control bits indicating the events for which the application should be registered. One bit is currently defined in the control flag: 0x80. Setting of this bit indicates that the application should be registered for Initialize commands issued to the Realtime Control Microcode on the co-processor adapter.

semset The flag indicating the registration of a semaphore to release when a special event occurs. The semaphore handle/object is set in the **semhandle** parameter.

- If the flag value is 0, the **semhandle** parameter is ignored and an ICASEWAIT must be used to perform a wait for the special events.
- If the flag value is 1, the **semhandle** parameter is used to register the semaphore. The caller can then wait for the semaphore handle/object using, for example, the **WaitForSingleObject** function (WIN32 API) or the **KeWaitForSingleObject** function (DDK API).

semhandle The semaphore handle or object to register and later release when a special event occurs.

- For user mode applications, this parameter must be a semaphore handle, usually acquired using the **CreateSemaphore** function (Win32 API).
- For kernel mode drivers, this parameter must be a pointer on a dispatcher object of type semaphore, usually initialized using the **KeInitializeSemaphore** function (DDK API).

Returns

retcode The return code set by the device driver:

E_ICA_BAD_OPEN_HANDLE
E_ICA_INVALID_REQUEST
E_ICA_INVALID_COPROC
E_ICA_INVALID_CONTROL
E_ICA_ALREADY_REG
E_ICA_RESOURCE_SHORTAGE
E_ICA_BAD_SEMAPHORE

Remarks The **Special Events Register** function allows applications to be notified of Initialize commands issued to the Realtime Control Microcode with the **Special Events Wait** function. An application must first register with this function before being notified of Initialize commands issued to the Realtime Control Microcode.

Semaphores are released *each time* a special event occurs. Therefore, counting semaphores should be initialized with a sufficient maximum count. This is a different behavior from the one described in the **Special Events Wait** function.

When used without specifying a semaphore, the **Special Events Register** function performs the same function as the C Language Interface routine **IcaSeReg**. Otherwise, the **Special Events Register** function performs the same function as the C Language Interface routine **IcaDevNotify**.

Related Topics Special Events Deregister, Special Events Wait

Special Events Wait

Purpose Waits until the Realtime Control Microcode on a co-processor adapter receives an Initialize command.

Command ICASEWAIT

Format

```
typedef struct
{
   OUT ULONG      retcode;         // IOCTL completion code
   IN UCHAR       coprocnum;       // Co-processor adapter number
   UCHAR          align[3];        // Alignment
   IN LONG        timeout;         // Timeout
   IN ULONG       reserved;        // Reserved parameter (must be 0)
} ICASEWAIT_PARMS, * PICASEWAIT_PARMS;
```

Parameters

coprocnum The logical number of the co-processor adapter.

timeout The time in milliseconds to wait for the Realtime Control Microcode to receive an Initialize command. If it is 0, the call returns immediately, indicating whether or not the Realtime Control Microcode has previously received an Initialize command. If no Initialize command was received, the return code is set to *E_ICA_TIMEOUT*. If a timeout value of -1 is specified, the wait blocks indefinitely (or until the Special Event Wait is deregistered).

Returns

retcode The return code set by the device driver:

E_ICA_BAD_OPEN_HANDLE
E_ICA_INVALID_REQUEST
E_ICA_INVALID_COPROC
E_ICA_INVALID_TASK_STATUS
E_ICA_TIMEOUT
E_ICA_NOT_REG
E_ICA_INTR
E_ICA_IN_USE

Remarks The **Special Events Wait** function returns immediately with no error if an application previously registered and the Realtime Control Microcode received an Initialize command (0x00) prior to this call. If the Realtime Control Microcode has not yet received the Initialize command, the call blocks the user mode application until the Initialize command is received by the Realtime Control Microcode or the time specified in the **timeout** parameter expires. The kernel mode driver must use the I/O event to perform a synchronous wait.

The **Special Events Wait** function performs the same function as the C Language Interface routine **IcaSeWait**.

Related Topics Special Events Register

Write Memory

Purpose Writes to a co-processor adapter's memory from an application buffer.

Command ICAWRITEMEM

Format

```
typedef struct
{
   OUT ULONG   retcode;         // IOCTL completion code
   IN UCHAR    coprocnum;       // Co-processor adapter number
   IN UCHAR    addr_format;     // Address format
   IN USHORT   segpage;         // Segment/Page
   IN USHORT   offset;          // Offset
   UCHAR       align[2];        // Alignment
   IN ULONG    length;          // Length
   IN PUCHAR   source;          // Source buffer pointer
   IN ULONG    reserved;        // Reserved parameter (must be 0)
} ICAWRITEMEM_PARMS, * PICAWRITEMEM_PARMS;
```

Parameters

coprocnum The logical number of the co-processor adapter.

addr_format The control parameter determining the address format:

Value	Address interpretation
0x00	The **segpage** parameter is a segment in co-processor adapter memory, and the **offset** is an offset within that segment.
0xFF	The **segpage** parameter is a page in co-processor adapter memory, and the **offset** is an offset within that page.
0x01	The **segpage** and **offset** parameters are a 32-bit physical address in co-processor adapter memory. The least significant 16-bits are in the **offset** parameter.

Note: All addressing formats are converted to page:offset formats by the device driver prior to accessing co-processor adapter memory. Accesses across pages are handled and physical memory boundaries are checked. Any invalid memory access is indicated without effectively attempting to access the memory. When accessing the upper 1MB on a 2MB IBM ARTIC Portmaster Adapter/A, page:offset format must be used because the segment:offset format can only refer to addresses in the 0–1MB range.

segpage The segment or page of the co-processor adapter memory address.

offset The offset of the co-processor adapter memory address. The interpretation of this parameter is determined by the **addr_format** parameter.

length The number of bytes to be written to the co-processor adapter memory.

source A pointer to the data buffer containing data to copy into the co-processor adapter's memory.

- For user mode applications, this parameter must be a pageable or non-pageable user space virtual address.
- For kernel mode drivers, this parameter must be a non-pageable or locked kernel space virtual address.

Returns

retcode The return code set by the device driver:

E_ICA_BAD_OPEN_HANDLE
E_ICA_INVALID_REQUEST
E_ICA_RESOURCE_SHORTAGE
E_ICA_INVALID_COPROC
E_ICA_INVALID_PAGE
E_ICA_INVALID_OFFSET
E_ICA_INVALID_FORMAT

Remarks The **Write Memory** function writes to the co-processor adapter's memory from a system unit application's buffer. The address in co-processor memory can be specified either as a segment and offset or as a page and offset.

The **Write Memory** function performs the same function as the C Language Interface routine **IcaWriteMem**.

Related Topics

Read Memory

Chapter 5. Application Loader Utility

The Application Loader utility (**icaldric**) is used to load the Realtime Control Microcode or tasks to the co-processor adapter. The Application Loader utility consists of two files:

- An executable file **icaldric.exe** that can be invoked from the keyboard, a command file, or an application program.
- A message file **icaldric.msg** containing messages that can be displayed to standard output or standard error by the Application Loader utility.

The following Application Loader utility files are in (product installation directory)\bin.

icaldric.exe	Application program loader
icaldric.msg	Application loader message file

The first task loaded onto a co-processor adapter must be the Realtime Control Microcode (file **icaaim.com** or **icarcm.com**), provided with the co-processor adapter or the ARTIC Windows support program file. After the Realtime Control Microcode is loaded, you can load applications or other tasks onto the co-processor adapter. Each task must have a unique task number that is assigned when the task is loaded. The **MAXTASK** field in the registry defines the highest task number under which a task can be loaded. For additional information about the registry, see Chapter 2, “Registry Description for Windows NT” and Chapter 3, “Registry Description for Windows 98.”

The following example resets co-processor adapter 0 and loads the Realtime Control Microcode onto co-processor adapter 0:

```
icaldric 0 icaxxx.com 0 -reset
```

where:

xxx = **aim** for the following co-processor adapters:

IBM ARTIC X.25 PCI
IBM ARTIC X.25 ISA
IBM ARTIC X.25 MCA
IBM ARTIC Dual Port
IBM ARTIC Multiport
IBM ARTIC Multiport Model/2
IBM ARTIC Multiport 8-Port 232
IBM ARTIC186 8-Port
IBM ARTIC186 8-Port PCI

and *xxx* = **rcm** for the following co-processor adapters:

IBM ARTIC Multiport Model II
IBM ARTIC Portmaster Adapter/A
IBM ARTIC186 Model II ISA/PCI

The first zero represents the first logical co-processor adapter. The last zero represents the task number, which is always zero for the Realtime Control Microcode.

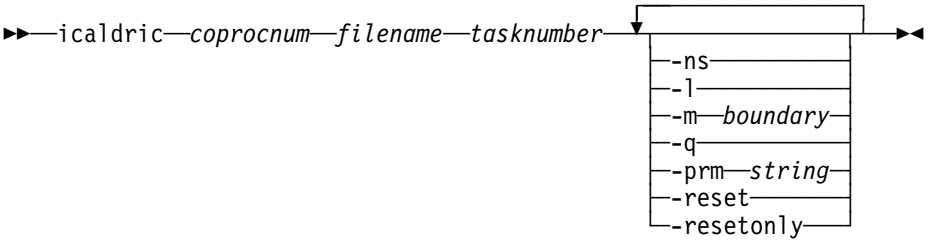
The Application Loader utility can load **.com** or **.exe** files only. The maximum length of a **.com** file is 64KB, whereas the length of an **.exe** file is restricted by the amount of free storage on the co-processor adapter at the time the load is attempted. (Refer to the *ARTIC Firmware Technical Reference* or the *ARTIC C Language Support Version 1.03.01 User's Guide, Volume II - Co-Processor Adapter*, if programming in C Language.)

The Application Loader utility sets up the initial values for the Code Segment Register, Data Segment Register, Stack Segment Register, and Stack Pointer Register in the task header.

Starting the Application Loader Utility

The Application Loader utility requires command line parameters to indicate which task to load and how it should be loaded. The first four parameters are required and must appear in the order in which they are shown. The remaining flags are optional and may appear in any order. Flags and parameters are separated by spaces and/or tab characters.

Application Loader Utility Syntax



icaldric The name of the application loader program.

Parameter	Description
coprocnum	The logical co-processor adapter number where the task is to be loaded. This parameter is required and must be first on the command line.
filename	The file name of the task to be loaded. You can specify a full path name if the task file is not in the current working directory. This parameter is required and must be second on the command line.
tasknumber	The task number to be assigned to the task to be loaded. Realtime Control Microcode must be loaded as task 0. Application tasks may be loaded in the range of 1 to the **MAXTASK** value, as assigned in the parameter file. This parameter is required and must be third on the command line.
-ns	The no-start flag. If specified, the task is loaded onto the co-processor adapter only; it is not started. If not used, the task is started by default. This parameter is optional.
-l	The load-low flag for loading in low co-processor adapter memory. Use **-l** (el) to load the task at the lowest possible address. Omit the **-l** flag to load the task at the highest possible address. This parameter is optional.
-m *boundary*	The memory boundary in paragraphs on which the task should be loaded. The default task load boundary is one paragraph (16 bytes). If a boundary other than one paragraph is required, you can enter the required boundary using this option. The boundary must be an exact power of 2 (only one bit on in the entire word). The boundary is specified in hexadecimal format. This parameter is optional.
-q	The quiet flag. If specified, the **icaldric** command does not display any messages to standard out or to standard error, even if an error occurs. If messages are suppressed, the application loader sends a return code upon completion. This return code is the same as the

application loader message number. Therefore, if messages are suppressed and a non-zero return code is received from the application loader, the meaning of the return code can be found by looking at the application loader message with the same message number. For a description of the application loader messages, see "Application Loader Utility Information Messages" on page D-1. The default is to report the success or failure of the load to standard out or standard error. This parameter is optional.

-prm *string* The parameter string passed to the task parameter block area (offset 0x1C in the task's header segment). The maximum length of the parameter area is 128 bytes. The parameters are passed as a NULL terminated string. To ensure the parameter string is passed correctly, enclose multiple-word strings, special characters, and escape sequences in double quotation marks.

Note: When loading Realtime Control Microcode as task 0 on the Realtime Interface Co-Processor Portmaster Adapter/A, if a **-prm** string is not explicitly specified, the application loader uses the following parameter string to disable peer services:

```
ICARCM.COM 1
```

Peer services are not supported by the ARTIC Windows support.

-reset This flag causes a reset of the indicated co-processor adapter prior to loading task 0. It is ignored for tasks other than task 0. Note that the reset flag causes task 0 load time to be increased by up to 20 seconds, which is the time it takes the co-processor adapter to execute its self-test.

-resetonly This flag causes a reset of the indicated co-processor adapter.

Note: The Realtime Control Microcode will not be loaded on the co-processor adapter.

Examples

In the following example, the task USERTASK.EXE is loaded on co-processor adapter 1 as task 2 with messages suppressed. All other parameters have the default values.

```
icaldric 1 USERTASK.EXE 2 -q
```

The following example loads TASK.EXE as task 1 on co-processor adapter 0.

```
icaldric 0 TASK.EXE 1
```

The next example loads TASK.EXE as task 2 on co-processor adapter 1. The task is passed the parameter string "TASK.EXE parameter string".

```
icaldric 1 TASK.EXE 2 -prm "parameter string"
```

Note that the parameter string is enclosed by quotation marks to include the spaces in the parameter string.

Application Loader Utility Messages and Return Codes

Application Loader utility messages are displayed to show the status of the application loader's operation. These messages are listed in "Application Loader Utility Information Messages" on page D-1. The Application Loader utility also returns corresponding numeric values as its program return value. These return codes are described in "Application Loader Utility Return Codes" on page C-3.

Chapter 6. Online Dump Utility

The Online Dump utility is a debugging tool that dumps the memory contents and I/O port values of a co-processor adapter to a disk file. These can then be formatted using **frmtdump**, the Dump Formatter utility. Dump data can be obtained interactively by the user or automatically with the AutoDump feature. The Online Dump utility consists of two files:

- An executable file **icadpric.exe** that can be invoked from the keyboard, a command file, or an application program.
- A message file **icadpric.msg** that contains all the messages that can be displayed to standard output or standard error by the Online Dump utility.

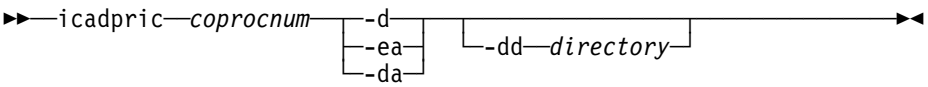
The following Online Dump utility files are in (product installation directory)\bin.

icadpric.exe	Online dump utility
icadpric.msg	Online dump utility message file

Starting the Online Dump Utility

The Online Dump utility requires command line parameters. These are shown in the following syntax diagram:

Online Dump Utility Syntax



icadpric The name of the Online Dump utility program.

Parameter	Description
coprocnum	The logical number of the co-processor adapter to be dumped. The co-processor adapter number is determined by the co-processor adapter's position relative to other co-processor adapters in the system.
-d	Indicates that a dump of the co-processor adapter identified by **coprocnum** should be done immediately. If AutoDump has previously been enabled (armed), the dump is done immediately. After the dump is complete, AutoDump is no longer enabled.
-ea	Indicates that the co-processor adapter identified by **coprocnum** should be dumped whenever a level 1 error occurs on the co-processor adapter. This is referred to as the *AutoDump feature.*
-da	Cancels a previous request for an Auto Dump on the co-processor identified by **coprocnum**.

Note: When the user selects which co-processor adapter to arm for AutoDump, the Online Dump utility acquires the breakpoint and watchdog timer vectors on the co-processor adapter. It also sets the watchdog timer to a

timeout length of approximately 12 milliseconds. When the watchdog timer expires or a breakpoint is reached (level 1 error), the co-processor adapter is automatically dumped without user intervention. The user should verify that the dump drive has enough free storage before arming a co-processor adapter for AutoDump. The dump drive is the drive determined at the time the co-processor adapter is armed for the dump.

-dd *directory* Specifies the directory where the dump files are to be located. If this parameter is omitted, the dump directory is the current directory.

Output Files

A dump creates two files:

ICAME*n*.DMP The file that contains the memory image of the co-processor adapter.

ICASYS*n*.DMP A file containing system information (in binary format) such as the co-processor adapter software version number, the co-processor adapter register contents, I/O ports, the free memory listing, and task information.

The names of the files produced by the Online Dump utility are based on the co-processor adapter number where *n* is the logical number of the co-processor adapter that was dumped.

Example

The following example enables AutoDump on co-processor adapter 2 and directs the output of any resulting dump to be written to the **\tmp** directory.

```
icadpric 2 -ea -dd\tmp
```

The following command can be used to cancel a previously enabled AutoDump on co-processor adapter 2:

```
icadpric 2 -da
```

The following example performs an immediate dump of co-processor adapter 0 placing the resulting dump files in the current directory:

```
icadpric 0 -d
```

Online Dump Utility Messages and Return Codes

The Online Dump utility displays messages to show the status of the dump program operation. These messages are listed in “Online Dump Utility Information Messages” on page D-4. The Online Dump utility also returns corresponding numeric values as its program return value. These return codes are described in “Online Dump Utility Return Codes” on page C-6.

Chapter 7. Dump Formatter Utility

The Dump Formatter utility converts the machine-readable images generated by the Online Dump utility into a format that can be viewed and/or printed. The Formatter organizes the dump data into an easy-to-read format, using headers and blocks to group related information. The Dump Formatter utility consists of three files:

- An executable file **frmtdump.exe** that can be invoked from the keyboard, a command file, or an application program.
- A message file **frmtdump.msg** that contains all the messages that can be displayed to standard output or standard error by the Dump Formatter utility program.
- A profile **frmtdump.pro** that can be used to tailor the output of the formatter for different printers and cause the display of select areas of memory when the memory image file is generated.

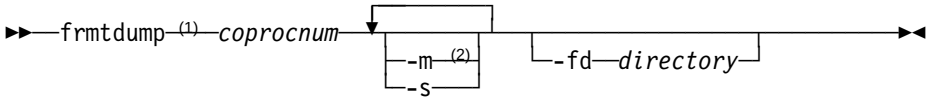
The following Dump Formatter utility files are in (product installation directory)\bin.

frmtdump.exe	Dump formatter utility
frmtdump.msg	Dump formatter utility message file
frmtdump.pro	Dump formatter utility profile

Starting the Dump Formatter Utility

The Dump Formatter utility requires command line parameters. These are shown in the following syntax diagram:

Dump Formatter Utility Syntax



Notes:

1. The corresponding dump files ICAME *n*.DMP or ICASYS*n*.DMP, or both, must be in the current directory.
2. One or more of the flags in this group can be used in any order, but each flag can be specified only once.

frmtdump The name of the Dump Formatter utility program.

Parameter	**Description**
coprocnum	The logical co-processor adapter number *n* of the co-processor adapter that produced the dump files **ICAME*n*.DMP** and **ICASYS*n*.DMP**.
-m	Generates the memory image file from the file **ICAME*n*.DMP**, where *n* is the logical co-processor adapter number of the co-processor adapter that produced the memory dump file. The file **ICAME*n*.DMP** must be in the current directory. If both this and the **-s** flag are omitted, the Dump Formatter utility produces both a memory image file and a system information file.

-s Generates the system information file from the file **ICASYS*n*.DMP**, where *n* is the logical co-processor adapter number of the co-processor adapter that produced the system file. The file **ICASYS*n*.DMP** must be in the current directory. If both this and the **-m** flag are omitted, the Dump Formatter utility produces both a memory image file and a system information file.

-fd *directory* Specifies the directory where the formatted files are to be located. If this parameter is omitted, the files will be in the current directory.

Example

The following example formats the system information dump file for co-processor adapter 1 and places the formatted output in the **c:\me\my_formatter_output_dir** directory:

```
frmtdump 1 -s -fd c:\me\my_formatter_output_dir
```

Output Files

The Dump Formatter utility creates a memory image file or a system information file (or both). The memory image file is an ASCII representation of an Online Dump Formatter utility memory image file. The system information file is an ASCII representation of a Dump Formatter utility system file. See Appendix A, “Output File Format for Dump Formatter Utility” for more details on the format of the files produced by the Dump Formatter utility.

The names of the files produced by the Dump Formatter utility follow:

- **MEMORY*n*.PRT**, the memory image file, where *n* is the logical number of the co-processor adapter that produced the memory image dump file.
- **SYSINFO*n*.PRT**, the system information file, where *n* is the logical number of the co-processor adapter that produced the system information dump file.

Profile

The output of the Dump Formatter utility can be tailored for different printers and select portions of co-processor memory can be displayed by setting parameters in the Dump Formatter utility profile **frmtdump.pro**. If no profile exists, the default parameters are used. The default parameters are listed under “Default Profile” on page 7-8.

The following conventions apply to parameters in **frmtdump.pro**:

- Each parameter must begin on a new line.
- Numbers can be entered in decimal or hexadecimal format. Hexadecimal numbers must be immediately followed by an uppercase or lowercase **h**. Unless otherwise specified, any number outside the proper range is ignored.
- Commas or blanks can be used as delimiters in lists of integers.

Profile Parameters

In listing the parameters, the following assumptions are made:

- An integer in the range 0h through FFh (0 through 255 decimal) inclusive is represented by **nn**.
- An integer in the range 0h through FFFFh (0 through 65535 decimal) inclusive is represented by **NN**.
- Brackets ([ ]) indicate an optional parameter.
- The vertical bar (|) represents a choice. One of the options separated by a vertical bar can be chosen.

Following is a list of the Dump Formatter utility profile parameters:

BOXCHARS

Printer codes for box characters

The printer codes for generating box characters in the output files can be specified by adding the following line to the profile:

```
BOXCHARS  = nn [,] nn [,] nn [,] nn [,] nn [,] nn [,]
            nn [,] nn [,] nn [,] nn [,] nn
```

Each **nn** represents the ASCII character code of a box character. The 11 codes are assigned in the order listed in the following table.

| Vertical Bar	Default value = B3h (179 decimal)
– Horizontal Bar	Default value = C4h (196 decimal)
┌ Upper Left Corner	Default value = DAh (218 decimal)
┐ Upper Right Corner	Default value = BFh (191 decimal)
┘ Lower Right Corner	Default value = D9h (217 decimal)
└ Lower Left Corner	Default value = C0h (192 decimal)
┤ Horizontal Line on Vertical Bar (left)	Default value = B4h (180 decimal)
├ Horizontal Line on Vertical Bar (right)	Default value = C3h (195 decimal)
┬ T Junction	Default value = C2h (194 decimal)
┴ Inverted T Junction	Default value = C1h (193 decimal)
┼ Cross	Default value = C5h (197 decimal)

FORM_FEED

Printer formfeed sequence

This is the printer sequence for generating a formfeed. To set this parameter, add the following line to the profile:

```
FORM_FEED = NONE | nn[[,]nn]...
```

A list of up to 16 ASCII character codes can be entered for the formfeed sequence. This allows the user to set the profile for whatever printer is being used. If more than 16 codes are specified, or if any code exceeds 255, the default value 0Ch (12 decimal) is used. 0Ch is the standard ASCII code for formfeed.

Specifying **FORM_FEED = NONE** indicates that no formfeed character exists for the printer being used. Instead, blank lines are printed in place of the formfeed character to bring the printer to the

top of the next page. The PAGE_LINES parameter indicates how many blank lines to print.

LONG

Complete memory listings

If the keyword **LONG** is included in the profile, every line of memory is displayed, regardless of the contents of memory. Omitting the keyword **LONG** can make the memory file **MEMORY*n*.PRT** shorter because redundant lines of memory are not redisplayed.

MEMLIST

Memory dumped by location

This parameter affects the contents of **MEMORY*n*.PRT**, but not of **SYSINFO*n*.PRT**. It specifies which blocks of memory are to be included in the output.

To set this parameter, add the following line to the profile:

```
MEMLIST = [ ALL ] [ NONE ] [ (NN [,]
[+]NN) [,] ] ...
```

Specifying **ALL** means that all co-processor adapter memory locations will be displayed in **MEMORY*n*.PRT**. Specifying **NONE** means that none of co-processor adapter memory will be displayed in **MEMORY*n*.PRT**.

Ranges of co-processor adapter memory can be specified in terms of paragraphs (16 bytes). Ranges can be either a lower and upper boundary or a lower boundary and a length. Specifying **(NN1 NN2)** gives memory contents from paragraphs **NN1** to **NN2**, inclusive. **NN1** and **NN2** specify paragraph boundaries and are numbers with the same range as described previously. Either number can be the smaller of the two. They do not need to be ordered.

Specifying **(NN1 +NN2)** gives memory contents starting at paragraph **NN1** and continuing for **NN2** paragraphs. For example,

```
MEMLIST = (1000h, +20h)
```

designates a total of 20h consecutive paragraphs, starting at paragraph 1000h (start address = 1000:0000).

```
MEMLIST = ALL
```

gives the contents of all installed co-processor adapter memory.

```
MEMLIST = (1000h, 1FFFh) (3000h, 3FFFh)
```

gives the memory contents of memory addresses 1000:000 through 1FFF:000F and addresses 3000:0000 through 3FFF:000F.

```
MEMLIST = (1000h, +1000h) (3000h, +1000h)
```

gives the same results as the previous example.

Note: In this example, memory addresses 2000:0000 through 2FFF:0 00F are not generated.

The default value for MEMLIST is **NONE**. The keywords ALL or NONE override parameters to the left of them. If conflicting keywords are found, the last (right-most) keyword overrides the others.

PAGE_LINES

The number of lines per page

If a formfeed code is not available on the target printer, this parameter allows the Dump Formatter utility to compute how many blank lines to generate.

This parameter is set by adding the following line to the profile:

```
PAGE_LINES = nn
```

The default value is 66 lines per page.

POSTSTRING

Printer postfix sequence

This sequence comes last in the **MEMORY*n*.PRT** and **SYSINFO*n*.PRT** files. Use it to return the printer to a desired state (for example, 80 character-per-line mode and 8 lines per inch).

This parameter is set by adding the following line to the profile:

```
POSTSTRING = NONE | [nn[,]...]
```

Specifying **NONE** results in no string being generated at the end of **MEMORY*n*.PRT** or **SYSINFO*n*.PRT**.

The length of this string cannot exceed 256 bytes. If more than 256 integers are entered, or if any integer exceeds a value of 255, the default sequence of POSTSTRING is used.

The default sequence is as follows:

```
POSTSTRING = 12h
```

This character sequence stops compressed character mode.

PRESTRING

Printer prefix sequence

This is the sequence that comes first in the **MEMORY*n*.PRT** and **SYSINFO*n*.PRT** files. It is used to force the printer into a desired state.

Set this parameter by adding the following line to the profile:

```
RESTRING = NONE | [ nn [,] ...]
```

Specifying **NONE** results in no string being presented at the start of **MEMORY*n*.PRT** or **SYSINFO*n*.PRT**. The file then begins with formatted output, instead of printer-specific information.

The length of this string cannot exceed 256 bytes; that is, no more than 256 integers can be presented on the line in **frmtdump.pro**. If more than 256 integers are presented, or if any integer exceeds a value of 255, the default sequence of PRESTRING is used.

The Dump Formatter utility defaults to the following:

```
PRESTRING = 18h, 0Fh, 1Bh, 41h, 0Ch, 1Bh, 32h,
            1Bh, 36h, 1Bh, 39h, 1Bh, 43h, 42h, 1Bh,
            46h, 1Bh, 48h, 1Bh, 54h, 1Bh, 55h, 0
```

This character sequence performs the following in this order:

1. Clears the printer buffer

2. Shifts the printer to *compressed character* mode (132 characters per line)
3. Sets line spacing to six lines per inch
4. Selects character set 2
5. Cancels any *ignore paper end* command
6. Sets the page length to 66 lines per page
7. Turns off printing in *emphasized* mode
8. Turns off printing in *double strike* mode
9. Turns off printing in *superscript* mode or *subscript* mode
10. Sets the printer for bidirectional printing.

Printer codes are explained in the documentation for your printer.

PRINT_LINES

The number of lines to print per page

This parameter allows you to set the actual number of lines you want printed on a page. This parameter applies to **SYSINFO*n*.PRT** and **MEMORY*n*.PRT**. It cannot exceed the value of the PAGE_LINES parameter.

Set this parameter by adding the following line to the profile:

```
PRINT_LINES = nn
```

The default value is 60 lines per page.

REPCHARSET

Representable character set

Use this parameter to control characters.

Characters are entered as a series of ASCII character code ranges or individual ASCII character codes. ASCII character codes are represented as two integers separated by a comma or a space, inside parentheses. All the characters within the range, including the lower and upper bounds, are added to the representable character set. Integers represent individual ASCII character codes.

Set this parameter by adding the following line to the profile:

```
REPCHARSET = [(nn[,]nn) | nn[,]...]
```

For example, a representable character set of 30 and the range 32 through 255 might be entered as follows:

```
REPCHARSET = 30,(32,255)
```

TASKLIST

Task memory dumped

This parameter affects only the contents of **MEMORY*n*.PRT**. It specifies a list of the tasks to be included in the output. If a task is included in the list, all of its memory is stored in formatted form in **MEMORY*n*.PRT**.

Set this parameter by adding the following line to the profile:

```
TASKLIST = [ ALL ] [NONE ] [[-|+] nn ] [,] ...
```

where *nn* is a task number.

Specifying **ALL** adds all tasks and their associated memory to **MEMORY*n*.PRT**. Specifying **NONE** subtracts all task-related output

from **MEMORY*n*.PRT**. The list of tasks to be included may be modified by specifying task numbers individually. A minus sign (-) in front of a task number removes it from the list of tasks being printed. A plus sign (+) in front of a task number adds it to the list of tasks being printed.

Note: If a task number does not have a "+" or a "-" preceding it, "+" is assumed.

Some examples of the TASKLIST line follow. The first example shows how to display all tasks except task 210.

```
TASKLIST = ALL -210
```

The second example shows how to display tasks 15h and 16h.

```
TASKLIST = 15h 16h
```

The default setting for TASKLIST is **NONE**.

TITLE

Title for formatted output

This parameter assigns a title to the Dump Formatter utility output files. This title is printed at the top of each page in the **MEMORY*n*.PRT** and **SYSINFO*n*.PRT** files.

To set this parameter, add the following line to the profile:

TITLE = title_string

The title string ends with a carriage return; that is, it must fit on a single line.

The default setting for TITLE is the character string "Dump Information".

USER_SEG

User-selected segment

This parameter sets a special field indicating memory addresses with an offset from the beginning of a user-selected segment.

An address falling within the 64KB block of memory starting at this selected segment appears in the form segment:offset. The physical address and page:offset addresses are displayed for memory outside the 64KB block. Memory addresses outside the 64KB block are not displayed in segment:offset format, since different segment values are required to represent these addresses.

To set this parameter, add the following line to the profile:

```
USER_SEG = nn
```

The default for this parameter is segment 0044h, which is the start of the co-processor adapter Interface Block (IB).

Default Profile

The following profile contains default values and is supplied under file name **frmtdump.pro**. The values are assumed if **frmtdump.pro** cannot be found. The profile need not be present for the Dump Formatter utility to work.

```
BOXCHARS = B3H,C4H,DAH,BFH,D9H,C0H,C3H,B4H,C2H,C1H,C5H
FORM_FEED = 0CH
MEMLIST = NONE
PAGE_LINES = 66
POSTSTRING =  12H
PRESTRING = 18H, 0FH, 1BH, 41H, 0CH, 1BH, 32H, 1BH, 36H, 1BH, 39H,
            1BH, 43H, 42H, 1BH, 46H, 1BH, 48H, 1BH, 54H, 1BH, 55H, 0
PRINT_LINES= 60
REPCHARSET =  (32, 255)
TASKLIST =  NONE
TITLE = Untitled
USER_SEG = 44H
```

Note: The preceding printer-specific parameters are for the IBM Proprinter.

Dump Formatter Messages and Return Codes

The messages and a brief explanation of each one displayed by the Dump Formatter utility are listed in “Dump Formatter Utility Information Messages” on page D-8. “Dump Formatter Utility Return Codes” on page C-8 lists the Dump Formatter return codes and a brief explanation of each.

Chapter 8. Display Utility

The Display utility (**icadisp**) provides a menu of co-processor adapter data structures which can be displayed. The Display utility consists of one file:

- An executable file **icadisp.exe** that can be invoked from the command prompt.

The following Display utility file is in (product installation directory)\bin.

icadisp.exe Display utility

The following example uses **icaldric** to reset the co-processor adapter 0 and load RCM onto the co-processor adapter. It then uses **icadisp** to display information about task 0 on co-processor adapter 0.

```
icaldric 0 icaaim.com 0 -reset
icadisp 0 0
```

where

icadisp is the system unit program which displays the information about the co-processor adapter data structures.

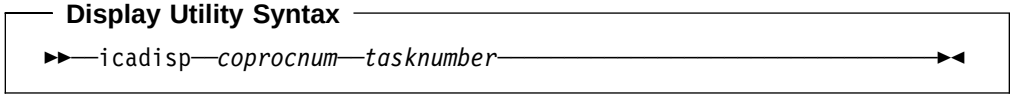
The first **0** is the co-processor adapter number.

The second **0** is the co-processor adapter task number.

Starting the Display Utility

The Display utility command line parameters are optional and can be entered from the menu to indicate which information of the co-processor and task should be displayed. If the parameters are entered on the command line, they must appear in the order in which they are shown. Parameters are separated by white space (spaces and/or tab characters).

Display Utility Syntax



icadisp The name of the display program.

Parameter	Description
coprocnum	The logical co-processor adapter number to be displayed. This parameter is not required, but it must appear first on the command line if the parameter is to be specified.
tasknumber	The task number to be displayed. This parameter is not required, but it must appear second on the command line if the parameter is to be specified.

Examples

In the following example, the co-processor adapter and task number is not specified. All parameters are entered from the menu.

```
icadisp
```

The following example displays the information about co-processor adapter 0.

```
icadisp 0
```

The next example displays information about task 2 on co-processor adapter 1.

```
icadisp 1 2
```

The co-processor adapter data structures are defined in the *Realtime Interface Co-Processor Firmware Technical Reference.*

Chapter 9. C Language Interface Routines

This chapter describes the C Language Interface routines for system unit applications to the ARTIC Windows support device driver and any installed co-processor adapters. Applications linked with the ARTIC Windows NT library (**icaclib.lib**) can issue commands and access the co-processor adapter memory and task parameters through the **icaclib.dll** dynamic link library. The file **icaclib.h** contains declarations for the C language interface routines. Include **icaclib.h** in each of your application programs' source files.

Call Example

Following is an example of a call to the C Language Interface library routine **IcaSecStatBuf**:

```
#include <icaclib.h>    // C Language Interface routine declarations


HANDLE    filehandle;           // Device driver file handle
ULONG     rc;                   // Return code
ICABUFFER ssb;                  // Secondary status buffer
      .
      .
                                // Open access to device driver
filehandle = CreateFile  (ICA186_DRIVERNAME,
                          (GENERIC_READ | GENERIC_WRITE),
                          (FILE_SHARE_READ | FILE_SHARE_WRITE),
                          0,
                          NULL,
                          OPEN_EXISTING,
                          FILE_ATTRIBUTE_NORMAL,
                          NULL);

   if ((filehandle == NULL)
   {
      printf("cannot open icaricio");
      .
      .
   }
   else {
      .
      .
                                   // Get the secondary status buffer
                                   // address, length for task 7, card 0
      if ((rc = IcaSecStatBuf(filehandle,0,7,&ssb)) != 0) {
         printf("call to IcaSecStatBuf failed, return code = 0x%x.\n",rc);
      .
      .
      }
      else {
      .
      .
```

Declarations

The following declarations define the function calls and parameter types used by the C Language Interface routines and are prototyped in **icaclib.h**. Return codes are passed back as function values:

IcaDevRegSemaphore
IcaDevRemSemaphore
IcaDevNotify
IcaDevRemNotify
IcaGetBuffers
IcaGetParms
IcaGetPrimStat
IcaGetVer
IcaInBuf
IcaIntDereg
IcaIntReg
IcaIntWait
IcaIssueCmd
IcaOutBuf
IcaSecStatBuf
IcaSeDereg
IcaSeReg
IcaSeWait
IcaReadMem
IcaReset
IcaWriteMem

Note: You can also use the ARTIC AIX C Language Interface routine names.

IcaDevRegSemaphore

Purpose Registers a semaphore with the device driver for notification of a specific task interrupt.

Format

```
ULONG IcaDevRegSemaphore(HANDLE  filehandle,  // File handle
                         UCHAR   cardnum,     // Co-processor adapter number
                         UCHAR   tasknum,     // Task number
                         HANDLE  semhandle);  // Semaphore handle
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

tasknum The task number.

semhandle The semaphore handle to release when a specific task interrupt occurs. The semaphore must be a counting semaphore.

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_INVALID_FD
E_ICA_RESOURCE_SHORTAGE
E_ICA_INVALID_TASK
E_ICA_INVALID_REQUEST
E_ICA_BAD_SEMAPHORE

Remarks The **IcaDevRegSemaphore** routine allows applications to be notified of task interrupts by way of Win32 API wait functions, for example, the **WaitForSingleObject** function. An application must first register with **IcaDevRegSemaphore** before being notified of task interrupts. Each task interrupt results in the registered semaphore to be released. Therefore, applications should ensure their maximum semaphore count is large enough. For the **IcaDevRegSemaphore** routine, the error task (0xFE) is always a valid task number and will not result in the E_ICA_INVALID_TASK return code.

Related Topics IcaDevRemSemaphore

IcaDevRemSemaphore

Purpose Cancels the semaphore registered by the application process to be notified of a specific task interrupt.

Format

```
ULONG IcaDevRemSemaphore(HANDLE filehandle,  // File handle
                         UCHAR  cardnum,     // Co-processor adapter number
                         UCHAR  tasknum,     // Task number
                         HANDLE semhandle);  // Semaphore handle
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

tasknum The task number.

semhandle The semaphore handle to cancel.

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_NOT_REG
E_ICA_INVALID_FD
E_ICA_INVALID_TASK
E_ICA_INVALID_REQUEST
E_ICA_BAD_SEMAPHORE

Remarks The **IcaDevRemSemaphore** routine cancels a previous application request to be notified of a specific task interrupt through a semaphore. Applications should cancel all requests for task interrupt notification prior to terminating. However, the semaphore is always released one time prior to being canceled. This ensures that any outstanding Win32 API wait function for that semaphore completes.

Related Topics IcaDevRegSemaphore

IcaDevNotify

Purpose Registers a semaphore with the device driver for notification of the Realtime Control Microcode's receipt of an Initialize command.

Format

```
ULONG IcaDevNotify(HANDLE  filehandle,  // File handle
                   UCHAR   cardnum,     // Co-proc adapter number
                   UCHAR   ctrlflag,    // Control flag
                   HANDLE  semhandle);  // Semaphore handle
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

ctrlflag Control bits indicating the events for which the application should be registered. One bit is currently defined in the **ctrlflag** parameter: 0x80. Setting this bit indicates that the application should be registered for Initialize commands issued to the Realtime Control Microcode on the co-processor adapter.

semhandle The semaphore handle to release when a special event occurs. The semaphore must be a counting semaphore.

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_INVALID_CONTROL
E_ICA_ALREADY_REG
E_ICA_INVALID_FD
E_ICA_RESOURCE_SHORTAGE
E_ICA_INVALID_REQUEST
E_ICA_BAD_SEMAPHORE

Remarks The **IcaDevNotify** routine allows applications to be notified of Initialize commands issued to the Realtime Control Microcode by way of Win32 API wait functions (for example, the **WaitForSingleObject** function). An application must first register with **IcaDevNotify** before being notified of Initialize commands issued to the Realtime Control Microcode. Each special event occurrence will result in the release of the registered semaphore. Therefore, applications should ensure their maximum semaphore count is large enough.

Related Topics IcaDevRemNotify

IcaDevRemNotify

Purpose Cancels the request that the semaphore registered by the application be notified of the Realtime Control Microcode's receipt of an Initialize command.

Format

```
ULONG IcaDevRemNotify(HANDLE  filehandle,  // File handle
                      UCHAR   cardnum,     // Co-proc adapter number
                      HANDLE  semhandle);  // Semaphore handle
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

semhandle The semaphore handle to cancel.

Returns

NO_ERROR
E_ICA_INVALID_REQUEST
E_ICA_INVALID_COPROC
E_ICA_INVALID_TASK_STATUS
E_ICA_NOT_REG
E_ICA_INVALID_FD
E_ICA_BAD_SEMAPHORE

Remarks The **IcaDevRemNotify** routine cancels a previous application request to be notified of the Realtime Control Microcode's receipt of an Initialize command through a semaphore. Applications should cancel all requests for such notification prior to terminating. However, the semaphore is always released one time prior to being canceled to ensure that any outstanding Win32 API wait function for that semaphore completes.

Related Topics IcaDevNotify

IcaGetBuffers

Purpose Gets the address and length of a task's input buffer, output buffer, and secondary status buffer.

Format

```
ULONG IcaGetBuffers(HANDLE    filehandle, // File handle
                    UCHAR     cardnum,    // Co-proc adapter number
                    UCHAR     tasknum,    // Task number
                    ICABUFFER *ib,        // Input buffer
                    ICABUFFER *ob,        // Output buffer
                    ICABUFFER *ssb);      // Secondary status buffer
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

tasknum The task number.

ib, ob, and ssb The addresses of three structures to receive the address and length of the input structure, the output structure, and the secondary status buffers. Each structure has the following format:

```
typedef struct
{
  USHORT length;         // Length of buffer
  USHORT offset;         // Offset of buffer address
  UCHAR  page;           // Page of buffer address
  UCHAR  align[3];
}ICABUFFER;
```

where the parameters are defined as follows:

ib

ib.length	The length of the task's input buffer.
ib.offset	The page offset of the task's input buffer.
ib.page	The page number of the task's input buffer.

ob

ob.length	The length of the task's output buffer.
ob.offset	The page offset of the task's output buffer.
ob.page	The page number of the task's output buffer.

ssb

ssb.length	The length of the task's secondary status buffer.
ssb.offset	The page offset of the task's secondary status buffer.
ssb.page	The page number of the task's secondary status buffer.

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_INVALID_TASK_STATUS
E_ICA_INVALID_FD
E_ICA_INVALID_TASK
E_ICA_INVALID_REQUEST

Remarks The **IcaGetBuffers** routine returns the address in page:offset format only.

Related Topics None

IcaGetParms

Purpose Obtains configuration parameter information for a co-processor adapter.

Format

```
ULONG IcaGetParms(HANDLE    filehandle,   // File handle
                  UCHAR     cardnum,      // Task number
                  ICAPARMS  *prmbuf);     // Parameter buffer
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

prmbuf The address of a structure to receive the parameter information. The structure has the following format:

```
typedef struct
  {
    ULONG io_addr;       // Address of I/O ports
    ULONG maxtask;       // Maximum task number
    ULONG maxpri;        // Maximum task priorities
    ULONG maxqueue;      // Maximum queues
    ULONG maxtime;       // Maximum timers
    ULONG int_level;     // Adapter interrupt level
    ULONG ssw_size;      // Shared storage window size
    ULONG adapter_type;  // Adapter type
    ULONG adapter_mem;   // Memory installed
  }ICAPARMS;
```

where the parameters are defined as follows:

io_addr The base address of the co-processor adapter's I/O ports.

Note: See Chapter 2, "Registry Description for Windows NT" and Chapter 3, "Registry Description for Windows 98" for more information about the **maxtask, maxpri, maxqueue** and **maxtime** parameters that follow.

maxtask The highest task number that can be loaded on the co-processor adapter.

maxpri The highest value of a task's priority. The highest priority level is 1, whereas the lowest priority level has the maximum value.

maxqueue The highest queue number that can be allocated on the co-processor adapter.

maxtime The highest timer number that can be allocated on the co-processor adapter.

int_level The interrupt level on which the co-processor adapter interacts with the system unit.

ssw_size A code indicating the size of the shared storage window.

The following table indicates what size window each value represents:

Size Code	Window Size (in KB)
0	8
1	16
2	32
3	64

adapter_type The type of co-processor adapter installed. The following values are defined.

Type	Value	Name
PCI	IBM_X25PCI_ADAPTER IBM_8PORT_PCI_ADAPTER IBM_MM2ISA_PCI_ADAPTER	IBM ARTIC X.25 PCI IBM ARTIC186 8-Port PCI IBM ARTIC186 Model II ISA/PCI
ISA	IBM_X25ISA_ADAPTER IBM_DLPISA_ADAPTER IBM_MM2ISA_ADAPTER IBM_M8PISA_ADAPTER	IBM ARTIC X.25 ISA IBM ARTIC Dual Port IBM ARTIC Multiport Model II IBM ARTIC Multiport IBM ARTIC Multiport 8-Port 232 IBM ARTIC186 8-Port Adapter
MCA	IBM_X25MCA_ADAPTER IBM_PMAMCA_ADAPTER IBM_MP2MCA_ADAPTER	IBM ARTIC X.25 MCA IBM ARTIC PortMaster Adapter/A IBM ARTIC Multiport/2

adapter_mem

Size (in bytes) of the co-processor adapter's memory accessible by the system unit. (The actual physical amount of memory mounted on the co-processor adapter might be greater than the value defined here.) The following values are defined.

0x80000
512KB

0xf0000
960KB

0x100000
1 MB

0x200000
2 MB

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_INVALID_FD
E_ICA_INVALID_REQUEST

Remarks

Some of the parameters (MAXTASK, MAXPRI, MAXTIME, and MAXQUEUE) returned by the **IcaGetParms** routine can be defined in the registry described in Chapter 2, "Registry Description for Windows NT" and Chapter 3, "Registry Description for Windows 98." The device driver uses the parameters or defaults when loading the Realtime Control Microcode onto a co-processor adapter.

Related Topics

None

IcaGetPrimStat

Purpose Obtains the primary status byte for a task.

Format

```
ULONG IcaGetPrimStat(HANDLE filehandle,  // File handle
                     UCHAR  cardnum,     // Co-processor adapter number
                     UCHAR  tasknum,     // Task number
                     PUCHAR primstat);   // Primary status byte
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

tasknum The task number.

primstat A pointer on the returned value of the task's primary status byte. This parameter must be a pageable or non-pageable user space virtual address.

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_INVALID_FD
E_ICA_INVALID_REQUEST
E_ICA_INVALID_TASK

Remarks See the *ARTIC Firmware Technical Reference* for the definition of the bits in the primary status byte.

Related Topics None

IcaGetVer

Purpose Gets the release level of this version of the device driver.

Format

```
ULONG IcaGetVer(HANDLE    filehandle,    // File handle
                USHORT    *verno);       // Version/release number
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

verno The major and minor version code of the device driver. The most significant byte is an unsigned character and represents the minor version code (such as 0x02 if version was 1.2). The least significant byte is the major version code and is also meant to be interpreted as an unsigned character.

Returns

NO_ERROR
E_ICA_INVALID_FD

Remarks The **IcaGetVer** routine allows an application to know which version of the device driver is installed.

Related Topics None

IcaInBuf

Purpose Gets the address and length of a task's input buffer.

Format

```
ULONG IcaInBuf(HANDLE    filehandle,   // File handle
               UCHAR     cardnum,      // Co-processor adapter number
               UCHAR     tasknum,      // Task number
               ICABUFFER *ib);         // Input buffer information
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

tasknum The task number.

ib The address of a structure to receive the input buffer's address and length. The structure has the following format:

```
typedef struct
{
  USHORT length;      // Length of buffer
  USHORT offset;      // Offset of buffer address
  UCHAR  page;        // Page of buffer address
  UCHAR  align[3];
}ICABUFFER;
```

where the parameters are defined as follows:

length The input buffer's length.

offset The input buffer's offset (page:offset format).

page The input buffer's page number.

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_INVALID_TASK_STATUS
E_ICA_INVALID_FD
E_ICA_INVALID_REQUEST
E_ICA_INVALID_TASK

Remarks The **IcaInBuf** routine returns the address in page:offset format only.

Related Topics None

IcaIntDereg

Purpose Cancels the request by the application process to be notified of a specific task interrupt.

Format

```
ULONG IcaIntDereg(HANDLE  filehandle,  // File handle
                  UCHAR   cardnum,     // Co-processor adapter number
                  UCHAR   tasknum);    // Task number
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

tasknum The task number.

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_NOT_REG
E_ICA_INVALID_FD
E_ICA_INVALID_TASK
E_ICA_INVALID_REQUEST

Remarks The **IcaIntDereg** routine cancels a previous application request to be notified of a specific task interrupt. Processes should cancel all requests for task interrupt notification prior to terminating. However, if there is any outstanding **IcaIntWait** for that client, the **IcaIntWait** is terminated with the return code *E_ICA_INTR.*

Related Topics IcaIntReg, IcaIntWait

IcaIntReg

Purpose Registers an application process with the device driver for notification of a specific task interrupt.

Format

```
ULONG IcaIntReg(HANDLE  filehandle,  // File handle
                UCHAR   cardnum,     // Co-processor adapter number
                UCHAR   tasknum);    // Task number
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

tasknum The task number.

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_ALREADY_REG
E_ICA_INVALID_FD
E_ICA_RESOURCE_SHORTAGE
E_ICA_INVALID_TASK
E_ICA_INVALID_REQUEST

Remarks The **IcaIntReg** routine allows applications to be notified of task interrupts by way of the **IcaIntWait** routine. An application must first register with **IcaIntReg** before being notified of task interrupts. Note that for the **IcaIntReg** routine, the error task (0xFE) is always a valid task number and will not result in the E_ICA_INVALID_TASK return code.

Related Topics IcaIntDereg, IcaIntWait

IcaIntWait

Purpose Waits until a specific task on a co-processor adapter interrupts the system unit.

Format

```
ULONG IcaIntWait(HANDLE filehandle,    // File handle
                 UCHAR  cardnum,       // Co-processor adapter number
                 UCHAR  tasknum,       // Task number
                 LONG   timeout);      // Timeout
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter

tasknum The task number

timeout The time in milliseconds to wait for a task interrupt. If this parameter is 0, the call returns immediately, indicating whether or not a previous task interrupt has occurred. If no interrupt occurred, the return code is set to E_ICA_TIMEOUT. If a timeout value of -1 is specified, the wait will block indefinitely (or until the Interrupt Wait is deregistered).

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_TIMEOUT
E_ICA_NOT_REG
E_ICA_INVALID_FD
E_ICA_INVALID_TASK
E_ICA_INTR
E_ICA_INVALID_REQUEST
E_ICA_IN_USE

Remarks The **IcaIntWait** routine returns immediately with no error if an application previously registered for the task interrupt and the interrupt occurred prior to this call. If the interrupt did not occur previously, **IcaIntWait** blocks the user mode application until the specified task interrupts or the time specified in the **timeout** parameter expires. If multiple interrupts by the same task occur prior to this call, the application is notified only once. An application must first register with **IcaIntReg** before being notified of task interrupts. Note that for the **IcaIntWait** routine, the error task (0xFE) is always a valid task number and will not result in the E_ICA_INVALID_TASK return code.

Related Topics IcaIntReg

IcaIssueCmd

Purpose Issues a command to a task with an option to copy parameter data from an application buffer to the task's output buffer before issuing the command.

Format

```
ULONG IcaIssueCmd(HANDLE  filehandle, // File handle
                  UCHAR   cardnum,    // Co-processor adapter number
                  UCHAR   tasknum,    // Task number
                  UCHAR   cmdcode,    // Command Code
                  USHORT  length,     // Length of parameter buffer
                  ULONG   timeout,    // Timeout
                  PUCHAR  prmptr);    // Pointer to parameters
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

tasknum The task number.

cmdcode The command code to put in the task's Buffer Control Block (BCB).

length The number of bytes in the parameter block to be copied to the task's output buffer. A value of 0 indicates that nothing should be written to the task's output buffer.

timeout The number of milliseconds to wait for the Realtime Control Microcode to respond to the command.

Note: If a value of 0 is specified, the device driver gives Realtime Control Microcode the minimum amount of time (100 microseconds) to respond to the command. In heavy traffic conditions, this might not be sufficient and could lead to a timeout. Therefore, set a value greater than 0 for this parameter.

prmptr A pointer to the application buffer containing the data to be written to the task's output buffer. This parameter field is ignored if the length parameter is 0. This field must be a pageable or non-pageable user space virtual address.

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_INVALID_TASK_STATUS
E_ICA_TIMEOUT
E_ICA_BAD_PCSELECT
E_ICA_CMD_REJECTED
E_ICA_OB_SIZE
E_ICA_INVALID_FD
E_ICA_INVALID_TASK
E_ICA_INVALID_FORMAT
E_ICA_INVALID_REQUEST
E_ICA_INVALID_SHORTAGE

Remarks The **IcaIssueCmd** routine issues a command to a task. The caller has the option of copying parameter information into the task's output buffer before the command is issued.

The timeout value applies only to the time to wait for the acknowledgement from Realtime Control Microcode after the command is issued to the task. It does not apply to the time to wait for the task's output buffer to be free if it is currently in the BUSY state. If the task's output buffer is in the BUSY state, the command is rejected with the E_ICA_INVALID_TASK_STATUS return code. It is the responsibility of the application to ensure that any parameter data to be copied to the task's output buffer is in Intel x86 format.

Related Topics None

IcaOutBuf

Purpose Gets the address and length of a task's output buffer.

Format

```
ULONG IcaOutBuf(HANDLE    filehandle,   // File handle
                UCHAR     cardnum,      // Co-processor adapter number
                UCHAR     tasknum,      // Task number
                ICABUFFER *ob);         // Output buffer information
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

tasknum The task number.

ob The address of a structure to receive the output buffer's address and length. The structure has the following format:

```
typedef struct
{
  USHORT length;      // Length of buffer
  USHORT offset;      // Offset of buffer address
  UCHAR  page;        // Page of buffer address
  UCHAR  align[3];
}ICABUFFER;
```

where the parameters are defined as follows:

length The output buffer's length.

offset The output buffer's offset (page:offset format).

page The output buffer's page number.

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_INVALID_TASK_STATUS
E_ICA_INVALID_FD
E_ICA_INVALID_TASK
E_ICA_INVALID_REQUEST

Remarks The **IcaOutBuf** routine returns the address in page:offset format only.

Related Topics None

IcaSecStatBuf

Purpose Gets the address and length of a task's secondary status buffer.

Format

```
ULONG IcaSecStatBuf(HANDLE    filehandle, // File handle
                    UCHAR     cardnum,    // Co-proc adapter number
                    UCHAR     tasknum,    // Task number
                    ICABUFFER *ssb);      // Secondary status buffer
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

tasknum The task number.

ssb The address of a structure to receive the secondary status buffer's address and length. The structure has the following format:

```
typedef struct
{
  USHORT length;      // Length of buffer
  USHORT offset;      // Offset of buffer address
  UCHAR  page;        // Page of buffer address
  UCHAR  align[3];
}ICABUFFER;
```

where the parameters are defined as follows:

length The secondary status buffer's length

offset The secondary status buffer's offset (page:offset format).

page The secondary status buffer's page number.

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_INVALID_TASK_STATUS
E_ICA_INVALID_FD
E_ICA_INVALID_TASK
E_ICA_INVALID_REQUEST

Remarks The **IcaSecStatBuf** routine returns the address in page:offset format only.

Related Topics None

IcaSeDereg

Purpose Cancels the request by the application to be notified of the Realtime Control Microcode's receipt of an Initialize command.

Format

```
ULONG IcaSeDereg(HANDLE  filehandle,  // File handle
                 UCHAR   cardnum);    // Co-proc adapter number
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

Returns

NO_ERROR
E_ICA_INVALID_REQUEST
E_ICA_INVALID_COPROC
E_ICA_INVALID_TASK_STATUS
E_ICA_NOT_REG
E_ICA_INVALID_FD

Remarks The **IcaSeDereg** routine cancels a previous application request to be notified of the Realtime Control Microcode's receipt of an Initialize command. Processes should cancel all requests for such notification prior to terminating.

Related Topics IcaSeReg

IcaSeReg

Purpose Registers an application with the device driver for notification of the Realtime Control Microcode's receipt of an Initialize command.

Format

```
ULONG IcaSeReg(HANDLE  filehandle,  // File handle
               UCHAR   cardnum,     // Co-proc adapter number
               UCHAR   ctrlflag);   // Control flag
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

ctrlflag Control bits indicating the events for which the application should be registered. One bit is currently defined in the **ctrlflag** parameter: 0x80. Setting this bit indicates that the application should be registered for Initialize commands issued to the Realtime Control Microcode on the co-processor adapter.

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_INVALID_CONTROL
E_ICA_ALREADY_REG
E_ICA_INVALID_FD
E_ICA_RESOURCE_SHORTAGE
E_ICA_INVALID_REQUEST

Remarks The **IcaSeReg** routine allows applications to be notified of Initialize commands issued to the Realtime Control Microcode by way of the **IcaSeWait** routine. An application must first register with **IcaSeReg** before being notified of Initialize commands issued to the Realtime Control Microcode.

Related Topics IcaSeDereg, IcaSeWait

IcaSeWait

Purpose Waits until the Realtime Control Microcode on a specified co-processor adapter receives an Initialize command.

Format

```
ULONG IcaSeWait(HANDLE   filehandle,    // File handle
                UCHAR    cardnum,       // Co-proc adapter number
                LONG     timeout);      // Timeout
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

timeout The time in milliseconds to wait for the Realtime Control Microcode to receive an Initialize command. If it is 0, the call returns immediately, indicating whether or not the Realtime Control Microcode has previously received an Initialize command. (If no Initialize command was received, the return code is set to *E_ICA_TIMEOUT*). If a timeout value of -1 is specified, the wait blocks indefinitely (or until the Special Event Wait is deregistered).

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_INVALID_TASK_STATUS
E_ICA_TIMEOUT
E_ICA_NOT_REG
E_ICA_INVALID_FD
E_ICA_INTR
E_ICA_INVALID_REQUEST

Remarks The **IcaSeWait** routine returns immediately with no error if an application previously registered with the **IcaSeReg** call, and the Realtime Control Microcode received an Initialize command prior to this call. If the Realtime Control Microcode has not yet received the Initialize command, this function blocks the application until the Initialize command is received by the Realtime Control Microcode or the time specified in the **timeout** parameter has expired.

Related Topics IcaSeReg

IcaReadMem

Purpose Reads from a co-processor adapter's memory into an application buffer.

Format

```
ULONG IcaReadMem(HANDLE filehandle,   // File handle
                 UCHAR  cardnum,      // Co-processor adapter number
                 ULONG  length,       // Length
                 USHORT segpage,      // Segment/Page
                 USHORT offset,       // Offset
                 UCHAR  addr_format,  // Address format
                 PUCHAR buffptr);     // Destination buffer pointer
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

length The number of bytes to read from co-processor adapter memory.

segpage The segment or page of the co-processor adapter memory address. The interpretation of this field is determined by the **addr_format** field.

offset The offset of the co-processor adapter memory address. The interpretation of this field is determined by the **addr_format** field.

addr_format The control field determining the address format.

Value	Address Interpretation
0x00	The **segpage** parameter is a segment in co-processor adapter memory, and the **offset** is an offset within that segment.
0xFF	The **segpage** parameter is a page in co-processor adapter memory, and the **offset** is an offset within that page.
0x01	The **segpage** and **offset** parameters are a 32-bit physical address in co-processor adapter memory. The least significant 16-bits are in the **offset** field.

Note: All addressing formats are converted to page:offset formats by the device driver prior to accessing co-processor adapter memory. Accesses across pages are handled and physical memory boundaries are checked. Any invalid memory access will be indicated without effectively attempting to access the memory. When accessing the upper 1 MB on a 2 MB IBM ARTIC Portmaster Adapter/A page:offset format must be used because the segment:offset format can refer only to addresses in the 0–1 MB range.

buffptr A pointer to the application buffer where the co-processor adapter memory is to be copied. This parameter must be a pageable or non-pageable user space virtual address.

Returns

NO_ERROR
E_ICA_RESOURCE_SHORTAGE
E_ICA_INVALID_COPROC
E_ICA_INVALID_PAGE
E_ICA_INVALID_OFFSET
E_ICA_INVALID_FORMAT
E_ICA_INVALID_FD
E_ICA_INVALID_REQUEST

Remarks The **IcaReadMem** routine reads from co-processor adapter memory into a system unit application buffer. The address in co-processor adapter memory can be specified either as a segment and offset or as a page and offset. It is the responsibility of the application to recognize that any data read from the co-processor adapter memory will be in Intel x86 format.

Related Topics IcaWriteMem

IcaReset

Purpose Issues a hardware reset to the co-processor adapter.

Format

```
ULONG IcaReset(HANDLE   filehandle,  // File handle
               UCHAR    cardnum);    // Co-processor adapter number
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter to reset.

Returns

NO_ERROR
E_ICA_INVALID_REQUEST
E_ICA_INVALID_COPROC
E_ICA_RESET_FAILED
E_ICA_INVALID_FD

Remarks The **IcaReset** routine issues a hardware reset to the co-processor adapter. The Realtime Control Microcode and all other tasks are unloaded and the co-processor adapter performs a power-on self-test (POST).

RCM parameters (MAXTASK, MAXPRI, MAXQUEUE, and MAXTIME) are refreshed during reset with values read from the registry for the adapter being reset.

A reset of the co-processor adapter may also be performed when loading the Realtime Control Microcode to the co-processor adapter by using the **-reset** Application Loader utility parameter. See page 5-4 for additional information on this option.

Related Topics None

IcaWriteMem

Purpose Writes to a co-processor adapter's memory from an application buffer.

Format

```
ULONG IcaWriteMem(HANDLE  filehandle,     // File handle
                  UCHAR   cardnum,        // Co-processor adapter number
                  ULONG   length,         // Length
                  USHORT  segpage,        // Segment/Page
                  USHORT  offset,         // Offset
                  UCHAR   addr_format,    // Address format
                  PUCHAR  buffptr);       // Source buffer pointer
```

Parameters

filehandle The file handle for the device driver returned by a previous call to the **CreateFile** function.

cardnum The logical number of the co-processor adapter.

length The number of bytes to write from co-processor adapter memory.

segpage The segment or page of the co-processor adapter memory address. The interpretation of this field is determined by the **addr_format** field.

offset The offset of the co-processor adapter memory address. The interpretation of this field is determined by the **addr_format** field.

addr_format The control field determining the address format.

Value	Address Interpretation
0x00	The **segpage** parameter is a segment in co-processor adapter memory, and the **offset** is an offset within that segment.
0xFF	The **segpage** parameter is a page in co-processor adapter memory, and the **offset** is an offset within that page.
0x01	The **segpage** and **offset** parameters are a 32-bit physical address in co-processor adapter memory. The least significant 16-bits are in the offset field.

Note: All address formats are converted to page:offset formats by the ARTIC device driver prior to accessing co-processor adapter memory. Accesses across pages are handled and physical memory boundaries are checked. Any invalid memory access will be indicated without effectively attempting to access the memory. When accessing the upper 1 MB on a 2 MB IBM ARTIC Portmaster Adapter/A, page:offset format must be used because the segment: format can refer only to addresses in the 0–1 MB range.

buffptr A pointer to the application buffer that contains the data to be written to co-processor adapter memory.

This parameter must be a pageable or non-pageable user space virtual address.

Returns

NO_ERROR
E_ICA_INVALID_COPROC
E_ICA_INVALID_PAGE
E_ICA_INVALID_OFFSET
E_ICA_INVALID_FORMAT
E_ICA_INVALID_FD
E_ICA_INVALID_REQUEST
E_ICA_RESOURCE_SHORTAGE

Remarks The **IcaWriteMem** routine writes to co-processor adapter memory from a system unit application buffer. The address in adapter memory can be specified either as a segment and offset or as a page and offset.

Related Topics IcaReadMem

Appendix A. Output File Format for Dump Formatter Utility

This appendix shows samples of the output files generated by the Dump Formatter utility described in Chapter 7, “Dump Formatter Utility,” which formats a co-processor adapter’s dump file for viewing and printing.

Memory Image File

Following is a sample of the formatted output file **MEMORY***n*.**PRT** that was produced from the dump of a Multiport/2 adapter with 512KB of memory. Each page of the output listing is preceded by a form feed character.

```
CO-PROCESSOR ADAPTER 0 MEMORY DUMP                                          12:05:01     Fri Feb 21, 1997     PAGE 1
Dump Information

USER-SET |PHY.   |  8K    |                              MEMORY CONTENTS                                |      CHARACTER REPRESENTATION
 SEGMENT |ADDR.  | PAGES  |+00      +04      +08      +0C      |+10      +14      +18      +1C      | 0123456789ABCDEF 0123456789ABCDEF
---------|-------|--------|-------------------------------------|-------------------------------------|---------------------------------
         |000000|00:0000|001BCE78 56312E34 00000000 00000000|00000000 00000000 00000000 00000000| ...xV1.4........ .......... ....
         |000020|00:0020|00000000 00000000 00000000 00000000|00000000 00000000 00000000 00000000| ................ .......... ....
Addresses 00040 through 00070:                  Same As The Line Above
         |000080|00:0080|FF00FF00 7000C800 00000000 00000000|00000000 00000000 00000000 00000000| ....p........... .......... ....
         |0000A0|00:00A0|00000000 00000000 00000000 00000000|00000000 00000000 00000000 00000000| ................ .......... ....
Addresses 000C0 through 000F0:                  Same As The Line Above
         |000100|00:0100|54000700 26000000 54000700 26000000|74000000 0000C000 0A000000 0000C000| T...&...T...&... t.......... ....
         |000120|00:0120|54000700 20000000 54000700 20000000|74000000 0000C000 0A000000 0000C000| T.......T....... t.......... ....
         |000140|00:0140|00000000 00000000 00000000 00000000|00000000 00000000 00000000 00000000| ................ .......... ....
Addresses 00160 through 00170:                  Same As The Line Above
         |000180|00:0180|9C007000 30004000 5600FF00 FA00F000|00000000 00000000 00002A00 2A00FB00| ..p.0.@.V....... ..........* *...
         |0001A0|00:01A0|06003000 00000000 00000000 FF00FF00|FF00FF00 FF00FF00 05000500 2500FF00| ..0............. .......... %...
         |0001C0|00:01C0|06000000 FF003B00 00000000 00000000|06000000 FF003B00 00000000 00000000| ......;......... ......;.... ....
         |0001E0|00:01E0|00000000 00000000 00000000 00000000|00000000 00000000 00000000 00000000| ................ .......... ....
         |000200|00:0200|C1000000 00000000 00000000 00000000|00000000 00000000 00000000 00000000| ................ .......... ....
         |000220|00:0220|00000000 00000000 00000000 00000000|00000000 00000000 00000000 00000000| ................ .......... ....
Addresses 00240 through 00270:                  Same As The Line Above
         |000280|00:0280|C1000000 00000000 00000000 00000000|00000000 00000000 00000000 00000000| ................ .......... ....
         |0002A0|00:02A0|60000000 0A00F000 00005000 21000400|0000FF00 07000000 C0000000 00000000| ..........P.!... .......... ....
         |0002C0|00:02C0|00000000 00000000 00000000 00000000|00000000 00000000 00000000 00000000| ................ .......... ....
         |0002E0|00:02E0|00000000 00000000 00000000 00FC0000|00000000 00000000 00000000 00000000| ................ .......... ....
         |000300|00:0300|FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF|FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF| ................ .......... ....
         |000320|00:0320|3000FF80 12801280 00000700 00000100|02800100 03000300 02003000 07000700| 0............... ..........0 ....
         |000340|00:0340|FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF|00000200 00002820 00000200 000029A0| ................ ......(.... ..).
         |000360|00:0360|0A060024 FFFF2180 00000000 00000000|FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF| ...$..!......... .......... ....
         |000380|00:0380|FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF|FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF| ................ .......... ....
         |0003A0|00:03A0|38F0F83F 7900F841 B8C0FFFF FFFFFFFF|FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF| ...?y........... .......... ....
         |0003C0|00:03C0|0000F0FF 0000F0FF 00000000 00000000|0000F0FF 0000F0FF 00000000 00000000| ................ .......... ....
         |0003E0|00:03E0|FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF|FFFFFFFF FFFFFFFF FFFFFFFF FFFFFF20| ................ .......... ....
         |000400|00:0400|440707FC 862000B0 448007C1 86AB0002|30000000 0000C047 0A1D0074 0000C000| D.... ..D....... 0......G...t....
         |000420|00:0420|44230723 802300FC 44050700 8000007A|3007007D 0000C000 0A040008 0084C000| D#.#.#..D......z 0..}....... ....
         |000440|00:0440|FFFF0700 20202020 C500CE78 8703CE78|90005B02 00000F00 00009D04 6200CE78| ...........x...x ..[........ b..x
         |000460|00:0460|5505CE78 C903CE78 00000000 00000000|00000000 00000000 0080C1C1 03000303| U..x...x........ .......... ....
                                          .
                                          .
                                          .

         |07F660|3F:1660|EB098B16 264D8A87 0A4DEEF8 C38AD88B|162A4DEC 3CC07405 B402EB17 9032FFD1| ....&M...M...... .*M.<.t.... .2..
         |07F680|3F:1680|E38B1628 4DED2287 224D7505 B401EB03|90B400B0 198866FF F9C3B018 F9C30000| ...(M."."Mu..... ......f.... ....
         |07F6A0|3F:16A0|00000000 00000000 00000000 00000000|00000000 00000000 00000000 00000000| ................ .......... ....
Addresses 7F6C0 through 7FFB0:                  Same As The Line Above
         |07FFC0|3F:1FC0|00000000 00000000 00000000 00000000|00000000 00000000 00000000 9E13E707| ................ .......... ....
         |07FFE0|3F:1FE0|00000000 87032500 E2560000 58FF0000|00000000 CE788F06 917A46F2 00000000| ......%..V..X... .....x...zF ....
```

System Information File

Following is a sample of the formatted output file **SYSINFO*n*.PRT** that was produced from the dump of an IBM ARTIC Multiport/2 co-processor adapter with 512KB of memory. The co-processor adapter had four RS-232 ports and four RS-422 ports and had Realtime Control Microcode (**icaaim.com**) and Realtime Interface Co-Processor Extended Services (**ricps.com** and **riccs.com**) loaded in the following configuration:

Task name	**Task number**
icaaim.com	0
ricps.com	2
riccs.com	3

Each page of the output listing is preceded by a form feed character.

```
CO-PROCESSOR ADAPTER 0 SYSTEM INFORMATION DUMP                          12:09:25    Fri Feb 21, 1997    PAGE 1
Dump Information
```

GENERAL INFORMATION			
Dump Date	02/22/91	Dump Time	12:02:28
Co-Proc. Logical #	0	RAM Size	512K
Windows NT	4.0	ROS/ROM Version	01.4
Dump Version	1.00	Formatter Version	1.00
I/O Base Address	02A0		

```
CO-PROCESSOR ADAPTER 0 SYSTEM INFORMATION DUMP                          12:09:25    Fri Feb 21, 1997    PAGE 2
Dump Information
```

CO-PROCESSOR ADAPTER 80186 CPU REGISTER VALUES	
AX=0000 BX=0000 CS=FC00 SS=0000 DS=0000 ES=0000 BP=03C9 CX=0000 DX=0280 IP=151E SP=02F0 SI=005B DI=0008 CS:IP=FD51E SS:SP=002F0 SS:BP=003C9 DS:SI=0005B ES:DI=00008	FLAGS=F046 OF DF IF SF ZF AF PF CF 0 0 0 0 1 0 1 0

CO-PROCESSOR ADAPTER 80186 PERIPHERAL CONTROL BLOCK (BASE I/O ADDRESS = FF00H)					
OFFSET INTO PCB	REGISTER NAME	VALUE	OFFSET INTO PCB	REGISTER NAME	VALUE
50	Timer 0 Count.....	0000	A8	MPCS.....	C0B8
52	Timer 0 Max Count A	0002	C0	DMA Channel 0 Source Pointer	0000
54	Timer 0 Max Count B.....	0000	C2	DMA Channel 0 Srce Pointer (upper 4 bits)..	FFF0
56	Timer 0 Mode/Control Word	2028	C4	DMA Channel 0 Destination Pointer	0000
58	Timer 1 Count.....	0001	C6	DMA Channel 0 Dest. Pointer (upper 4 bits).	FFF0
5A	Timer 1 Max Count A	0002	C8	DMA Channel 0 Transfer Count	0000
5C	Timer 1 Max Count B.....	0000	CA	DMA Channel 0 Control Word.....	0000
5E	Timer 1 Mode/Control Word	A029	D0	DMA Channel 1 Source Pointer	0000
60	Timer 2 Count.....	0811	D2	DMA Channel 1 Source Pointer (upper 4 bits)	FFF0
62	Timer 2 Max Count A	2400	D4	DMA Channel 1 Destination Pointer	0000
66	Timer 2 Mode/Control Word.....	8021	D6	DMA Channel 1 Dest. Pointer (upper 4 bits).	FFF0
A0	UMCS	F038	D8	DMA Channel 1 Transfer Count	0000
A2	LMCS.....	3FF8	DA	DMA Channel 1 Control Word.....	0000
A4	PACS	0079	FE	Relocation Register	20FF
A6	MMCS.....	41F8			
INTERRUPT CONTROLLER REGISTERS (MASTER MODE)					
22	EOI.....	80FF	32	Timer Control.....	0001
24	Poll	0000	34	DMA 0 Control	0003
26	Poll Status.....	0000	36	DMA 1 Control.....	0003
28	Mask	0000	38	INT0 Control	0002
2A	Priority Mask.....	0007	3A	INT1 Control.....	0030
2C	In-Service	0001	3C	INT2 Control	0007
2E	Interrupt Request...	0001	3E	INT3 Control.....	0007
30	Interrupt Controller Status	8002			

SPECIAL CO-PROCESSOR ADAPTER 80186 I/O PORTS/REGISTERS					
CONFIGURATION SWITCHES (L1,L2,L4,BN,XT,BW,M1,M2) = 11111111					
PORT/REGISTER NAME	I/O ADDRESS	VALUE	PORT/REGISTER NAME	I/O ADDRESS	VALUE
Initialization Even	0004	0A	Parity 1	000E	04
Initialization Odd	0006	F0	Parity 2	0010	00
NMI Mask	0008	00	Daughter Board 0 ID	0200	C1
NMI Status	000A	50	Daughter Board 1 ID	0280	C1
Parity 0	000C	21	Extended Interface	0086	C8
Window Size	001A	00	Clocking opt. 0 & 1	0880	0000

SPECIAL SYSTEM UNIT I/O PORT/REGISTER VALUES								
NAME	I/O ADDRESS	VALUE	NAME	I/O ADDRESS	VALUE	NAME	I/O ADDRESS	VALUE
Page Location	02A0	60	Data Register	02A3	00	Command Reg.	02A6	10
Meg. Location	02A1	00	Task Register	02A4	FF			
Pointer Reg.	02A2	09	CPU Page Reg.	02A5	07			

DATA REGISTER Values Indexed By POINTER REGISTER					
POINTER	REGISTER ACCESSED	VALUE	POINTER	REGISTER ACCESSED	VALUE
08	Interrupt Level	02	0C	Degate Compare 0	10
09	Interrupt Co-Processor	00	0D	Degate Compare 1	E0
0A	Parity Address Low	00	0E	Degate Compare 2	0F
0B	Parity Address High And Status	00	0F	Gate Array ID (SSTIC)	C0

8030 SCC 00 REGISTER VALUES					
80186 I/O ADDRESS	REGISTER NAME	VALUE	80186 I/O ADDRESS	REGISTER NAME	VALUE
0100	RR0B.....	54	0120	RR0A.....	54
0102	RR1B	07	0122	RR1A	07
0104	RR2B.....	26	0124	RR2A.....	20
0106	RR3B	00	0126	RR3A	00
0110	RR8B.....	74	0130	RR8A.....	74
0114	RR10B	00	0134	RR10A	00
0118	RR12B....	0A	0138	RR12A....	0A
011A	RR13B	00	013A	RR13A	00
011E	RR15B....	C0	013E	RR15A....	C0

8030 SCC 01 REGISTER VALUES					
80186 I/O ADDRESS	REGISTER NAME	VALUE	80186 I/O ADDRESS	REGISTER NAME	VALUE
0400	RR0B.....	44	0420	RR0A.....	44
0402	RR1B	07	0422	RR1A	07
0404	RR2B.....	86	0424	RR2A.....	80
0406	RR3B	00	0426	RR3A	00
0410	RR8B.....	30	0430	RR8A.....	30
0414	RR10B	00	0434	RR10A	00
0418	RR12B....	0A	0438	RR12A....	0A
041A	RR13B	00	043A	RR13A	00
041E	RR15B....	C0	043E	RR15A....	C0

8030 SCC 02 REGISTER VALUES					
80186 I/O ADDRESS	REGISTER NAME	VALUE	80186 I/O ADDRESS	REGISTER NAME	VALUE
0600	RR0B.....	5C	0620	RR0A.....	5C
0602	RR1B	07	0622	RR1A	07
0604	RR2B.....	96	0624	RR2A.....	90
0606	RR3B	00	0626	RR3A	00
0610	RR8B.....	30	0630	RR8A.....	30
0614	RR10B	40	0634	RR10A	00
0618	RR12B....	0A	0638	RR12A....	0A
061A	RR13B	00	063A	RR13A	00
061E	RR15B....	C0	063E	RR15A....	C0

8030 SCC 03 REGISTER VALUES					
80186 I/O ADDRESS	REGISTER NAME	VALUE	80186 I/O ADDRESS	REGISTER NAME	VALUE
0700	RR0B.....	5C	0720	RR0A.....	54
0702	RR1B	07	0722	RR1A	07
0704	RR2B.....	E6	0724	RR2A.....	E0
0706	RR3B	00	0726	RR3A	00
0710	RR8B.....	30	0730	RR8A.....	30
0714	RR10B	40	0734	RR10A	40
0718	RR12B....	0A	0738	RR12A....	0A
071A	RR13B	00	073A	RR13A	00
071E	RR15B....	C0	073E	RR15A....	C0

80186 I/O ADDRESS	8036 CIO 0 PORT/REGISTER NAME	VALUE
0180	Master Interrupt Control.....	9C
0182	Master Configuration Control	70
0184	Port A Interrupt Vector.....	30
0186	Port B Interrupt Vector	40
0188	Counter/Timer Interrupt Vector.....	56
018A	Port C Data Path Polarity	FF
018C	Port C Data Direction.....	FA
018E	Port C Special I/O Control	F0
0190	Port A Command and Status.....	00
0192	Port B Command and Status	00
0194	Counter/Timer 1 Command and Status.	00
0196	Counter/Timer 2 Command and Status	00
0198	Counter/Timer 3 Command and Status.	00
019A	Port A Data	2A
019C	Port B Data.....	2A
019E	Port C Data	FB
01A0	Counter/Timer 1 Current Count MSB..	06
01A2	Counter/Timer 1 Current Count LSB	30
01A4	Counter/Timer 2 Current Count MSB..	00
01A6	Counter/Timer 2 Current Count LSB	00
01A8	Counter/Timer 3 Current Count MSB..	00
01AA	Counter/Timer 3 Current Count LSB	00
01AC	Counter/Timer 1 Time Constant MSB..	FF
01AE	Counter/Timer 1 Time Constant LSB	FF

80186 I/O ADDRESS	8036 CIO 0 PORT/REGISTER NAME	VALUE
01B0	Counter/Timer 2 Time Constant MSB..	FF
01B2	Counter/Timer 2 Time Constant LSB	FF
01B4	Counter/Timer 3 Time Constant MSB..	FF
01B6	Counter/Timer 3 Time Constant LSB	FF
01B8	Counter/Timer 1 Mode Specification.	05
01BA	Counter/Timer 2 Mode Specification	05
01BC	Counter/Timer 3 Mode Specification.	25
01BE	Current Vector	FF
01C0	Port A Mode Specification.....	06
01C2	Port A Handshake Specification	00
01C4	Port A Data Path Polarity.....	FF
01C6	Port A Data Direction	3B
01C8	Port A Special I/O Control.....	00
01CA	Port A Pattern Polarity	00
01CC	Port A Pattern Transition...	00
01CE	Port A Pattern Mask	00
01D0	Port B Mode Specification.....	06
01D2	Port B Handshake Specification	00
01D4	Port B Data Path Polarity.....	FF
01D6	Port B Data Direction	3B
01D8	Port B Special I/O Control.....	00
01DA	Port B Pattern Polarity	00
01DC	Port B Pattern Transition.....	00
01DE	Port B Pattern Mask	00

80186 I/O ADDRESS	8036 CIO 1 PORT/REGISTER NAME	VALUE
0500	Master Interrupt Control.....	84
0502	Master Configuration Control	70
0504	Port A Interrupt Vector.....	F0
0506	Port B Interrupt Vector	F8
0508	Counter/Timer Interrupt Vector.....	F6
050A	Port C Data Path Polarity	F0
050C	Port C Data Direction.....	F0
050E	Port C Special I/O Control	F0
0510	Port A Command and Status.....	00
0512	Port B Command and Status	08
0514	Counter/Timer 1 Command and Status.	00
0516	Counter/Timer 2 Command and Status	00
0518	Counter/Timer 3 Command and Status.	00
051A	Port A Data	FC
051C	Port B Data.....	00
051E	Port C Data	F0
0520	Counter/Timer 1 Current Count MSB..	05
0522	Counter/Timer 1 Current Count LSB	FB
0524	Counter/Timer 2 Current Count MSB..	00
0526	Counter/Timer 2 Current Count LSB	00
0528	Counter/Timer 3 Current Count MSB..	00
052A	Counter/Timer 3 Current Count LSB	00
052C	Counter/Timer 1 Time Constant MSB..	FF
052E	Counter/Timer 1 Time Constant LSB	FF

80186 I/O ADDRESS	8036 CIO 1 PORT/REGISTER NAME	VALUE
0530	Counter/Timer 2 Time Constant MSB..	FF
0532	Counter/Timer 2 Time Constant LSB	FF
0534	Counter/Timer 3 Time Constant MSB..	00
0536	Counter/Timer 3 Time Constant LSB	01
0538	Counter/Timer 1 Mode Specification.	05
053A	Counter/Timer 2 Mode Specification	05
053C	Counter/Timer 3 Mode Specification.	00
053E	Current Vector	FF
0540	Port A Mode Specification.....	06
0542	Port A Handshake Specification	00
0544	Port A Data Path Polarity.....	FF
0546	Port A Data Direction	FF
0548	Port A Special I/O Control.....	00
054A	Port A Pattern Polarity	00
054C	Port A Pattern Transition...	00
054E	Port A Pattern Mask	00
0550	Port B Mode Specification.....	00
0552	Port B Handshake Specification	00
0554	Port B Data Path Polarity.....	FF
0556	Port B Data Direction	C0
0558	Port B Special I/O Control.....	00
055A	Port B Pattern Polarity	00
055C	Port B Pattern Transition.....	00
055E	Port B Pattern Mask	00

CO-PROCESSOR ADAPTER FREE MEMORY LIST							
(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)
(0090, +0170)	(025B, +6CFF)	(, +)	(, +)	(, +)	(, +)	(, +)	(, +)

TASK-RELATED INFORMATION FOR TASK 00

TASK HEADER BEGINS AT 7A91:0000 = 3D:0910 (8K PG) = 7A910 TASK IN MEMORY FROM 7A910 TO 7FFFC

TASK HEADER

Task ID = 0002 | Module Length = 000056EC Bytes

Task Number	Priority	Debug Flag	Extension Offset	Resource Request Block Pointer	Sixth Byte	STACK Seg. Off.	COMMAND VECTOR Seg. Off.	INITIAL ENTRY Seg. Off.	DATA SEGMENT Seg. Off.
00	00	00	0000	0000	00	7A91:56EC 3F:1FFC (8K PG) 7FFFC	7A91:0000 3D:0910 (8K PG) 7A910	7A91:0380 3D:0C90 (8K PG) 7AC90	7A91:0000 3D:0910 (8K PG) 7A910

RESOURCES OWNED BY TASK

SCC/CIO's OWNED: | CPU DMA's OWNED:

CIO Timer's OWNED:

SCC's OWNED:

CIO's OWNED:

RS232's OWNED: | RS422's OWNED:
V.35's OWNED: | EIA-530 OWNED:
X.21's OWNED: | OTHER :

SOFTWARE TIMERS OWNED:

VECTORS OWNED:

QUEUES OWNED:

MEMORY BLOCKS OWNED

(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)
(, +)	(, +)	(, +)	(, +)	(, +)	(, +)	(, +)	(, +)

SEMAPHORES OWNED

HANDLE COUNT	HANDLE COUNT	HANDLE COUNT	HANDLE COUNT	HANDLE COUNT	HANDLE COUNT

NAMING RESOURCES OWNED

DEVICE TYPE	DEVICE NO.	NAME POINTER	NAME	DEVICE TYPE	DEVICE NO.	NAME POINTER	NAME

EXTENDED MEMORY BLOCKS OWNED

HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES
(,)	(,)	(,)	(,)	(,)	(,)	(,)	(,)

TASK-RELATED INFORMATION FOR TASK 02

TASK HEADER BEGINS AT 764C:0000 = 3B:04C0 (8K PG) = 764C0 — TASK IN MEMORY FROM 764C0 TO 78CDF

TASK HEADER

Task ID = 5053			Module Length = 0000281F Bytes			STACK Seg. Off.	COMMAND VECTOR Seg. Off.	INITIAL ENTRY Seg. Off.	DATA SEGMENT Seg. Off.
Task Number	Priority	Debug Flag	Extension Offset	Resource Request Block Pointer	Sixth Byte	764C:281F	764C:07D0	764C:031F	764C:0000
02	02	00	0000	0086	00	3C:0CDF (8K PG)	3B:0C90 (8K PG)	3B:07DF (8K PG)	3B:04C0 (8K PG)
						78CDF	76C90	767DF	764C0

RESOURCES OWNED BY TASK

SCC/CIO's OWNED:	CPU DMA's OWNED:
CIO Timer's OWNED:	
SCC's OWNED:	
CIO's OWNED:	

RS232's OWNED: V.35's OWNED: X.21's OWNED:	RS422's OWNED: EIA-530 OWNED: OTHER :

SOFTWARE TIMERS OWNED:
20

VECTORS OWNED:

QUEUES OWNED:

MEMORY BLOCKS OWNED

(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)
(0200, +005B)	(, +)	(, +)	(, +)	(, +)	(, +)	(, +)	(, +)

SEMAPHORES OWNED

HANDLE	COUNT	HANDLE	COUNT	HANDLE	COUNT	HANDLE	COUNT	HANDLE	COUNT	HANDLE	COUNT

NAMING RESOURCES OWNED

DEVICE TYPE	DEVICE NO.	NAME POINTER	NAME	DEVICE TYPE	DEVICE NO.	NAME POINTER	NAME

EXTENDED MEMORY BLOCKS OWNED

HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES
(,)	(,)	(,)	(,)	(,)	(,)	(,)	(,)

TASK-RELATED INFORMATION FOR TASK 03

TASK HEADER BEGINS AT 6F5A:0000 = 37:15A0 (8K PG) = 6F5A0 — TASK IN MEMORY FROM 6F5A0 TO 764BD

TASK HEADER

Task ID = 0000				Module Length = 00006F1D Bytes		STACK Seg. Off.	COMMAND VECTOR Seg. Off.	INITIAL ENTRY Seg. Off.	DATA SEGMENT Seg. Off.
Task Number	Priority	Debug Flag	Extension Offset	Resource Request Block Pointer	Sixth Byte	6F5A:6F1D	6F5A:3255	6F5A:177C	6F5A:0000
03	03	00	0000	0168	00	3B:04BD (8K PG)	39:07F5 (8K PG)	38:0D1C (8K PG)	37:15A0 (8K PG)
						764BD	727F5	70D1C	6F5A0

RESOURCES OWNED BY TASK

SCC/CIO's OWNED: | CPU DMA's OWNED:

CIO Timer's OWNED:

SCC's OWNED:

CIO's OWNED:

RS232's OWNED: | RS422's OWNED:
V.35's OWNED: | EIA-530 OWNED:
X.21's OWNED: | OTHER :

SOFTWARE TIMERS OWNED:
00

VECTORS OWNED:
75 71

QUEUES OWNED:

MEMORY BLOCKS OWNED

(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)	(START,+SIZE)
(, +)	(, +)	(, +)	(, +)	(, +)	(, +)	(, +)	(, +)

SEMAPHORES OWNED

HANDLE	COUNT	HANDLE	COUNT	HANDLE	COUNT	HANDLE	COUNT	HANDLE	COUNT	HANDLE	COUNT

NAMING RESOURCES OWNED

DEVICE TYPE	DEVICE NO.	NAME POINTER	NAME	DEVICE TYPE	DEVICE NO.	NAME POINTER	NAME

EXTENDED MEMORY BLOCKS OWNED

HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES	HANDLE,PAGES
(,)	(,)	(,)	(,)	(,)	(,)	(,)	(,)

Appendix B. C Language Support and Include Files

This appendix describes C language support, the device driver, and the C Language Interface include files.

The following files are part of the C language support included with the device driver to give user mode applications and kernel mode drivers access:

- **icaioctl.h**

  The include file consists of the following major sections:

  – Parameter structure declarations for each device driver function and a union of all such structures
  – Function code definitions for the device driver
  – Definitions that allow easier access by system unit applications to common data structures used by the Realtime Control Microcode and other co-processor adapter tasks
  – Miscellaneous definitions of commonly used constants

- **icaerror.h**

  The include file consists of error code definitions.

- **icaclib.h**

  The include file consists of the following major sections.

  – Function declarations for C Language Interface Library Routines
  – Data structure definitions for C Language Interface Library Routines
  – Definitions that allow easier access by system unit applications to common data structures used by the Realtime Control Microcode and other co-processor adapter tasks
  – Miscellaneous definitions of commonly used constants

To communicate with the device driver:

- Customers writing a user mode application include only **icaclib.h**. This include file will itself include the provided **icaerror.h** file.
- Customers writing a kernel mode driver must include (in any order) **icaioctl.h** and **icaclib.h**. The **icaclib.h** include file will itself include the provided **icaerror.h** file.

Note: No specific variable needs to be defined when compiling modules using the C language support include files.

Appendix C. Return Codes

This appendix explains the return codes for the device driver routines, the C Language Interface routines, the Application Loader utility, the Online Dump utility, and the Dump Formatter utility.

Device Driver Return Codes

The following messages may be returned for the device driver functions and the C Language Interface routines, unless noted otherwise:

0x0000 NO_ERROR

Explanation: The requested driver function was completed successfully.

0xE006EE01L E_ICA_INVALID_COPROC

Explanation: An invalid co-processor adapter number is specified.

0xE006EE02L E_ICA_INVALID_TASK_STATUS

Explanation: The task is not in a correct state for this operation.

0xE006EE03L E_ICA_INVALID_PAGE

Explanation: An invalid page is about to be accessed.

0xE006EE04L E_ICA_INVALID_OFFSET

Explanation: An invalid offset is about to be accessed (I/O or memory).

0xE006EE05L E_ICA_INVALID_FORMAT

Explanation: A memory or I/O address is specified using a wrong format.

0xE006EE06L E_ICA_TIMEOUT

Explanation: The operation has timed out.

0xE006EE07L E_ICA_INVALID_CONTROL

Explanation: An invalid special control flag was specified.

0xE006EE08L E_ICA_BAD_PCSELECT

Explanation: The command could not be issued because the PC Select resource is busy.

0xE006EE09L E_ICA_CMD_REJECTED

Explanation: The command has been rejected by RCM.

0xE006EE0AL E_ICA_OB_SIZE

Explanation: The task's parameter string exceeds the tasks's output buffer size.

0xE006EE0BL E_ICA_ALREADY_REG

Explanation: The registration has already been made for this type of operation.

0xE006EE0CL E_ICA_NOT_REG

Explanation: The registration has not been made prior to issuing a wait operation.

0xE006EE0EL E_ICA_INVALID_TASK

Explanation: An invalid task number is specified.

0xE006EE0FL E_ICA_INTR

Explanation: The operation has been interrupted before completion.

0xE006EE10L E_ICA_BAD_OPEN_HANDLE

Explanation: A system-space access to the device driver is performed using an invalid file object handle.

0xE006EE11L E_ICA_SYSTEM_ERROR

Explanation: An internal device driver error has been detected.

0xE006EE12L E_ICA_RESET_FAILED

Explanation: The reset operation has timed out; reset failed.

0xE006EE13L E_ICA_IN_USE

Explanation: The resource is already in use and cannot be accessed.

0xE006EE14L E_ICA_INVALID_REQUEST

Explanation: The IOCTL request is incorrectly formatted and does not conform to specification.

0xE006EE15L E_ICA_RESOURCE_SHORTAGE

Explanation: The system experiences a shortage in mandatory resources to process the IOCTL request.

0xE006EE16L E_ICA_BAD_SEMAPHORE

Explanation: Invalid semaphore handle specified.

Application Loader Utility Return Codes

The following are the Application Loader utility return codes that correspond to the Application Loader utility messages listed in “Application Loader Utility Information Messages” on page D-1.

00 Normal Termination.

Explanation: The Application Loader utility loaded the requested task. No errors were found.

06 Error opening task, driver, or message file.

Explanation: The Application Loader utility was unable to open the task file or the Application Loader Message file.

07 Error reading task or messages file.

Explanation: The Application Loader utility was unable to read a task file or the loader message file.

08 Error closing task or messages file.

Explanation: The Application Loader utility was unable to close a task file or the Application Loader utility message file.

09 Illegal flag.

Explanation: An illegal flag was entered on the loader command line.

10 Invalid task number.

Explanation: The specified task number is greater than the **MAXTASK** value specified in the parameter file (or the default value if a parameter file is not used).

12 Task 0 already loaded.[1]

Explanation: The Realtime Control Microcode is already loaded on the specified co-processor adapter.

13 Task 0 status invalid.

Explanation: The error bit in the Realtime Control Microcode’s primary status byte is set.

14 Task 0 not loaded and initialized.

Explanation: An attempt was made to load a task before the Realtime Control Microcode was loaded on the specified co-processor adapter.

15 Task already loaded.

Explanation: The specified task is already loaded.

17 Task 0 output buffer size invalid.

Explanation: The output buffer length field in the Realtime Control Microcode’s Buffer Control Block has been overwritten and is invalid.

[1] This message can be returned only when Realtime Control Microcode is being loaded.

18 Command not accepted.

Explanation: The Realtime Control Microcode has rejected a command because its RAM-resident code and/or data has been inadvertently modified.

19 Cannot start task - task not loaded.

Explanation: The specified task was not correctly assigned the "loaded" state in its primary status byte following the Load Task command. This indicates that the task's RAM-resident code and/or data have been modified.

20 File relocation error.

Explanation: An error occurred while the Application Loader utility attempted to relocate a task on the co-processor adapter.

21 No device response.

Explanation: The Application Loader utility did not receive an interrupt from the co-processor adapter in the allocated time. This situation could be caused by a software error on the system unit or by a co-processor adapter hardware error.

22 Invalid PC Select Byte.

Explanation: The command could not be issued because the PC select byte in the task interface block was invalid. This signals that there was an error in some previous communication between the system unit and the co-processor adapter, or this area of storage was inadvertently overwritten.

23 ARTIC device driver is not installed.

Explanation: The device driver is not installed so the Application Loader utility cannot execute.

25 Invalid co-processor adapter number.

Explanation: The specified adapter was not initialized by the device driver and is not recognized as being installed.

26 The device driver returned an error code.

Explanation: A general message for error codes returned to the Application Loader utility by the device driver that are not addressed by any of the other Application Loader utility error messages.

27 Invalid or missing command line argument(s):

Explanation: The command line was found to be in error.

28 E_USAGE_TEXT1

Explanation: Usage text

29 E_USAGE_TEXT2

Explanation: Usage text.

30 E_USAGE_TEXT3

Explanation: Usage text.

31 Driver ioctl error on specified adapter.

Explanation: A device driver IOCTL failed on the co-processor adapter.

32 E_PSB_SSB

Explanation: Primary Status, Secondary Status message.

33 E_CANT_LOAD

Explanation: Cannot download task.

Online Dump Utility Return Codes

The following are the Online Dump utility return codes and correspond to the Online Dump utility messages in “Online Dump Utility Information Messages” on page D-4.

0 No error.

Explanation: The dump completed successfully.

01 Invalid co-processor adapter number.

Explanation: The co-processor adapter was not initialized by the device driver and is not recognized as being installed.

02 Co-processor already enabled for AutoDump.

Explanation: An attempt was made to enable AutoDump on a co-processor adapter that has already been enabled for AutoDump.

03 Co-processor not enabled for AutoDump.

Explanation: An attempt was made to disable AutoDump on a co-processor adapter that has not been enabled for AutoDump.

04 Illegal flag.

Explanation: An illegal flag was entered on the command line.

05 Cannot access directory.

Explanation: The directory (specified with the -dd flag) does not exist or cannot be accessed.

06 Unable to perform dump. Coproc not responding.

Explanation: The co-processor adapter is not responding to commands from the Online Dump utility.

07 AutoDump not enabled on coproc. Coproc not responding.

Explanation: The co-processor adapter is not responding to commands from the Online Dump utility.

08 Dump data will not fit on file system. Dump of coproc canceled.

Explanation: The Online Dump utility detected that the file system where the dump data is to be stored does not have enough free space.

09 AutoDump data will not fit on file system. AutoDump of coproc canceled.

Explanation: The Online Dump utility detected that the file system where the dump data is to be stored does not have enough free space.

10 Device driver is not installed.

Explanation: The device driver is not installed, so the Online Dump utility cannot execute.

11 Error opening a file.

Explanation: The Online Dump utility encountered an error while attempting to open a file.

12 Error reading a file.

Explanation: The Online Dump utility encountered an error while attempting to read a file.

13 Error closing a file.

Explanation: The Online Dump utility encountered an error while attempting to close a file.

14 Cannot create AutoDump tag file.

Explanation: The Online Dump utility could not create the AutoDump tag file.

15 Cannot allocate memory for dump.

Explanation: The Online Dump utility could not allocate enough memory for the dump.

16 Error writing to system or memory dump files.

Explanation: The Online Dump utility could not write to the System or Memory dump files.

17 Device Driver error.

Explanation: The Online Dump utility encountered an error during a device driver call.

Dump Formatter Utility Return Codes

The following codes are returned by the Dump Formatter utility and correspond to the error messages in “Online Dump Utility Error Messages” on page D-5.

00 No errors.

Explanation: The Dump Formatter utility completed successfully.

01 Invalid co-processor adapter specified

Explanation: The co-processor adapter number does not fit in the range 0–15 of valid co-processor adapter numbers.

02 Cannot access file *xxxxxxxx.xxx*

Explanation: The file *xxxxxxxx.xxx* could not be accessed by the Dump Formatter utility, or the dump files **ICAME***n***.DMP** and/or **ICASYS***n***.DMP** could not be opened by the Dump Formatter utility.

03 Illegal command option(s)

Explanation: An illegal option was entered on the command line.

Appendix D. Messages

The messages in this appendix are returned by the Application Loader utility, the Online Dump utility, and the Dump Formatter utility.

Application Loader Utility Information Messages

Normal Termination. Task *yy* loaded on coproc *xx*.

Explanation: The Application Loader utility loaded a task onto co-processor adapter *xx* as task number *yy*. No errors were found.

Programmer Response: None

Application Loader Utility Error Messages

ICALDR06E Error opening *filename*. Return code = *nnnn*.

Explanation: The Application Loader utility was unable to open *filename* which is a task file, the loader message file, or the device entry **/dev/artic**. *nnnn* is the status returned by Windows.

Programmer Response: Check that the file name is correctly spelled.

ICALDR07E Error reading *filename*. Return code = *nnnn*.

Explanation: The Application Loader utility was unable to read *filename* which is a task file or the loader message file. *nnnn* is the status returned by Windows.

Programmer Response: Check that you have read file permission.

ICALDR08E Error closing *filename*. Return code = *nnnn*.

Explanation: The Application Loader utility was unable to close *filename* which is a task file or the loader message file. *nnnn* is the status returned by Windows.

Programmer Response: Retry the command.

ICALDR09E Illegal flag "-xxx".

Explanation: An illegal flag (*-xxx*) was entered on the Application Loader utility command line.

Programmer Response: Correct the error and retry the command.

ICALDR10E Invalid task number. Task number = *nn*.

Explanation: The specified task number, *nn*, is greater than the **maxtask** value specified in the parameter file (or the default value if a parameter file is not used).

Programmer Response: Correct the task number and retry the command.

ICALDR12E Task 0 already loaded. Status = *nn.*

Explanation: The Realtime Control Microcode is already loaded on the specified co-processor adapter. Task 0's primary status byte is *nn*.

Programmer Response: If the Realtime Control Microcode appears to be functioning properly, do not reload the Realtime Control Microcode. If a reload of the Realtime Control Microcode is required, use the **-reset** Application Loader utility option.

ICALDR13E Task 0 invalid status. Status = *nn.*

Explanation: The error bit in the Realtime Control Microcode's primary status byte is set. Task 0's primary status byte is *nn*.

Programmer Response: If a dump has not occurred, you can use the Online Dump utility to dump the co-processor adapter to attempt to locate the cause of the error. Reload the Realtime Control Microcode on the failed co-processor adapter using the **-reset** Application Loader utility option.

ICALDR14E Task 0 not loaded and initialized. Status = *nn***.**

Explanation: An attempt was made to load a task before the Realtime Control Microcode was loaded on the specified co-processor adapter. Task 0's primary status byte is *nn*.

Programmer Response: Load the Realtime Control Microcode.

ICALDR15E Task already loaded. Status = *nn***.**

Explanation: The specified task is already loaded. The previously loaded task's primary status byte is *nn*.

Programmer Response: Ensure that the task number is correct.

ICALDR17E Task 0 output buffer size invalid. Status = *nn***.**

Explanation: The output buffer length field in the Realtime Control Microcode's Buffer Control Block has been overwritten and is invalid. Task 0's primary status byte is *nn*.

Programmer Response: You may use the Online Dump utility to dump the co-processor adapter to attempt to find the cause of the error. Reload the Realtime Control Microcode using the **-reset** Application Loader utility option.

ICALDR18E Command not accepted. Status = *nn***.**

Explanation: The Realtime Control Microcode has rejected a command because its RAM-resident code and/or data have been inadvertently modified. Task 0's primary status byte is *nn*.

Programmer Response: Use the Online Dump utility to dump the co-processor adapter. Reload the Realtime Control Microcode using the **-reset** Application Loader utility option.

ICALDR19E Cannot start task - task not loaded.

Explanation: The specified task was not correctly assigned the "loaded" state in its primary status byte following the Load Task command. The task's RAM-resident code and/or data have been modified.

Programmer Response: Use the Online Dump utility to dump the adapter. Reload the Realtime Control Microcode using the **-reset** Application Loader utility option. Retry the command to load the task.

ICALDR20E File relocation error.

Explanation: An error occurred while the Application Loader utility attempted to relocate a task on the co-processor adapter.

Programmer Response: Verify that the task file is of proper format, either **.com** or **.exe**.

ICALDR21E No device response. Co-processor = *nn*. Status = *nn*.

Explanation: The Application Loader utility did not receive an interrupt from the co-processor adapter *nn* in the allocated time. This situation could be caused by a software error on the system unit or by a co-processor adapter hardware error.

Programmer Response: Run diagnostics on the failing adapter.

ICALDR22E Invalid PC Select Byte. PC Select = *nn*.

Explanation: The command could not be issued because the PC Select Byte in the task interface block was invalid. This signals that there was an error in some previous communication between the system unit and the co-processor adapter, or this area of storage was inadvertently overwritten.

Programmer Response: First, isolate the error. To recover from the error, issue a hardware reset to the co-processor adapter to clear the condition.

ICALDR23E ARTIC device driver is not installed.

Explanation: The Application Loader utility cannot execute because the device driver is not installed. The device entry **/dev/artic** does not exist.

Programmer Response: Check the event viewer log to determine if the driver is installed. Install the device driver if required.

ICALDR25E Invalid co-processor adapter number:*nnnn*.

Explanation: The co-processor adapter number *nnnn* was not initialized by the device driver and is not recognized as being installed.

Programmer Response: Verify that the adapter number is correct. Check the event viewer log to determine if the driver is installed. Install the device driver if required.

ICALDR26E The device driver returned error code *nn*.

Explanation: A general message for error codes returned to the Application Loader utility by the device driver which are not addressed by any of the other loader error messages. The error code returned by the device driver is *nn*.

Programmer Response: Check “Application Loader Utility Return Codes” on page C-3 for the source of the problem.

ICALDR27E Invalid or missing command line argument(s):

Explanation: The startup command line was found to be in error. The correct usage format is displayed.

Usage: icaldric coprocnum filename tasknumber ([-ns] [-l]) [-m boundary] [-q] [-prm string] [-reset]

Programmer Response: Correct the command and retry.

ICALDR31E Device driver ioctl error on “devname”. Error code = *nnnn*.

Explanation: A device driver function failed on the device *devname*.

Programmer Response: Check “Application Loader Utility Return Codes” on page C-3 for the source of the problem.

ICALDR32E PriStatus: *nn*. SecStatus: *nn nn nn nn nn nn ...*

Explanation: This message is displayed following error messages ICALDR19E, ICALDR21E, and ICALDR33E—response, cannot start task, and task not initialized. The primary and secondary status bytes are those of Task 0.

Programmer Response: Decode the status bytes to aid in problem determination.

Online Dump Utility Information Messages

ICADPR30I Dump Completed.

Explanation: The Online Dump utility completed dumping a co-processor adapter's memory and I/O ports to disk.

Programmer Response: The output files can be formatted using the Dump Formatter utility (see Chapter 7, "Dump Formatter Utility" for a description).

ICADPR31I Writing dump data....

Explanation: The Online Dump utility is writing dump data to specified file(s).

Programmer Response: Wait for the Online Dump utility to finish dumping the co-processor adapter's memory and I/O ports.

ICADPR32I AutoDump beginning....

Explanation: A co-processor adapter requested that its memory and I/O ports be dumped to disk when a Level 1 error occurs on the adapter. The write-to-disk is beginning.

Programmer Response: Wait for the Online Dump utility to finish dumping the co-processor adapter's memory and I/O ports. The output files can be formatted using the Dump Formatter utility.

ICADPR34I Use event viewer to obtain the specific device driver information.

Explanation: The device driver had an error and displayed a return code.

Programmer Response: Correct the condition described in the event viewer.

Online Dump Utility Error Messages

(none) **usage: icadpric coprocnum -[d | ea | da] [-dd directory]**

Explanation: Invalid entry on the command line.

Programmer Response: Re-enter correct information using the indicated format.

ICADPR01E **Invalid co-processor adapter number:** *nn***.**

Explanation: Co-Processor adapter number *nn* was not initialized by the device driver and is not recognized as being installed.

Programmer Response: Check the co-processor adapter number to make sure that it matches the co-processor adapter to be dumped.

ICADPR02E **Co-processor** *nn* **already enabled for AutoDump.**

Explanation: An attempt was made to enable AutoDump on a co-processor adapter (*nn*) that has already been enabled for AutoDump.

Programmer Response: Verify that the co-processor adapter is enabled for AutoDump by checking to see if the process identification (PID) number contained in the AutoDump tag file (/tmp/AUTODUMP.n, where n is the co-processor adapter number) is an active **icadpric** process.

Example: If the PID number 6789 is contained in the tag file AUTODUMP.1 (for co-processor adapter 1), then use the "ps 6789" command to check the status of process 6789. If the output of the "ps" command contains "icadpric 1 -ea", then the process is already enabled for AutoDump. If PID number 6789 was not found, remove the tag file and attempt to enable the co-processor adapter again.

ICADPR03E **Co-processor nn not enabled for AutoDump.**

Explanation: An attempt was made to disable AutoDump on a co-processor adapter (*nn*) that has not been enabled for AutoDump.

Programmer Response: Verify that the co-processor adapter is enabled for AutoDump by checking to see if the AutoDump tag file (/tmp/**AUTODUMP**.X, where X is the co-processor adapter number) exists. If it does, the co-processor adapter is enabled for AutoDump. If it does not, it was never enabled.

ICADPR04E **Illegal flag** *-xxx*

Explanation: An illegal flag (*-xxx*) was entered on the command line.

Programmer Response: Start the Online Dump utility by using the indicated format.
usage: icadpric coprocnum -[d | ea | da] [-dd directory].

ICADPR05E **Cannot access directory** *xxx***.**

Explanation: The directory *xxx* (specified with the *-dd* flag) does not exist or cannot be accessed.

Programmer Response: If the directory does not exist, create it or specify a different path. If the directory cannot be accessed, obtain write permission for that directory or specify a different path.

ICADPR06E **Unable to perform dump. Coproc** *nn* **not responding.**

Explanation: Co-processor adapter *nn* is not responding to commands from the Online Dump utility.

Programmer Response: Make sure that the Realtime Control Microcode is loaded and running on the co-processor adapter.

ICADPR07E AutoDump not enabled on coproc *nn*. Coproc not responding.

Explanation: Co-processor adapter *nn* is not responding to commands from the Online Dump utility.

Programmer Response: Make sure that the Realtime Control Microcode is loaded and running on the co-processor adapter.

ICADPR08E Dump data will not fit on file system. Dump of coproc *nn* canceled.

Explanation: The Online Dump utility detected that the file system where the dump data is to be stored does not have enough free space.

Programmer Response: Make room on the target file system by removing files, or choose a different file system for the dump data.

ICADPR09E AutoDump data will not fit on file system. AutoDump of coproc *nn* canceled.

Explanation: The Online Dump utility detected that the file system where the dump data is to be stored does not have enough free space.

Programmer Response: Make room on the target file system by removing files or choose a different file system for the dump data.

ICADPR10E ARTIC device driver is not installed.

Explanation: The device driver is not installed so the Online Dump utility cannot execute.

Programmer Response: Install the device driver.

ICADPR11E Error opening *xxx*. Return code = *nn*.

Explanation: The Online Dump utility encountered an error while attempting to open file *xxx*. Return code *nn* from the **open** system call is also displayed.

Programmer Response: Verify that file *xxx* exists and has read/write permission.

ICADPR12E Error reading *xxx*. Return code = *nn*.

Explanation: The Online Dump utility encountered an error while attempting to read file *xxx*. Return code *nn* from the **read** system call is also displayed.

Programmer Response: Verify that file *xxx* exists, has read permission, and contains data to be read.

ICADPR13E Error closing *xxx*. Return code = *nn*.

Explanation: The Online Dump utility encountered an error while attempting to close file *xxx*. Return code *nn* from the **close** system call is also displayed.

Programmer Response: Verify that file *xxx* exists.

ICADPR14E Cannot create AutoDump tag file *xxx*.

Explanation: The Online Dump utility could not create the AutoDump tag file.

Programmer Response: Verify directory **/tmp** has write permission and that the file system where /tmp resides has enough space.

ICADPR15E Cannot allocate memory for dump.

Explanation: The Online Dump utility could not allocate enough memory for the dump.

Programmer Response: Reduce system load and try again.

ICADPR16E Error writing to system or memory dump files.

Explanation: The Online Dump utility could not write to the system or memory dump files.

Programmer Response: Check to make sure that the system or memory dump files were not removed before the dump was completed. Also check to verify that the file system did not run out of space for the dump files.

ICADPR17E Device Driver error, return code = ***nn*****.**

Explanation: The Online Dump utility encountered an error during a device driver call. The return code from the device driver is *nn*.

Programmer Response: Refer to Appendix C, “Return Codes” for the source of the problem.

Dump Formatter Utility Information Messages

Message File missing or invalid.

Explanation: The message file **frmtdump.msg** could not be accessed. The default messages (U.S. English) are being used.

File already exists: *xxxxxxxx.xxx*
Press:
0 To Abort
1 To Overwrite

Explanation: The Dump Formatter utility output file *xxxxxxxx.xxx* already exists. The user is being prompted to decide whether to abort the formatting process or continue and overwrite the old output file.

Programmer Response: Press 0 to Abort or 1 to Overwrite.

Dump Formatter Utility Error Messages

ICAFRM001E Invalid co-processor adapter specified.

Explanation: The co-processor adapter number specified is not between 0 and 15.

ICAFRM002E Error accessing *xxx.*

Explanation: The output directory *xxx* (specified with the **-fd** flag) cannot be found or the Dump Formatter does not have write access to it.

ICAFRM003E Invalid command line option(s).

Explanation: An illegal parameter (*-xxx*) was entered on the command line.

Programmer Response: Correct the error and retry the command.

usage: frmtdump coprocnum -[m | s] [-fd directory]

Appendix E. Notices and Trademarks

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Trademarks and Service Marks

The following terms are trademarks of the IBM Corporation in the United States or other countries, or both:

IBM Micro Channel

Other company, product or service names may be the trademarks or service marks of others.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Printed in U.S.A.