# iRMX®
# Basic I/O System Calls Reference Manual

Order Number: 462915-001

intel®

# iRMX®
# Basic I/O System Calls Reference Manual

Order Number: 462915-001

Intel Corporation
3065 Bowers Avenue
Santa Clara, California 95051

In locations outside the United States, obtain additional copies of Intel documentation by contacting your local Intel sales office. For your convenience, international sales office addresses are located directly after the reader reply card in the back of the manual.

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update or to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

Above
BITBUS
COMMputer
CREDIT
Data Pipeline
Genius
$\overset{\Delta}{\imath}$
i
I$^2$ICE
ICE
iCEL
iCS
iDBP
iDIS

iLBX
i$_m$
iMDDX
iMMX
Insite
int$_e$l
Intel376
Intel386
int$_e$lBOS
Intelevision
int$_e$ligent Identifier
int$_e$ligent Programming
Intellec
Intellink
iOSP
iPDS
iPSB

iPSC
iRMX
iSBC
iSBX
iSDM
iSSB
iSXM
Library Manager
MCS
Megachassis
MICROMAINFRAME
MULTIBUS
MULTICHANNEL
MULTIMODULE
OpenNET
ONCE

Plug-A-Bubble
PROMPT
Promware
QUEST
QueX
Ripplemode
RMX/80
RUPI
Seamless
SLD
UPI
VLSiCEL
376
386
386SX

XENIX, MS-DOS, Multiplan, and Microsoft are trademarks of Microsoft Corporation. UNIX is a trademark of Bell Laboratories. Ethernet is a trademark of Xerox Corporation. Centronics is a trademark of Centronics Data Computer Corporation. Chassis Trak is a trademark of General Devices Company, Inc. VAX and VMS are trademarks of Digital Equipment Corporation. Smartmodem 1200 and Hayes are trademarks of Hayes Microcomputer Products, Inc. IBM, PC/XT, and PC/AT are registered trademarks of International Business Machines. Soft-Scope is a registered trademark of Concurrent Sciences.

| REV. | REVISION HISTORY | DATE |
|---|---|---|
| -001 | Original Issue. | 03/89 |

# PREFACE

This manual documents the system calls of the Basic I/O System, one of the subsystems of the iRMX® I and iRMX II Operating Systems. The information provided in this manual is intended as a reference to the system calls and provides detailed descriptions of each call.

## READER LEVEL

This manual is intended for programmers who are familiar with the concepts and terminology introduced in the *iRMX® I Nucleus User's Guide* or the *iRMX® II Nucleus User's Guide* and with the PL/M programming language.

## CONVENTIONS

System call names appear as headings on the outside upper corner of each page. The first appearance of each system call name is printed in blue ink; subsequent appearances are in black.

Throughout this manual, system calls are shown using a generic shorthand (such as A$CREATE$FILE instead of RQ$A$CREATE$FILE). This convention is used to allow easier alphabetic arrangement of the calls. The actual PL/M external-procedure names must be used in all calling sequences.

You can also invoke the system calls from assembly language, but you must obey the PL/M calling sequences when doing so. For more information on these calling sequences, refer to the *iRMX® I Programming Techniques Reference Manual* or the *iRMX® II Programming Techniques Reference Manual.*

# CONTENTS

## Chapter 1. iRMX® Basic I/O System Calls

1.1 Introduction ........ 1-1
1.2 System Call Command Dictionary ........ 1-3
A$ATTACH$FILE ........ 1-6
A$CHANGE$ACCESS ........ 1-11
A$CLOSE ........ 1-19
A$CREATE$DIRECTORY ........ 1-22
A$CREATE$FILE ........ 1-28
A$DELETE$CONNECTION ........ 1-36
A$DELETE$FILE ........ 1-39
A$GET$CONNECTION$STATUS ........ 1-45
A$GET$DIRECTORY$ENTRY ........ 1-49
A$GET$EXTENSION$DATA ........ 1-52
A$GET$FILE$STATUS ........ 1-55
A$GET$PATH$COMPONENT ........ 1-64
A$OPEN ........ 1-68
A$PHYSICAL$ATTACH$DEVICE ........ 1-72
A$PHYSICAL$DETACH$DEVICE ........ 1-76
A$READ ........ 1-79
A$RENAME$FILE ........ 1-83
A$SEEK ........ 1-89
A$SET$EXTENSION$DATA ........ 1-92
A$SPECIAL ........ 1-95
A$TRUNCATE ........ 1-129
A$UPDATE ........ 1-132
A$WRITE ........ 1-136
CREATE$USER ........ 1-140
DELETE$USER ........ 1-142
ENCRYPT ........ 1-143
GET$DEFAULT$PREFIX ........ 1-145
GET$DEFAULT$USER ........ 1-147
GET$GLOBAL$TIME ........ 1-149
GET$TIME ........ 1-151
INSPECT$USER ........ 1-152
SET$DEFAULT$PREFIX ........ 1-154
SET$DEFAULT$USER ........ 1-156
SET$GLOBAL$TIME ........ 1-158
SET$TIME ........ 1-160
WAIT$IO ........ 1-161

## Appendix A. I/O Request/Result Segment

A.1 Overview ..... A-1
A.2 Structure of I/O Request/Result Segment ..... A-2

## Index

# iRMX® BASIC I/O SYSTEM CALLS 1

## 1.1 INTRODUCTION

The *iRMX® Basic I/O System Calls* manual provides a detailed description of each Basic I/O System call, listed alphabetically.

BIOS system calls can be divided into two categories according to their names. The first category consists of system calls having names of the form

RQ$XXXXX

where XXXXX is a brief description of what the system call does. The second category consists of system calls having names of the form

RQ$A$XXXXX

System calls of the first category, without the A, are synchronous calls. They begin running as soon as your application invokes them, and continue running until they detect an error or accomplish everything they must do. Then they return control to your application.

System calls of the second category (those with the A) are called asynchronous because they accomplish their objectives by using tasks that run concurrently with your application. This allows your application to accomplish some work while the Basic I/O System deals with devices such as disk drives and tape drives.

This manual describes the PL/M calling sequences to the Basic I/O System calls. Basic I/O operations are declared as typed or untyped external procedures for PL/M. PL/M programs perform I/O operations by making external procedure calls.

The information for each system call is organized in this order:

- A brief sketch of the effects of the call.
- The PL/M calling sequence for the system call.
- Definitions of the input parameters, if any.
- Definitions of the output parameters, if any.
- A detailed description of the effects of the call.
- The condition codes that can result from using the call, with a description of the possible causes of each condition.

Throughout this manual, PL/M data types, such as BYTE, WORD, and SELECTOR are used. In addition, the iRMX® data type TOKEN (always capitalized) is used. If your compiler supports the SELECTOR data type, a TOKEN can be declared literally as SELECTOR or WORD. Because TOKEN is not a PL/M data type, you must declare it to be literally a SELECTOR or a WORD every place you use it. An asterisk (*) is used as a STRUCTURE and ARRAY size indicator. You must substitute a value for the asterisk in STRUCTURE and ARRAY declarations.

## NOTE

> The values NIL and SELECTOR$OF(NIL) are used throughout this manual. For the iRMX I Operating System, you may also use a value of zero in place of NIL and SELECTOR$OF(NIL). However, Intel recommends that you use NIL and SELECTOR$OF(NIL) in your iRMX I code to maintain upward compatibility with the iRMX II Operating System. For a description of the SELECTOR$OF and NIL built-in functions, refer to the PL/M-86 or PL/M-286 user's guides.

The Basic I/O System does not distinguish between upper and lowercase letters. For example, file "xyz" is equal to file "XYZ".

The system call dictionary on these next few pages lists system calls by function rather than alphabetically. This dictionary includes short descriptions and page numbers of the complete descriptions that follow.

## 1.2 SYSTEM CALL COMMAND DICTIONARY

This dictionary summarizes the Basic I/O System calls by function and, where applicable, indicates the file types to which they apply:

| | |
|---|---|
| PF | Physical file |
| SF | Stream file |
| NF | Named data file |
| ND | Named directory file |

The page reference listed with each call points to the detailed description for the call.

| JOB-LEVEL SYSTEM CALLS | | | |
|---|---|---|---|
| **Call** | **Description** | | **Page** |
| ENCRYPT | Encodes user password. | | 1-143 |
| GET$DEFAULT$-PREFIX | Inspect default prefix. | | 1-145 |
| GET$DEFAULT$USER | Inspect default user. | | 1-147 |
| SET$DEFAULT$-PREFIX | Set default prefix for job. | | 1-154 |
| SET$DEFAULT$USER | Set default user for job. | | 1-156 |
| **DEVICE-LEVEL SYSTEM CALLS** | | | |
| A$PHYSICAL$-ATTACH$DEVICE | Attach device. | | 1-72 |
| A$PHYSICAL$-DETACH$DEVICE | Detach device. | | 1-76 |
| A$SPECIAL | Perform device-level function. | | 1-95 |
| **FILE/CONNECTION-LEVEL SYSTEM CALLS** | | | |
| **Call** | **Description** | **Files** | **Page** |
| A$ATTACH$FILE | Attach file. | All | 1-6 |
| A$CREATE$-DIRECTORY | Directory file creation. | ND | 1-22 |

| FILE/CONNECTION-LEVEL SYSTEM CALLS (continued) | | | |
|---|---|---|---|
| **Call** | **Description** | **Files** | **Page** |
| A$CREATE$FILE | Data file creation. | PF,SF,NF | 1-28 |
| A$DELETE$CONNECTION | Delete file connection. | All | 1-36 |
| A$DELETE$FILE | Data or directory file deletion. | SF,NF,ND | 1-39 |
| **FILE-MODIFICATION SYSTEM CALLS** | | | |
| A$CHANGE$ACCESS | Change access rights to file. | NF,ND | 1-11 |
| A$RENAME$FILE | Rename file. | NF,ND | 1-83 |
| A$TRUNCATE | Truncate file. | NF | 1-129 |
| **FILE INPUT/OUTPUT SYSTEM CALLS** | | | |
| A$CLOSE | Close file. | All | 1-19 |
| A$OPEN | Open file. | All | 1-68 |
| A$READ | Read file. | All | 1-79 |
| A$SEEK | Move file pointer. | PF,NF,ND | 1-89 |
| A$UPDATE | Finish writing to output device. | PF,NF,ND | 1-132 |
| WAIT$IO | Wait for status after I/O. | All | 1-161 |
| A$WRITE | Write file. | PF,SF,NF | 1-136 |
| **GET STATUS/ATTRIBUTE SYSTEM CALLS** | | | |
| A$GET$CONNECTION$STATUS | Get connection status. | All | 1-45 |
| A$GET$DIRECTORY$ENTRY | Inspect directory entry. | ND | 1-49 |
| A$GET$FILE$STATUS | Get file status. | All | 1-55 |
| A$GET$PATH$COMPONENT | Obtains path name from connection token. | NF,ND | 1-64 |

| USER OBJECT SYSTEM CALLS | | | |
|---|---|---|---|
| **Call** | **Description** | **Files** | **Page** |
| CREATE$USER | Create a user object. | | 1-140 |
| DELETE$USER | Delete a user object. | | 1-142 |
| INSPECT$USER | Get IDs in a user object. | | 1-152 |
| **EXTENSION DATA SYSTEM CALLS** | | | |
| A$GET$EXTENSION$-DATA | Receive a file's extension data. | NF,ND | 1-52 |
| A$SET$EXTENSION$-DATA | Store a file's extension data. | NF,ND | 1-92 |
| **TIME/DATE SYSTEM CALLS** | | | |
| GET$TIME | Get date/time value in internally-stored format. | | 1-151 |
| SET$TIME | Set date/time value in internally-stored format. | | 1-160 |
| **CALLS FOR ACCESSING THE GLOBAL TIME-OF-DAY CLOCK** | | | |
| GET$GLOBAL$TIME | Obtains the time of day from the battery backed-up hardware clock. | | 1-149 |
| SET$GLOBAL$TIME | Sets the battery backed-up hardware clock to a specified time. | | 1-158 |

# A$ATTACH$FILE

A$ATTACH$FILE creates a connection to an existing file.

```
    CALL RQ$A$ATTACH$FILE(user, prefix, subpath$ptr, resp$mbox,
                          except$ptr);
```

## Input Parameters

user
: A TOKEN for the user object to be inspected in any access checking that takes place. A SELECTOR$OF(NIL) specifies the default user for the calling task's job. This parameter is ignored when attaching physical or stream files. Access checking does occur for named files.

prefix
: A TOKEN for the connection object to be used as the path prefix. A SELECTOR$OF(NIL) specifies the default prefix for the calling task's job.

subpath$ptr
: A POINTER to a STRING containing the subpath of the file to be attached. A null string indicates that the new connection is to the file designated by the prefix. The new connection will not be open, regardless of the open mode of the prefix.
(This parameter is ignored for physical and stream files.)

## Output Parameters

resp$mbox
: A TOKEN for the mailbox into which the Basic I/O System places a token for the result object of the call. This result object is a new file connection if the call succeeds or an I/O request/result segment (IORS) otherwise (for details on the IORS, see Appendix A). To ascertain the type of object returned, use the Nucleus system call GET$TYPE.

If the object received is an IORS, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

except$ptr
: A POINTER to a WORD where the sequential condition code will be returned.

## Description

A$ATTACH$FILE creates a connection to an existing file. Once the connection is established, it remains in effect until the connection object is deleted, or until the creating job is deleted. Once attached, the file may be opened, closed, read, written, etc., as many times as desired. A$ATTACH$FILE has no effect on the owner ID or the access list for the file.

## Special Considerations for iRMX®-NET

Unlike a local named file, the access rights of a remote named file are not checked when a connection to the file is created. Instead, the remote named file's access rights are checked during operations on the connection.

The above discrepancy won't affect your programs if you do the following:

- Open, delete, and rename files prior to changing their access lists.
- Establish connections to files after changing their access lists.

## Condition Codes

A$ATTACH$FILE returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide*.

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

### Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$DEV$OFFLINE | 002EH | The prefix parameter in this system call refers to a logical connection. One of the following is true of the device associated with the connection:<br>• It has been physically attached but is now off-line.<br>• It has never been physically attached. (See Appendix E in the *iRMX® Basic I/O User's Guide* for a more detailed explanation.) |

E$EXIST 0006H One of the following is true:

- One or more of the following parameters is not a token for an existing object:
  - The user parameter
  - The prefix parameter
  - The resp$mbox parameter
- The prefix connection is being deleted.
- The connection for a remote driver is no longer active.

E$LIMIT 0004H Processing this call would cause one or more of these limits to be exceeded:

- The object limit for this job.
- The number of I/O operations that can be outstanding at one time for the user object specified in the call (255 decimal).
- The number of I/O operations that can be outstanding at one time for the caller's job (255 decimal).
- The number of outstanding I/O operations for a remote connection has been exceeded.

E$MEM 0002H The memory available to the calling task's job is not sufficient to complete the call.

E$NOPREFIX 8022H The calling task specified a default prefix (prefix argument equals SELECTOR$OF(NIL)), but no default prefix can be found because of one of the following reasons:

- When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default prefix.
- The job's directory can have entries but a default prefix is not cataloged there.

| | | |
|---|---|---|
| E$NOUSER | 8021H | If the user parameter in this call is not SELECTOR$OF(NIL), the parameter is not a token for a user object.<br>If the user parameter is SELECTOR$OF(NIL), it specifies a default user, but no default user can be found because of one of the following reasons:<br>• When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default user.<br>• The job's directory can have entries but a default user is not cataloged there.<br>• The object that is cataloged with the name R?IOUSER is not a user object. The name R?IOUSER should be treated as a reserved word.<br>• The user parameter is a token for an object that is not a user object. |
| E$NOT$CONFIGURED | 0008H | This system call is not part of the present configuration. |
| E$PATHNAME$SYNTAX | 003EH | One or more of the following conditions caused this exception:<br>• The specified path name contains invalid characters or has a length of zero. The path name can include any printable ASCII character except the slash (/), up-arrow (↑), and circumflex (^).<br>• The subpath of the specified remote file exceeds 127 bytes in length. |
| E$TYPE | 8002H | One or more of the following conditions caused this exception:<br>• The prefix parameter is a token for an object that is not of the correct type. It must be either a connection object or a logical device object. (Logical device objects are created by the Extended I/O System.)<br>• The resp$mbox parameter in the call is a token for an object that is not a mailbox. |

## Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$DEV$DETACHING | 0039H | The file specified is on a device that the system is detaching. |
| E$FNEXIST | 0021H | A file in the specified path, or the target file itself, does not exist or is marked for deletion. |
| E$FTYPE | 0027H | The string pointed to by the subpath$ptr parameter contains a filename that should be the name of a directory, but is not. (Except for the last file, each file in a path must be a named directory.) |
| E$INVALID$FNODE | 003DH | The fnode for the specified file is invalid. The file cannot be accessed. |
| E$IO | 002BH | An I/O error occurred, which might have prevented the operation from completing. Examine the unit$status field of the IORS for more information. |
| E$IO$MEM | 0042H | The memory available to the Basic I/O System job is not sufficient to complete the call. |
| E$LIMIT | 0004H | Processing this call would deplete the remote server's resources. For a list of remote server resources, refer to the *iRMX® Networking Software User's Guide*. |
| E$NAME$NEXIST | 0049H | The user object does not represent a verified user or the user object is not properly defined in the remote server's User Definition File (UDF). |
| E$PASSWORD$-MISMATCH | 004BH | The password of the user object does not match the password of the corresponding user defined on the remote server. |
| E$PATHNAME$-SYNTAX | 003EH | The syntax of the specified remote file path name is illegal. Path names for remote files must follow the naming conventions of the server. |
| E$UDF$IO | 02D0H | An error occurred while accessing the remote server's User Definition File (UDF). |

A$CHANGE$ACCESS changes the access rights to a named data or directory file.

```
CALL RQ$A$CHANGE$ACCESS(user, prefix, subpath$ptr, id, access,
                        resp$mbox, except$ptr);
```

## Input Parameters

user
: A TOKEN for the user object to be inspected in access checking. A value of SELECTOR$OF(NIL) specifies the default user for the calling task's job.

prefix
: A TOKEN for the connection object to be used as the path prefix. A SELECTOR$OF(NIL) specifies the default prefix for the calling task's job.

subpath$ptr
: A POINTER to a STRING giving the subpath of the file whose access is to be changed. A null string indicates that the prefix itself designates the desired file.

id
: A WORD containing the ID number of the user whose access is to be changed. If this ID does not already exist in the ID-access mask list, it is added. This list may contain a total of three ID-access pairs.

access
: A BYTE mask giving the new access rights for the ID. If the entire BYTE is set to zero, the Basic I/O System removes the specified ID from the access list of the file. If the BYTE is nonzero, the meaning of the various bit settings depend upon whether the file is a data file or a directory file. The following two tables correlate the bit position and the kind of access. (System calls that start with "S$", like S$READ$MOVE, are part of the Extended I/O System.)

If the bit is set to 1, access is to be granted. If the bit is set to 0, access is to be denied. (Bit 0 is low-order bit.)

DATA FILE ACCESS RIGHTS

| Bit | Access |
|---|---|
| 0 | Delete--permission to delete the entire file by using the A$DELETE$FILE or S$DELETE$FILE system calls. Also allows changing the name of the file by using the A$RENAME$FILE or S$RENAME$FILE system calls.<br>This bit is ignored for remote files. |
| 1 | Read--permission to read data from the file by using the A$READ or S$READ$MOVE system calls. |
| 2 | Append--permission to write information only at the end of the file by using the A$WRITE or S$WRITE$MOVE system calls. This does not include permission to write over information already in the file or permission to truncate the file.<br>This bit must be set to the same value as bit 3 (Update) for remote files. |
| 3 | Update--permission to write over any information in the file by using the A$WRITE or S$WRITE$MOVE system calls, and permission to truncate the file using the A$TRUNCATE or S$TRUNCATE$FILE system calls. This does not include permission to add information to the end of the file.<br>This bit must be set to the same value as bit 2 (Append) for remote files. |
| 4-7 | Reserved. Set to zero. |

DIRECTORY ACCESS RIGHTS

Bits — Access

0 — Delete--permission to delete the directory by using the S$DELETE$FILE or A$DELETE$FILE system calls. Also allows changing the name of the directory by using the S$RENAME$FILE or A$RENAME$FILE system calls.

This bit is ignored for remote directories.

1 — Display--permission to read information from the directory by using the A$READ, A$GET$DIRECTORY$ENTRY, or S$READ$MOVE system calls.

2 — Add entry--permission to add files to the directory by using the A$CREATE$FILE, A$CREATE$DIRECTORY, A$RENAME$FILE, S$CREATE$FILE, S$CREATE$DIRECTORY, or S$RENAME$FILE system calls. This does not include permission to change existing entries.

3 — Change entry--permission to change the access list associated with a file contained in the directory. In other words, permission to use the A$CHANGE$ACCESS or S$CHANGE$ACCESS system calls. This does not include permission to add new entries or change the access list of the directory in which the file is cataloged.

This bit is ignored for remote directories.

4-7 — Reserved. Set to zero.

## Output Parameters

resp$mbox

A TOKEN for the mailbox that receives an IORS indicating the result of the call (for details on the IORS, see Appendix A). A value of SELECTOR$OF(NIL) means that you do not want to receive an IORS.

If it receives an IORS, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

except$ptr

A POINTER to a WORD where the sequential condition code will be returned.

## Description

A$CHANGE$ACCESS system call applies to named files only. This call has no effect on existing connections to the file. It is called to change the access rights to a named data or directory file. Depending on the contents of the "id" and "access" parameters specified in the system call, users may be added to or deleted from the file's ID-access mask list, or the access privileges granted to a particular user may be changed.

### NOTE

> The caller must be the owner of the file or must have change entry access to the file's parent directory. However, if the owner is "WORLD", that is, 0FFFFH, then any task may change the access mask of the file. If system manager support is configured, user 0 may change the access rights of any file regardless of which user is the owner.

## Special Considerations for iRMX®-NET

You cannot change the access rights of a virtual root directory, because a virtual root directory has no assigned owner. If you attempt to change the access rights of a virtual root directory, an E$FACCESS condition code is returned.

## Condition Codes

A$CHANGE$ACCESS returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide*.

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

## Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

E$OK 0000H No exceptional conditions.

E$DEV$OFFLINE 002EH The prefix parameter in this system call refers to a logical connection. One of the following is true of the device that is associated with the connection:

- It has been physically attached but is off-line.
- It has never been physically attached. (For example, LOGICAL$ATTACH$DEVICE, an EIOS call, was used. This call does not cause the device to be physically attached until another EIOS call references the logical device object.)

E$EXIST 0006H At least one of the following is true:

- One or more of the following parameters is not a token for an existing object:
  - The user parameter
  - The prefix parameter
  - The response mailbox parameter
- The prefix connection is being deleted.
- The remote driver connection is no longer active.

E$IFDR 002FH This system call applies only to named files, but the prefix and subpath parameters specify some other type of file.

E$LIMIT 0004H Processing this call would cause one or more of these limits to be exceeded:

- The object limit for this job.
- The maximum number of outstanding I/O operations for the user object specified in the call (255 decimal).
- The number of I/O operations that can be outstanding at one time for the caller's job (255 decimal).
- The number of outstanding I/O operations for a remote file has been exceeded.

E$MEM 0002H The memory available to the calling task's job is not sufficient to complete this call.

| | | |
|---|---|---|
| E$NOPREFIX | 8022H | The calling task specified a default prefix (prefix parameter equals SELECTOR$OF(NIL)), but no default prefix can be found because of one of the following:<br>• When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default prefix.<br>• The job's directory can have entries but no default prefix is cataloged there. |
| E$NOUSER | 8021H | If the user parameter in this call is not SELECTOR$OF(NIL), then the parameter is not a token for a user object.<br>If the user parameter is SELECTOR$OF(NIL), it specifies a default user, but no default user can be found because of one of the following reasons:<br>• When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default user.<br>• The job's directory can have entries but no default user is cataloged there.<br>• The object which is cataloged with the name R?IOUSER is not a user object. The name R?IOUSER should be treated as a reserved word. |
| E$NOT$CON-FIGURED | 0008H | This system call is not part of the present configuration. |
| E$PATHNAME$-SYNTAX | 003EH | One or more of the following conditions caused this exception:<br>• The specified path name contains invalid characters or has a length of zero. The path name can include any printable ASCII character except the slash (/), up-arrow (↑), and circumflex (^).<br>• The subpath of the specified remote file exceeds 127 bytes in length. |
| E$SUPPORT | 0023H | The connection was not created by this job. |

| | | |
|---|---|---|
| E$TYPE | 8002H | One or more of the following conditions caused this exception:<br>• The user token designates a connection of the wrong type.<br>• The prefix parameter is a token for an object that is not of the correct type. It must be either a connection object or a logical device object. (Logical device objects are created by the Extended I/O System.)<br>• The resp$mbox parameter in the call is a token for an object that is not a mailbox. |

### Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$DEV$DETACHING | 0039H | The file specified is on a device that the system is detaching. |
| E$FACCESS | 0026H | The user object in the parameter list is not the owner of the specified file, nor does it have "change entry" access to the parent directory. |
| E$FNEXIST | 0021H | A file in the specified path, or the target file itself, does not exist or is marked for deletion. |
| E$FTYPE | 0027H | The string pointed to by the subpath$ptr parameter contains a filename which should be the name of a directory, but is not. (Except for the last file, each file in a path must be a named directory.) |
| E$INVALID$FNODE | 003DH | The fnode for the specified file is invalid. The file cannot be accessed. |
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the IORS for more information. For information on IORS structures, see Appendix A. |
| E$IO$MEM | 0042H | The memory available to the Basic I/O System job is not sufficient to complete this call. |

| | | |
|---|---|---|
| E$LIMIT | 0004H | Processing this call would deplete the remote server's resources. For a list of remote server resources, refer to the *iRMX® Networking Software User's Guide*. |
| E$NAME$NEXIST | 0049H | The user object does not represent a verified user or the user object is not properly defined in the remote server's User Definition File (UDF). |
| E$NOT$FILE$CONN | 0032H | The subpath$ptr parameter is a null pointer and the prefix parameter is not a file connection. |
| E$PASSWORD$-MISMATCH | 004BH | The password of the user object does not match the password of the corresponding user defined on the remote server. |
| E$PATHNAME$-SYNTAX | 003EH | The syntax of the specified remote file path name is illegal. Path names for remote files must follow the naming conventions of the server. |
| E$SUPPORT | 0023H | The call attempted to add another access ID to the list of access ID's. The access list already contained the limit of three such ID's. |
| E$UDF$IO | 02D0H | An error occurred while accessing the remote server's User Definition File (UDF). |

A$CLOSE closes an open file connection.

```
    CALL RQ$A$CLOSE(connection, resp$mbox, except$ptr);
```

## Input Parameter

| | |
|---|---|
| connection | A TOKEN for the file connection to be closed. |

## Output Parameters

| | |
|---|---|
| resp$mbox | A TOKEN for the mailbox that receives an I/O request/result segment (IORS) indicating the result of the call (for details on the IORS, see Appendix A). A value of SELECTOR$OF(NIL) means that you do not want to receive an IORS.<br>If it receives an IORS, the calling task should call DELETE$SEGMENT to delete the segment after examining it. |
| except$ptr | A POINTER to a WORD where the sequential condition code will be returned. |

## Description

The A$CLOSE system call closes an open file connection. It is called when the application needs to change the open mode or shared status of the connection. The Basic I/O System will not close the connection until all existing I/O requests for the connection have been satisfied. In addition, the Basic I/O System will not send a response to the response mailbox until the file is closed.

## Condition Codes

A$CLOSE returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide.*

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

## Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | At least one of the following is true:<br>• One or more of the following parameters is not a token for an existing object:<br>- The connection parameter<br>- The resp$mbox parameter<br>• The connection is being deleted.<br>• The connection for a remote driver is no longer active. |
| E$LIMIT | 0004H | At least one of the following is true.<br>• The calling task's job has already reached its object limit.<br>• The number of outstanding I/O operations for a remote connection has been exceeded. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete this call. |
| E$NOT$CON-FIGURED | 0008H | This system call is not part of the present configuration. |
| E$SUPPORT | 0023H | The connection was not created by this job. |
| E$TYPE | 8002H | At least one of the following is true:<br>• The connection parameter is a token for an object that is not a connection.<br>• The resp$mbox parameter is a token for an object that is not a mailbox. |

## Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$CONN$NOT$OPEN | 0034H | The specified connection is not open. |
| E$IO | 002BH | An I/O error occurred, but the operation was successful anyway. |

# A$CREATE$DIRECTORY

A$CREATE$DIRECTORY creates a directory file.

```
CALL RQ$A$CREATE$DIRECTORY(user, prefix, subpath$ptr, access,
                            resp$mbox, except$ptr);
```

## Input Parameters

user — A TOKEN for the user object of the new directory's owner. The user object is inspected to make sure the caller has proper access to the new directory's parent. A SELECTOR$OF(NIL) specifies the default user for the calling task's job.

prefix — A TOKEN for the connection to be used as the path prefix. A SELECTOR$OF(NIL) specifies the default prefix for the calling task's job.

subpath$ptr — A POINTER to a STRING containing the subpath of the directory to be created. The subpath string must not be null, and it must point to an unused location in the directory tree.

access — A BYTE mask giving the owner's initial access rights to the directory. For each bit in the mask, a one grants access and a zero denies it. The possible bit settings are:

| Bit | Meaning |
|---|---|
| 0 | Delete |
| 1 | List |
| 2 | Add Entry |
| 3 | Change Entry |
| 4-7 | Reserved (set to 0) |

## Output Parameters

resp$mbox — A TOKEN for the mailbox that receives the result object of this call. This result object is a directory file connection if the call succeeded, or an I/O request/result segment (IORS) otherwise (for details on the IORS, see Appendix A). To determine the type of object returned, use the Nucleus system call GET$TYPE. If the object received is an IORS, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

| | |
|---|---|
| except$ptr | A POINTER to a WORD where the sequential condition code will be returned. |

## Description

The A$CREATE$DIRECTORY system call is applicable to named directory files only. When called, it creates a new directory file and returns a token for the new file connection. This system call cannot be used to create a connection to an existing directory. To attach to an existing file you should use the A$ATTACH$FILE system call.

### NOTE

The caller must have add-entry access to the parent of the new directory.

## Special Considerations for iRMX®-NET

You cannot create a remote directory with a virtual root directory as its parent. A virtual root directory has no owner and, thus, you cannot have write access to it. If an attempt is made to create such a remote directory, an E$FACCESS condition code is returned.

## Condition Codes

A$CREATE$DIRECTORY returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide*.

The list on the following pages is divided into two parts--one for sequential codes, and one for concurrent codes.

## Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

E$OK 0000H No exceptional conditions.

E$DEV$OFF$LINE 002EH The prefix parameter in this system call refers to a logical connection. One of the following is true of the device that is associated with the connection:

- It has been physically attached but is now off-line.
- It has never been physically attached. (See *iRMX® Basic I/O System User's Guide*, Appendix E for a more detailed explanation.)

E$EXIST 0006H At least one of the following is true:

- One or more of the following parameters is not a token for an existing object:
  - The user parameter
  - The prefix parameter
  - The response mailbox parameter
- The prefix connection is being deleted.
- The connection for a remote driver is no longer active.

E$IFDR 002FH This system call applies only to named directory files, but the prefix and subpath parameters specify some other type of file.

E$LIMIT 0004H Processing this call would cause one or more of these limits to be exceeded:

- The object limit for this job.
- The number of I/O operations that can be outstanding at one time for the user object specified in the call (255 decimal).
- The number of I/O operations that can be outstanding at one time for the caller's job (255 decimal).
- The number of outstanding I/O operations for a remote connection has been exceeded.

E$MEM 0002H The memory available to the calling task's job is not sufficient to complete this call.

E$NOPREFIX 8022H The task specified a default prefix (prefix parameter equals SELECTOR$OF(NIL)), but no default prefix can be found because of one or more of the following reasons:

- When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default prefix.
- The job's directory can have entries but no default prefix is cataloged there.

E$NOUSER 8021H If the user parameter in this call is not SELECTOR$OF(NIL), then the parameter is not a user object.

If the user parameter is SELECTOR$OF(NIL), it specifies a default user, but no default user can be found because of one of the following reasons:

- When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default user.
- The job's directory can have entries but no default user is cataloged there.
- The object that is cataloged with the name R?IOUSER is not a user object. The name R?IOUSER should be treated as a reserved word.

E$NOT$CON-FIGURED 0008H This system call is not part of the present configuration.

E$PATHNAME$-SYNTAX 003EH One or more of the following conditions caused this exception:

- The specified path name contains invalid characters or has a length of zero. The path name can include any printable ASCII character except the slash (/), up-arrow (↑), and circumflex (^).
- The subpath of the specified remote file exceeds 127 bytes in length.

| | | |
|---|---|---|
| E$TYPE | 8002H | At least one of the following is true:<br>• The prefix parameter is a token for an object that is not of the correct type. It must be either a connection object or a logical device object. (Logical device objects are created by the Extended I/O System.)<br>• The resp$mbox parameter in the call is a token for an object that is not a mailbox. |

### Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$DEV$DETACHING | 0039H | The file specified is on a device that the system is detaching. |
| E$FACCESS | 0026H | The user object in the parameter list is not qualified for "add-entry" access to the parent directory. |
| E$FEXIST | 0020H | A file with the specified path name already exists. |
| E$FNEXIST | 0021H | A file in the specified path does not exist or is marked for deletion. |
| E$FNODE$LIMIT | 003FH | The volume already contains the maximum number of files. No more fnodes are available for new files. |
| E$FTYPE | 0027H | The string pointed to by the subpath$ptr parameter contains a filename which should be the name of a directory, but is not. (Except for the last file, each file in a path must be a named directory.) |
| E$INVALID$FNODE | 003DH | The fnode for the specified file (or for a directory in the file's path) is invalid. The file with the invalid fnode cannot be accessed. |
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the IORS for more information. |
| E$IO$MEM | 0042H | The memory available to the Basic I/O System job is not sufficient to complete this call. |
| E$LIMIT | 0004H | Processing this call would deplete the remote server's resources. For a list of remote server resources, refer to the *iRMX® Networking Software User's Guide*. |

| | | |
|---|---|---|
| E$NAME$NEXIST | 0049H | The user object does not represent a verified user or the user object is not properly defined in the remote server's User Definition File (UDF). |
| E$PASSWORD$-MISMATCH | 004BH | The password of the user object does not match the password of the corresponding user defined on the remote server. |
| E$PATHNAME$-SYNTAX | 003EH | The syntax of the specified remote file path name is illegal. Path names for remote files must follow the naming conventions of the server. |
| E$SPACE | 0029H | At least one of the following is true:<br>• The volume is full.<br>• No more files can be created on the remote server's volume. The Remote File Driver cannot distinguish between an E$FNODE$LIMIT and an E$SPACE condition code. |
| E$SUPPORT | 0023H | The Basic I/O System is not configured to support space allocation. |
| E$UDF$IO | 02D0H | An error occurred while accessing the remote server's User Definition File (UDF). |

# A$CREATE$FILE

A$CREATE$FILE creates a physical, stream, or named file.

```
    CALL RQ$A$CREATE$FILE(user, prefix, subpath$ptr, access,
                          granularity, size, must$create,
                          resp$mbox, except$ptr);
```

## Input Parameters

user — A TOKEN for the user object of the owner of the new file. It also furnishes the user ID for any access checking that might occur. A SELECTOR$OF(NIL) specifies the default user for the calling task's job. This parameter is ignored for physical or stream files.

prefix — A TOKEN for a device or file connection. The file created by this call is of the type (physical, stream, or named) that is associated with this parameter. A SELECTOR$OF(NIL) for this parameter specifies the default prefix for the job.

For stream files, if the prefix is a device connection, a new stream file is created. If the prefix is a file connection, a new file connection to the same stream file is created.

For named files, the prefix acts as the starting point in a directory tree scan.

subpath$ptr — A POINTER to a STRING containing the subpath for the named file being created. This parameter does not apply to physical and stream files.

Entering NIL for this parameter, when using a named file driver, causes an unnamed file to be created. This file is automatically deleted when the last connection to it is deleted.

access

A BYTE mask giving the owner's initial access rights to the new file. For each bit, a one grants access and a zero denies it. (Bit 0 is the low-order bit.)

| Bit | Meaning |
|---|---|
| 0 | Delete |
| 1 | Read |
| 2 | Append |
| 3 | Update |
| 4-7 | Reserved (set to 0) |

This parameter does not apply to physical or stream files.

granularity

A WORD giving the granularity of the file being created. This is the size (in bytes) of each logical block of volume space to be allocated to the file. The value specified in this parameter is rounded up, if necessary, to a multiple of the volume granularity. Note that a contiguous file can become noncontiguous when it is extended.

The granularity parameter can have the following values:

| | |
|---|---|
| 0 | Same as volume granularity |
| FFFFH | The file must be contiguous |
| Other | Number of bytes per allocation |

When a contiguous file is extended, space is allocated in volume-granularity units. If "Other" is specified, a multiple of 1024 bytes is recommended. This parameter is ignored for physical, stream, and remote files.

size

A DWORD giving the number of bytes initially reserved for the file. For stream files and existing remote files, this value must equal zero. If you make this value greater than zero for stream files, the reserved space may contain unknown data. For physical files and non-existent remote files, this parameter is ignored.

must$create

A BYTE with values of 1 for TRUE, or 0 for FALSE. Only the least significant bit is checked. This BYTE determines the handling of input paths designating an existing file (see following Description). This parameter applies only to named files.

## Output Parameters

resp$mbox — A TOKEN for the mailbox that receives the result object of this call. This result object is a new file connection if the call succeeded; otherwise, it is an IORS (for details on the IORS, see Appendix A). To determine the type of object returned, use the Nucleus system call GET$TYPE.

If the object received is an IORS, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

## Description

The A$CREATE$FILE system call creates a physical, stream, or named data file and returns a token for the new file connection. If a named file designated by the prefix and subpath parameters already exists, one of the following occurs:

- Error: If the "must$create" parameter is TRUE (1), an error condition code (E$FEXIST) is returned.
- Truncate File: If the "must$create" parameter is FALSE (0) and the path designates an existing data file, a new connection to that file is returned (that is, A$CREATE$FILE acts like A$ATTACH$FILE). In this case, the file is truncated or expanded according to the "size" parameter, so data in the file might be lost. As in the case of A$ATTACH$FILE, the file's owner ID and access list are unchanged.
- Temporary File Created: If the "must$create" parameter is FALSE (0), and the path designates an existing directory file or device, an unnamed temporary file is created on the corresponding device. This file is deleted automatically when the last connection to it is deleted. Because this file is created without a path, it can be accessed only through a connection.

  Any task can create a temporary file by referring to any directory. This is true because temporary files are not listed as ordinary entries in the directory, so no add-entry access is required.

  Unlike local files, when you create a remote file, the remote temporary file is entered in the directory in which you are creating the remote file. Therefore, the task creating the remote file must have write access to this directory. Tasks can access this remote temporary file through its path name, as well as through connections to the file. The remote temporary file is deleted when all connections to it are deleted.

Many of the parameters specified in the A$CREATE$FILE call do not apply to physical and stream files. In these cases, the parameter is ignored.

## NOTE

The caller must have add-entry access to the parent directory of the new named file.

## Special Considerations for iRMX®-NET

You cannot create a remote file with a virtual root directory as its parent. A virtual root directory has no owner and, thus, you cannot have write access to it. If an attempt is made to create such a remote file, an E$FACCESS condition code is returned.

## Condition Codes

A$CREATE$FILE returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide.*

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

### Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$DEV$OFF$LINE | 002EH | The prefix parameter in this system call refers to a logical connection. One of the following is true of the device associated with the connection:<br>• It has been physically attached but is now off-line.<br>• It has never been physically attached. (See *iRMX® Basic I/O System User's Guide*, Appendix E for a more detailed explanation.) |

| | | |
|---|---|---|
| E$EXIST | 0006H | At least one of the following is true:<br>• One or more of the following parameters is not a token for an existing object:<br>- The user parameter<br>- The prefix parameter<br>- The resp$mbox parameter<br>• The prefix connection is being deleted.<br>• The connection for a remote driver is no longer active. |
| E$LIMIT | 0004H | Processing this call would cause one or both of the following limits to be exceeded:<br>• The object limit for this job.<br>• The number of outstanding I/O operations for a remote connection. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete this call. |
| E$NOPREFIX | 8022H | The call specified a default prefix (prefix argument equals SELECTOR$OF(NIL)), but no default prefix can be found because of one of the following reasons:<br>• When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default prefix.<br>• The job's directory can have entries but a default prefix is not cataloged there. |

| | | |
|---|---|---|
| E$NOUSER | 8021H | If the user parameter in this call is not SELECTOR$OF(NIL), then the parameter is not a token for a user object.<br>If the user parameter is SELECTOR$OF(NIL), it specifies a default user, but no default user can be found because of one of the following reasons:<br>• When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default user.<br>• The job's directory can have entries but a default user is not cataloged there.<br>• The object that is cataloged with the name R?IOUSER is not a user object. Another task cataloged an object (not a user object) under the name R?IOUSER. |
| E$NOT$CONFIGURED | 0008H | This system call is not part of the present configuration. |
| E$PATHNAME$SYNTAX | 003EH | At least one of the following is true:<br>• The specified path name contains invalid characters or has a length of zero. The path name can include any printable ASCII character except the slash (/), up-arrow (↑), and circumflex (^).<br>• The subpath of the specified remote file exceeds 127 bytes in length. |
| E$TYPE | 8002H | At least one of the following is true:<br>• The prefix parameter is a token for an object that is not of the correct type. It must be either a connection object or a logical device object. (Logical device objects are created by the Extended I/O System.)<br>• The resp$mbox parameter in the call is a token for an object that is not a mailbox. |

## Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

E$DEV$DETACHING 0039H The file specified is on a device that the system is detaching.

E$FACCESS 0026H One of the following is true:

- No file with the specified pathname exists, and the specified user object does not have "add-entry" access to the parent directory.
- A file with the specified pathname exists, but the specified user object does not have "update" access to the file.

E$FEXIST 0020H The "must$create" parameter in the call is TRUE, and the file already exists. (See the Description section.)

E$FNEXIST 0021H A file in the specified path does not exist or is marked for deletion.

E$FNODE$LIMIT 003FH The volume already contains the maximum number of files. No more fnodes are available for new files.

E$FTYPE 0027H The string pointed to by the subpath$ptr parameter contains a filename which should be the name of a directory, but is not. (Except for the last file, each file in a path must be a named directory.)

E$INVALID$FNODE 003DH The fnode for the specified file (or for a directory in the file's path) is invalid. The file with the invalid fnode cannot be accessed.

E$IO 002BH An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the IORS for more information.

E$IO$MEM 0042H The memory available to the Basic I/O System job is not sufficient to complete this call.

E$LIMIT 0004H Processing this call would deplete the remote server's resources. For a list of remote server resources, refer to the *iRMX® Networking Software User's Guide*.

| | | |
|---|---|---|
| E$NAME$NEXIST | 0049H | The user object does not represent a verified user or the user object is not properly defined in the remote server's User Definition File (UDF). |
| E$PASSWORD$-MISMATCH | 004BH | The password of the user object does not match the password of the corresponding user defined on the remote server. |
| E$PATHNAME$-SYNTAX | 003EH | The syntax of the specified remote file path name is illegal. Path names for remote files must follow the naming conventions of the server. |
| E$SHARE | 0028H | The file this call is attempting to create already exists and is open. It was opened with the characteristic "no share with writers." (See the A$OPEN call in this manual.) |
| E$SPACE | 0029H | At least one of the following is true:<br>• The volume is full.<br>• No more files can be created on the remote server's volume. The Remote File Driver cannot distinguish between an E$FNODE$LIMIT and an E$SPACE condition code. |
| E$SUPPORT | 0023H | One of the following is true:<br>• The file exists and the must$create parameter is FALSE. When the Basic I/O System was configured, an option was chosen that prevented this combination, so that files could not be automatically truncated to zero size. See the Description section for this system call.<br>• The Basic I/O System is not configured to allow space allocation on volumes.<br>• The Remote File Driver does not support creation of a contiguous file.<br>• The Remote File Driver does not support truncating existing remote files to zero size. |
| E$UDF$IO | 02D0H | An error occurred while accessing the remote server's User Definition File (UDF). |

# A$DELETE$CONNECTION

A$DELETE$CONNECTION deletes a named file connection created by A$CREATE$FILE, A$CREATE$DIRECTORY, or A$ATTACH$FILE.

```
CALL RQ$A$DELETE$CONNECTION(connection, resp$mbox, except$ptr);
```

## Input Parameter

connection — A TOKEN for the file connection to be deleted.

## Output Parameters

resp$mbox — A TOKEN for the mailbox that receives an I/O request/result segment (IORS) indicating the result of the call (for details on the IORS, see Appendix A). A value of SELECTOR$OF(NIL) means that you do not want to receive an IORS.

If it receives an IORS, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

## Description

The A$DELETE$CONNECTION system call deletes a connection object. It also deletes the associated file if both of the following are true:

- The file is already marked for deletion (by a previous A$DELETE$FILE call) or is an unnamed file.
- The specified connection is the only connection to the file.

If a connection is open when A$DELETE$CONNECTION is called, it is closed before being deleted.

### NOTE

Connections should be deleted when no longer needed.

## Condition Codes

A$DELETE$CONNECTION returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide*.

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

### Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | At least one of the following is true:<br>• One or more of the following parameters is not a token for an existing object:<br>- The connection parameter<br>- The resp$mbox parameter<br>• The connection is being deleted.<br>• The connection for a remote driver is no longer active. |
| E$LIMIT | 0004H | The calling task's job has already reached its object limit. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete this call. |
| E$NOT$CON-<br>FIGURED | 0008H | This system call is not part of the present configuration. |
| E$NOT$FILE$CONN | 0032H | The connection parameter is a device connection, not a file connection. |
| E$SUPPORT | 0023H | The specified connection was not created by this job. |
| E$TYPE | 8002H | One or more of the following is a token for an object that is not of the correct type:<br>• The connection parameter.<br>• The resp$mbox parameter. |

## Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$IO | 002BH | An I/O error occurred, but the connection was still deleted. |

A$DELETE$FILE marks a file for deletion.

```
    CALL RQ$A$DELETE$FILE(user, prefix, subpath$ptr, resp$mbox,
                              except$ptr);
```

## Input Parameters

user
: A TOKEN for the user object to be inspected in access checking. A SELECTOR$OF(NIL) specifies the default user for the calling task's job. This parameter does not apply to stream files.

prefix
: A TOKEN for the connection object to be used as the path prefix. A SELECTOR$OF(NIL) specifies the default prefix for the calling task's job.

subpath$ptr
: A POINTER to a STRING giving the subpath for the file being deleted. A null string indicates that the prefix itself designates the desired file. In this instance, the user parameter is ignored, since access checking was already performed when the file was attached. This parameter does not apply to stream files.

## Output Parameters

resp$mbox
: A TOKEN for a mailbox that receives an I/O request/result segment (IORS) when the file is marked for deletion (for details on the IORS, see Appendix A). The file will not actually be deleted until all connections to the file are deleted, as explained under the Description below. A value of SELECTOR$OF(NIL) means that you do not want to receive an IORS.

: If it receives an IORS, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

except$ptr
: A POINTER to a WORD where the sequential condition code will be returned.

## Description

The A$DELETE$FILE system call applies to stream and named files only. When called, it marks the designated file for deletion and removes the file's entry from the parent directory. The entry is removed immediately, but the file is not actually deleted until all connections to the file have been severed (by A$DELETE$CONNECTION calls). Directory files cannot be deleted unless they are empty.

### NOTE

The caller must have delete access to the file.

## Special Considerations for iRMX®-NET

You cannot delete a remote file which has a virtual root directory as its parent, because a virtual root directory has no assigned owner. To delete a file, you must have write access to its parent directory. If you attempt to delete a remote file whose parent directory is a virtual root directory, an E$FACCESS condition code is returned.

## Condition Codes

A$DELETE$FILE returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide*.

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

### Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$DEV$OFF$LINE | 002EH | The prefix parameter in this system call refers to a logical connection. One of the following is true of the device that is associated with the connection:<br>• It has been physically attached but is now off-line.<br>• It has never been physically attached. (See Appendix E in the *iRMX® Basic I/O System User's Guide* for a more detailed explanation.) |

| | | |
|---|---|---|
| E$EXIST | 0006H | At least one of the following is true:<br>• One or more of the following parameters is not a token for an existing object:<br>- The user parameter<br>- The prefix parameter<br>- The response mailbox parameter<br>• The prefix connection is being deleted.<br>• The connection for a remote driver is no longer active. |
| E$IFDR | 002FH | This system call applies only to named or stream files, but the prefix and subpath parameters specified a physical file. |
| E$LIMIT | 0004H | Processing this call would exceed one or more of the following limits:<br>• The object limit for this job.<br>• The number of I/O operations that can be outstanding at one time for the user object specified in the call (255 decimal).<br>• The number of I/O operations that can be outstanding at one time for the caller's job (255 decimal).<br>• The number of outstanding I/O operations for a remote connection has been exceeded. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete this call. |
| E$NOPREFIX | 8022H | The call specified a default prefix (prefix argument equals SELECTOR$OF(NIL)), but no default prefix can be found because of one of the following reasons:<br>• When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default prefix.<br>• The job's directory can have entries but no default prefix is cataloged there. |

| | | |
|---|---|---|
| E$NOUSER | 8021H | If the user parameter in this call is not SELECTOR$OF(NIL), then the parameter is not a token for a user object.<br>If the user parameter is SELECTOR$OF(NIL), it specifies a default user, but no default user can be found because of one of the following reasons:<br>• When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default user.<br>• The job's directory can have entries but no default user is cataloged there.<br>• The object that is cataloged with the name R?IOUSER is not a user object. The name R?IOUSER should be treated as a reserved word. |
| E$NOT$CONFIGURED | 0008H | This system call is not part of the present configuration. |
| E$PATHNAME$SYNTAX | 003EH | At least one of the following is true:<br>• The specified path name contains invalid characters or has a length of zero. The path name can include any printable ASCII character except the slash (/), up-arrow (↑), and circumflex (^).<br>• The subpath of the specified remote file exceeds 127 bytes in length. |
| E$SUPPORT | 0023H | The specified connection was not created by this job. |
| E$TYPE | 8002H | At least one of the following is true:<br>• The prefix parameter is a token for an object that is not of the correct type. It must be either a connection object or a logical device object. (Logical device objects are created by the Extended I/O System.)<br>• The resp$mbox parameter in the call is a token for an object that is not a mailbox. |

### Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$DEV$DETACHING | 0039H | The file specified is on a device that the system is detaching. |
| E$DIR$NOT$EMPTY | 0031H | The call is attempting to delete a directory containing entries. |
| E$FACCESS | 0026H | At least one of the following is true:<br>• The user object does not have delete access to the file.<br>• The call attempted to delete the root directory or a bit map file. |
| E$FNEXIST | 0021H | A file in the specified path, or the target file itself, does not exist or is marked for deletion. |
| E$FTYPE | 0027H | The string pointed to by the subpath$ptr parameter contains a string that should be the name of a directory, but is not. (Except for the last file, each file in a pathname must be a named directory.) |
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the IORS for more information. |
| E$IO$MEM | 0042H | The memory available to the Basic I/O System is not sufficient to complete the call. |
| E$LIMIT | 0004H | Processing this call would deplete the remote server's resources. For a list of remote server resources, refer to the *iRMX® Networking Software User's Guide*. |
| E$NAME$NEXIST | 0049H | The user object does not represent a verified user or the user object is not properly defined in the remote server's User Definition File (UDF). |
| E$NOT$FILE$CONN | 0032H | The subpath$ptr parameter is a null pointer and the prefix parameter is not a file connection. |
| E$PASSWORD$-<br>MISMATCH | 004BH | The password of the user object does not match the password of the corresponding user defined on the remote server. |

| | | |
|---|---|---|
| E$PATHNAME$-SYNTAX | 003EH | The syntax of the specified remote file path name is illegal. Path names for remote files must follow the naming conventions of the server. |
| E$UDF$IO | 02D0H | An error occurred while accessing the remote server's User Definition File (UDF). |

A$GET$CONNECTION$STATUS returns information about a file connection.

```
    CALL RQ$A$GET$CONNECTION$STATUS(connection, resp$mbox, except$ptr);
```

## Input Parameter

connection — A TOKEN for the file connection whose status is desired.

## Output Parameters

resp$mbox — A TOKEN for the mailbox that is to receive a connection-status segment. The calling task is responsible for deleting the connection-status segment after examining it.

The information in this segment is structured as follows:

```
DECLARE conn$status STRUCTURE(
                    status              WORD,
                    file$driver         BYTE,
                    flags               BYTE,
                    open$mode           BYTE,
                    share$mode          BYTE,
                    file$ptr            DWORD,
                    access              BYTE);
```

These fields are interpreted as follows:

status — A condition code giving the outcome of the status-fetch operation. If this code is not E$OK, the remaining fields must be considered invalid.

file$driver — Tells the type of file driver to which this connection is attached. Possible values are:

| Value | Type |
|---|---|
| 1 | Physical |
| 2 | Stream |
| 4 | Named |
| 5 | Remote |

flags — Contains two flag bits. If bit 1 is set to one, this connection is active and can be opened. If bit 2 is set, this connection is a device connection. (Bit 0 is the low-order bit.)

open$mode — The mode established when this connection was opened. Possible values are:

| | |
|---|---|
| 0 | Connection is closed |
| 1 | Open for reading |
| 2 | Open for writing |
| 3 | Open for reading and writing |

share$mode — The sharing mode established when this connection was opened. Possible values are:

| | |
|---|---|
| 0 | Private use only |
| 1 | Share with readers only |
| 2 | Share with writers only |
| 3 | Share with all users |

file$ptr — The current byte location of the file pointer for this connection.

access — The access rights for this connection. For each bit, a one grants access and a zero denies it. (Bit 0 is the low-order bit.)

| Bit | Data File | Directory |
|---|---|---|
| 0 | Delete | Delete |
| 1 | Read | List |
| 2 | Append | Add Entry |
| 3 | Update | Change Entry |
| 4-7 | Reserved | Reserved |

For remote files, the access bits are interpreted as follows:

| Bit | Data File | Directory |
|---|---|---|
| 0 | Ignored | Ignored |
| 1 | Read | Display |
| 2 | Write (must be set the same as bit 3) | Write |
| 3 | Write (must be set the same as bit 2) | Ignored |
| 4-7 | Reserved | Reserved |

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

## Description

The A$GET$CONNECTION$STATUS system call returns a segment containing status information about a file connection.

## Special Considerations for iRMX®-NET

When the status of a file connection to a virtual root directory is requested, display permission is granted and write permission is denied. As a result, bit 1 of the access field is set to 1 and bit 2 is set to 0.

Also, unlike a local named file, the access rights of a remote named file are not checked when a connection to the file is created. Instead, the remote named file's access rights are checked during operations on the connection.

The above discrepancy won't affect your programs if you do the following:

- Open, delete, and rename files prior to changing their access lists.
- Establish connections to files after changing their access lists.

## Condition Codes

A$GET$CONNECTION$STATUS returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide*.

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

### Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | At least one of the following is true:<br>• One or more of the following parameters is not a token for an existing object:<br>- The connection parameter<br>- The resp$mbox parameter<br>• The connection is being deleted. |

| | | |
|---|---|---|
| | | • The connection for a remote driver is no longer active. |
| E$LIMIT | 0004H | At least one of the following is true:<br>• The calling task's job has already reached its object limit.<br>• The number of outstanding I/O operations for a remote connection has been exceeded. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete the call. |
| E$NOT$CON-FIGURED | 0008H | This system call is not part of the present configuration. |
| E$SUPPORT | 0023H | The specified connection parameter is not valid in this system call because the connection was not created by this job. |
| E$TYPE | 8002H | At least one of the following is true:<br>• The connection parameter is a token for an object that is not a connection.<br>• The resp$mbox parameter is a token for an object that is not a mailbox. |

### Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$IO | 002BH | An I/O error occurred, which might or might not have prevented the operation from being completed. Examine the unit$status field of the IORS for more information. |
| E$NOT$FILE$CONN | 0032H | For remote files, the connection parameter must be a file connection, not a device connection. |

A$GET$DIRECTORY$ENTRY returns the file name associated with a named directory file entry.

```
CALL RQ$A$GET$DIRECTORY$ENTRY(connection, entry$num, resp$mbox,
                              except$ptr);
```

## Input Parameters

connection — A TOKEN for the directory file with the desired entry.

entry$num — A WORD giving the entry number of the desired file name. Entries within a directory are numbered sequentially starting from zero. The E$EMPTY$ENTRY condition code will be returned if there is no entry associated with this number.

## Output Parameters

resp$mbox — A TOKEN for the mailbox that will receive a directory-entry segment. The task making the A$GET$DIRECTORY$ENTRY call is responsible for deleting this segment after examining it.

Information in this segment is structured as follows:

```
DECLARE dir$entry$info  STRUCTURE(
                                status  WORD,
                                name (14)   BYTE);
```

where

status — Indicates how the operation was completed. E$OK, E$EMPTY$ENTRY, and E$DIR$END condition codes all indicate successful completion.

name — File name contained in the specified entry. The file name is left-justified and padded with blanks to the right. This field is valid only if status = E$OK.

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

## Description

The A$GET$DIRECTORY$ENTRY system call applies to named files only. When called, it returns the file name associated with a specified directory entry. This name is a single subpath component for a file whose parent is the designated directory. As an alternative to using this system call, an application task can open and read a directory file.

### NOTE

The caller must have display access to the designated directory.

## Special Considerations for iRMX®-NET

The A$GET$DIRECTORY$ENTRY system call is not supported for remote directories. However, remote directories can be read with the A$OPEN, A$READ, S$OPEN, and S$READ$MOVE system calls.

## Condition Codes

A$GET$DIRECTORY$ENTRY returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide*.

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

### Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | At least one of the following is true:<br>• One or more of the following parameters is not a token for an existing object:<br>- The connection parameter<br>- The resp$mbox parameter<br>• The connection is being deleted. |

| | | |
|---|---|---|
| E$IFDR | 002FH | At least one of the following is true:<br>• This system call applies only to named directories, but the connection parameter specifies another type of file.<br>• The connection parameter specifies a remote directory, but the Remote File Driver does not support this system call. |
| E$LIMIT | 0004H | The calling task's job has already reached its object limit. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete this call. |
| E$NOT$CON-FIGURED | 0008H | This system call is not part of the present configuration. |
| E$SUPPORT | 0023H | The specified connection was not created by this job. |
| E$TYPE | 8002H | At least one of the following is true:<br>• The connection parameter is a token for an object that is not a connection.<br>• The resp$mbox parameter is a token for an object that is not a mailbox. |

### Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$DIR$END | 0025H | The entry$num parameter is greater than the number of entries in the directory. |
| E$EMPTY$ENTRY | 0024H | The file entry designated in the call is empty. |
| E$FACCESS | 0026H | The specified connection is not qualified for "display" access to the directory. |
| E$FTYPE | 0027H | The specified connection does not refer to a directory. |
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. |

The A$GET$EXTENSION$DATA system call returns extension data stored with a Basic I/O System file.

```
CALL RQ$A$GET$EXTENSION$DATA(connection, resp$mbox, except$ptr);
```

## Input Parameters

connection — A TOKEN of a connection to a file whose extension data is desired.

## Output Parameter

resp$mbox — A TOKEN for the mailbox that will receive a segment containing the named file-status information. The calling task is responsible for deleting this segment after examining it.

Structure of the named file-status information is as follows:

```
DECLARE ext$data$seg STRUCTURE (
                        status      WORD,
                        count       BYTE,
                        info(*)     BYTE);
```

These fields are interpreted as follows:

status — A condition code indicating the outcome of the status-fetch operation. If this code is not E$OK, the remaining fields must be considered invalid.

count — A number (from 0 to 255 decimal) indicating the number of bytes returned. This field is always set to 0 for remote files.

info — The extension data.

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

## Description

Associated with each file created through the Basic I/O System is a file descriptor containing information about the file. Some of that information is used by the Basic I/O System and can be accessed by tasks through the A$GET$FILE$STATUS system call. Up to 255 additional bytes of the file descriptor, known as extension data, are available for use by Operating System extensions. For named volumes, the first three bytes of this extension data is reserved for use by the Basic I/O System. OS extensions can write extension data by using A$SET$EXTENSION$DATA and they can read extension data by using A$GET$EXTENSION$DATA.

When a task calls A$GET$EXTENSION$DATA, it specifies a response mailbox to which the system returns a segment with the extension data. The information is located in the low-memory portion of the segment. A$GET$EXTENSION$DATA can only be applied to connections created via the named file driver.

## Condition Codes

A$GET$EXTENSION$DATA can return condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide.*

The following list is divided into two parts--one for sequential codes and one for concurrent codes.

### Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | At least one of the following is true:<br>• One or more of the following parameters is not a token for an existing object:<br>- The connection parameter<br>- The resp$mbox parameter<br>• The connection is being deleted.<br>• The connection for a remote driver is no longer active. |

| | | |
|---|---|---|
| E$IFDR | 002FH | This system call applies only to named files, but the prefix and subpath parameters specify another type of file. |
| E$LIMIT | 0004H | At least one of the following is true:<br>• The calling task's job has already reached its object limit.<br>• The number of outstanding I/O operations for a remote connection has been exceeded. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete the call. |
| E$NOT$CON-FIGURED | 0008H | This system call is not part of the present configuration. |
| E$SUPPORT | 0023H | The specified connection was not created by this job. |
| E$TYPE | 8002H | At least one of the following is true:<br>• The connection parameter is a token for an object that is not a connection.<br>• The resp$mbox parameter is a token for an object that is not a mailbox. |

### Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. |

A$GET$FILE$STATUS returns status and attribute information about a file.

```
CALL RQ$A$GET$FILE$STATUS(connection, resp$mbox, except$ptr);
```

## Input Parameter

connection — A TOKEN for a connection to the file whose status is sought.

## Output Parameters

resp$mbox — A TOKEN for the mailbox that receives a segment containing a data structure with the status information for the specified file. The information in the first part of this structure--down to the dev$conn field--is returned for any file (physical, stream, or named), but information from the file$id field down to the end of the structure is provided only for named files. The contents of the named$file field indicates whether the file is a named file.

```
DECLARE file$info STRUCTURE(
          status          WORD,
          num$conn        WORD,
          num$reader      WORD,
          num$writer      WORD,
          share           BYTE,
          named$file      BYTE,
          dev$name(14)    BYTE,
          file$drivers    WORD,
          functs          BYTE,
          flags           BYTE,
          dev$gran        WORD,
          dev$size        DWORD,
          dev$conn        WORD,
```

Information from this point on is returned only if the file is a named file.

```
      file$id             WORD,
      file$type           BYTE,
      file$gran           BYTE,
      owner$id            WORD,
      create$time         DWORD,
      access$time         DWORD,
      modify$time         DWORD,
      file$size           DWORD,
      file$blocks         DWORD,
      vol$name(6)         BYTE,
      vol$gran            WORD,
      vol$size            DWORD,
      accessor$count      WORD,
      first$access        BYTE,
      first$ID            WORD,
      second$access       BYTE,
      second$ID           WORD,
      third$access        BYTE,
      third$ID            WORD,
      vol$flags           BYTE);
```

These fields are interpreted as follows:

status — A condition code indicating how the get file status operation was completed. If this code is not E$OK, the remaining fields must be considered invalid.

num$conn — The number of connections to the file.

For remote files, this field indicates the number of connections the calling job has to the file.

num$reader — The number of connections currently open for reading.

For remote files, this field is set as follows:

| Connection | num$reader |
|---|---|
| No connection | 0 |
| Connection open - read | 1 |
| Connection open - write | 0 |
| Connection open - read/write | 1 |

num$writer

The number of connections currently open for writing.

For remote files, this field is set as follows:

| Connection | num$writer |
|---|---|
| No connection | 0 |
| Connection open - read | 0 |
| Connection open - write | 1 |
| Connection open - read/write | 1 |

share

The current shared status of the file. Possible values are:

0 Private use only
1 Share with readers only
2 Share with writers only
3 Share with all users

For remote files, a value of 3 is returned if the specified remote file connection is not open. If the remote file is open, the share mode used to open the connection is returned.

named$file

Tells whether this structure contains any information beyond the dev$conn field. 0FFH means yes and 0 means no. 0FFh is always returned for remote files.

dev$name

The name of the physical device where this file resides (same name as in the DUIB). This name is left-justified and padded with blanks to the right.

For remote files, the name of the remote server on which the file resides is returned.

file$drivers

A bit map that tells what kinds of files can reside on this device. If bit n is on, then file driver n+1 can be used. Bit 0 is the low-order bit.

| Bit | Driver No. | Driver |
|---|---|---|
| 0 | 1 | Physical file |
| 1 | 2 | Stream file |
| 2 | 3 | reserved |
| 3 | 4 | Named file |
| 4 | 5 | Remote file |

functs

A bit map that describes the functions supported by the device where this file resides. A bit set to one indicates the corresponding function is supported. Bit 0 is the low-order bit.

This field is not supported by iRMX-NET. A value of 0 is always returned for remote files.

| Bit | Function |
|---|---|
| 0 | F$READ |
| 1 | F$WRITE |
| 2 | F$SEEK |
| 3 | F$SPECIAL |
| 4 | F$ATTACH$DEV |
| 5 | F$DETACH$DEV |
| 6 | F$OPEN |
| 7 | F$CLOSE |

For details on these functions, refer to the *iRMX® Device Drivers User's Guide.*

flags

Meaningful only for diskette drives. This field is interpreted as follows. (Bit 0 is the low-order bit.)

This field is not supported by iRMX-NET. A value of 0 is always returned for remote files.

| Bit | Meaning |
|---|---|
| 0 | 0 = bits 1-7 are not significant<br>1 = bits 1-7 are significant |
| 1 | 0 = single density<br>1 = double density |
| 2 | 0 = single sided<br>1 = double sided |
| 3 | 0 = 8-inch diskette<br>1 = 5 1/4-inch diskette |
| 4 | 0 = standard diskette, meaning that track 0 is single-density with 128-byte sectors<br>1 = a non-standard diskette or not a diskette |
| 5-7 | reserved |

dev$gran — The device granularity, in bytes, of the device where this file resides.

For remote files, this field indicates the buffer size of the server associated with the remote file.

dev$size — The storage capacity of the device, in bytes.

For remote files, this field indicates the total storage capacity of all server devices containing public files. The total capacity includes the portions of those devices that contain private files.

dev$conn — The number of connections to the device.

For remote files, this field contains the number of connections that local users have to files on the remote server.

The information from here to the end of the structure is returned only for named files, as indicated by a value of 0FFH in the named$file field.

file$id — A number that distinguishes this file from all other files on the same device. The Disk Verification Utility refers to this number as an FNODE. For information on the disk verify utility, see *iRMX® Disk Verification Utility Reference Manual.*

file$type — Indicates the type of the file: 6 means directory file; and 8 means data file.

file$gran — The file granularity, as a multiple of vol$gran. For example, if file$gran is 2 and vol$gran is 256, then the file's granularity is 512.

A value of 1 is always returned for remote files.

owner$id — The first ID in the user object that was presented to the Basic I/O System when the file was created.

create$time — The date and time when the file was created. Whether the Basic I/O System maintains this field is a configuration option. The Basic I/O System maintains the date/time value as the number of seconds since midnight, January 1, 1978.

access$time

The date and time when the file was last accessed. Whether the Basic I/O System maintains this field is a configuration option. The Basic I/O System maintains the date/time value as the number of seconds since midnight, January 1, 1978.

modify$time

The date and time when the file was last modified. Whether the Basic I/O System maintains this field is a configuration option. The Basic I/O System maintains the date/time value as the number of seconds since midnight, January 1, 1978.

file$size

The total size of the file, in bytes.

file$blocks

The number of volume blocks allocated to this file. A volume block is a contiguous area of storage that contains vol$gran bytes of data.

vol$name

The left-adjusted, null-padded ASCII name for the volume containing this file.

vol$gran

The volume granularity, in bytes.

vol$size

The storage capacity, in bytes, of the volume on which this file is stored.

accessor$count

The number of IDs in the file's accessor list. (This may have been added after file creation.)

first$access
second$access
third$access

Access masks for as many ID's as are indicated by accessor$count. The bits of the access masks are defined in the following table. An access right is granted if the appropriate bit is set to 1; otherwise, that right is denied. Bit 0 is the low-order bit.

| Bit | Data File | Directory File |
|---|---|---|
| 0 | Delete | Delete |
| 1 | Read | Display |
| 2 | Append | Add Entry |
| 3 | Update | Change Entry |
| 4-7 | Reserved | Reserved |

first$ID
second$ID
third$ID

ID values for the accessors.

**iRMX I Note:** The iRMX I Volume Label does not contain a vol$flags field.

vol$flags Contains flags for general volume information. The following flags are defined:

| Flag | Bit | Meaning |
|---|---|---|
| vf$integerity | 0 | 0 = The volume has been properly shut down.<br>1 = Indicates possible disk corruption (The volume was attached but was not subsequently shut down). |

except$ptr A POINTER to a WORD where the sequential condition code will be returned.

## Description

The A$GET$FILE$STATUS system call returns status and attribute information about the designated file. Certain information is returned for all file driver types. Additional information is returned for named files.

Note that this call returns device-dependent information.

## Condition Codes

A$GET$FILE$STATUS returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide.*

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

## Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | At least one of the following is true:<br>• One or more of the following parameters is not a token for an existing object:<br>- The resp$mbox parameter<br>• The connection is being deleted.<br>• The connection for a remote driver is no longer active. |
| E$LIMIT | 0004H | At least one of the following is true:<br>• The calling task's job has already reached its object limit.<br>• The number of outstanding I/O operations for a remote connection has been exceeded. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete the call. |
| E$NOT$CON-FIGURED | 0008H | This system call is not part of the present configuration. |
| E$SUPPORT | 0023H | The specified connection was not created by this job. |
| E$TYPE | 8002H | One or more of the following parameters is a token for an object of the wrong type:<br>• The connection parameter<br>• The resp$mbox parameter |

## Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. |
| E$NOT$FILE$CONN | 0032H | For remote files, the connection parameter must be a file connection, not a device connection. |

| | | |
|---|---|---|
| E$UDF$IO | 02D0H | An error occurred while accessing the remote server's User Definition File (UDF). |

# A$GET$PATH$COMPONENT

A$GET$PATH$COMPONENT returns the name of a named file as the file is known in its parent directory.

```
CALL RQ$A$GET$PATH$COMPONENT(connection, resp$mbox, except$ptr);
```

## Input Parameter

connection — A TOKEN for the file connection whose name is sought.

## Output Parameters

resp$mbox — A TOKEN for the mailbox that will receive the file$name segment. This segment contains the file name associated with the designated connection and is structured as follows:

```
DECLARE  file$name  STRUCTURE(
    status  WORD,
    name    STRING);
```

These fields are interpreted as follows:

status — A condition code indicating the outcome of the operation.

name — A STRING giving the desired file name. This name is the same as the last item in the subpath string specified when the file was created or renamed.

The task that makes the A$GET$PATH$COMPONENT call is responsible for deleting the file$name segment after examining it.

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

## Description

A caller who knows the token for a connection to a file can invoke this system call and receive the name of the file in return. This name is the name by which the file is cataloged in its parent directory. If the connection is to the root directory of a volume (that is, if no parent directory exists), a null string is returned. A null string is also returned if the file is marked for deletion or is a temporary file.

A$GET$PATH$COMPONENT can be called no matter what type of file is supported, but if a connection to a physical or stream file is specified, the call simply returns a null string.

The A$GET$PATH$COMPONENT system call can be used in combination with the A$ATTACH$FILE system call to derive all of the components of a path name. Suppose, for example, that a file has the path name A/B/C, and that your task has only a token for the file. The following sequence of calls will reveal all of the components for the path:

1. Call A$GET$PATH$COMPONENT to obtain the file name C.
2. Call A$ATTACH$FILE with the prefix parameter equal to the token for file C and the subpath equal to a circumflex (^). This call will return a token for a connection to directory file B.
3. After calling GET$TYPE to verify that the token is indeed for a connection, call A$GET$PATH$COMPONENT to obtain the file name B.
4. Call A$ATTACH$FILE with the prefix parameter equal to the token for file B and the subpath equal to a circumflex (^). This call will return a token for a connection to directory file A.
5. After calling GET$TYPE to verify that the token is indeed for a connection, call A$GET$PATH$COMPONENT to obtain the file name A.
6. Call A$ATTACH$FILE with the prefix parameter equal to the token for file A and the subpath equal to a circumflex (^). This call will return a token for a connection to the root of the file tree.
7. After calling GET$TYPE to verify that the token is indeed for a connection, call A$GET$PATH$COMPONENT again. This time, the null string will be returned, and this tells you that you now have all of the components of the desired path name.

## Condition Codes

A$GET$PATH$COMPONENT returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide*.

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

## Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | At least one of the following is true:<br>• One or more of the following parameters is not a token for an existing object:<br>- The connection parameter<br>- The resp$mbox parameter<br>• The connection is being deleted. |
| E$LIMIT | 0004H | The calling task's job has already reached its object limit. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete this call. |
| E$NOT$CON-<br>FIGURED | 0008H | This system call is not part of the present configuration. |
| E$NOT$FILE$CONN | 0032H | For remote files, the connection parameter must be a file connection, not a device connection. |
| E$SUPPORT | 0023H | The specified connection was not created by this job. |
| E$TYPE | 8002H | At least one of the following is true:<br>• The connection parameter is a token for an object that is not a connection.<br>• The resp$mbox parameter is a token for an object that is not a mailbox. |

## Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$FNEXIST | 0021H | The file is marked for deletion. (In this case, the string is undefined.) |
| E$INVALID$FNODE | 003DH | The fnode for the specified file is invalid. The file cannot be accessed. |

| | | |
|---|---|---|
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. |
| E$IO$MEM | 0042H | The memory available to the Basic I/O System job is not sufficient to complete the call. |

# A$OPEN

A$OPEN opens an asynchronous file connection for I/O operations.

```
CALL RQ$A$OPEN(connection, mode, share, resp$mbox, except$ptr);
```

## Input Parameters

connection — A TOKEN for the connection to be opened.

mode — A BYTE giving the mode desired for the open connection; possible values are:

1 Open for reading
2 Open for writing
3 Open for both reading and writing

Remote directories must be opened with the mode parameter set to 1.

share — A BYTE specifying the kind of sharing desired for the file to which you are opening a connection; possible values are:

0 Private use only
1 Share with readers only
2 Share with writers only
3 Share with all users

## Output Parameters

resp$mbox — A TOKEN for the mailbox that receives an I/O request/result segment (IORS) indicating the result of the call (for details on the IORS, see Appendix A). A value of SELECTOR$OF(NIL) means that you do not want to receive an IORS.

If it receives an IORS, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

## Description

The A$OPEN system call opens a connection for I/O operations. The connection must be opened before reading, writing, and seeking can be performed on the associated file.

A$OPEN also initializes the file pointer to byte-position zero. Subsequent Basic I/O System calls (A$SEEK, A$READ, and A$WRITE) will move this pointer.

The mode and share parameters are compared to the current sharing status of the file (which may have been set by a previous A$OPEN system call). If they are not compatible, an E$SHARE exceptional condition is returned. No deadlock occurs, however, because open calls are not queued. The system does not notify callers when the sharing status of the file changes. If such notification is important, users of the file should arrange a suitable protocol.

If the file is attached by multiple connections, the file might be open for reading by some connections and open for writing by others at the same time. Any modification of the file by a writer will be seen by readers that subsequently read the modified part of the file.

### NOTE

Directory files can be opened and read, but only by specifying a one (read) for the mode parameter and a three (share all) for the share parameter. Any other combination will return an error.

## Special Considerations for iRMX®-NET

Unlike a local named file, the access rights of a remote named file are not checked when a connection to the file is created. Instead, the remote named file's access rights are checked during operations on the connection.

The above discrepancy won't affect your programs if you do the following:

- Open, delete, and rename files prior to changing their access lists.
- Establish connections to files after changing their access lists.

## Condition Codes

A$OPEN returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide*.

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

### Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | At least one of the following is true:<br>• One or more of the following parameters is not a token for an existing object:<br>- The connection parameter<br>- The resp$mbox parameter<br>• The connection is being deleted.<br>• The connection for a remote driver is no longer active. |
| E$LIMIT | 0004H | At least one of the following is true:<br>• The calling task's job has already reached its object limit.<br>• The number of outstanding I/O operations for a remote connection has been exceeded. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete the call. |
| E$NOT$CON-<br>FIGURED | 0008H | This system call is not part of the present configuration. |
| E$PARAM | 8004H | The mode or share parameter has an invalid value (out of range 1-3 or 0-3, respectively.) |
| E$SUPPORT | 0023H | The specified connection was not created by this job. |
| E$TYPE | 8002H | At least one of the following is true:<br>• The connection parameter is a token for an object that is not a connection.<br>• The resp$mbox parameter is a token for an object that is not a mailbox. |

## Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$CONN$OPEN | 0035H | The connection is a file or directory connection that is already open. |
| E$NOT$FILE$CONN | 0032H | The connection is a device connection, not a file connection. |
| E$SHARE | 0028H | At least one of the following is true:<br>• The file's sharing attribute currently is not compatible with the mode or the share parameter in this call.<br>• This call is attempting to open a directory for some operation other than "read" (mode parameter) or "share with all users" (share parameter). (See Description for more information on sharing files.) |
| E$FACCESS | 0026H | The connection does not have access compatible with the mode specified in this call. |
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the IORS for more information. |
| E$LIMIT | 0004H | Processing this call would deplete the remote server's resources. For a list of remote server resources, refer to the *iRMX® Networking Software User's Guide.* |
| E$FTYPE | 0027H | The requested operation is not valid for this file type. |

The A$PHYSICAL$ATTACH$DEVICE system call attaches a device to the Basic I/O System.

## CAUTION

**Any task that uses this system call loses its device independence. To maintain as much device independence as possible in your application, a few selected tasks should perform all attaching and detaching of devices, passing tokens for the devices to other tasks as necessary.**

**Also, if a task uses this system call to attach devices, the devices are automatically detached (and connections to files on the device are automatically deleted) when the containing job is deleted. If you want to prevent the device from being detached in these circumstances, use the Extended I/O System's LOGICAL$ATTACH$DEVICE system call instead.**

```
CALL RQ$A$PHYSICAL$ATTACH$DEVICE(dev$name$ptr, file$driver,
                                  resp$mbox, except$ptr);
```

## Input Parameters

dev$name$ptr A POINTER to a STRING containing the name (as specified during configuration) of the device to be attached. The maximum string length is 14 characters. If you specify a longer string, the Basic I/O System truncates the string to 14 characters. To prevent possible duplication of names, do not specify a string longer than 14 characters.

For remote devices, specify the name of the server to be attached.

file$driver A BYTE specifying which file driver is to supply the connection to the device. Possible values are as follows:

| Value | File Driver |
|---|---|
| 1 | Physical |
| 2 | Stream |
| 4 | Named |
| 5 | Remote |

resp$mbox — A TOKEN for the mailbox that receives the result object of the call. This result object is a new connection if the call is successful, or an I/O request/result segment (IORS) otherwise (for details on the IORS, see Appendix A). To ascertain the type of object returned, use the Nucleus system call GET$TYPE.

If the object received is an IORS, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

## Output Parameter

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

## Description

A$PHYSICAL$ATTACH$DEVICE returns a device connection to the device specified by the dev$name$ptr parameter. The file$driver parameter specifies the kind of files (physical, stream, named, or remote) that the device will create when the returned device connection is used in subsequent calls to A$CREATE$FILE.

The device connection object is returned to the response mailbox if the call is successful; otherwise an IORS is returned to the response mailbox. The returned connection object can be used as a prefix in other system calls. It can be deleted only by calling A$PHYSICAL$DETACH$DEVICE.

In the case of a connection to a disk device, where the file$driver parameter specifies named files for the device, the connection is actually to a volume mounted on the disk hardware. Such volumes must be properly formatted. If they are not, an E$ILLVOL exceptional condition is returned. Refer to the *Operator's Guide To The iRMX® Human Interface* for information about formatting disks.

## Condition Codes

A$PHYSICAL$ATTACH$DEVICE can return condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide*. The following list is divided into two parts--one for sequential codes and one for concurrent codes.

## Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | The resp$mbox parameter is not a token for an existing object. |
| E$LIMIT | 0004H | Processing this call would cause one or more of the following limits to be exceeded:<br>• The object limit for this job.<br>• The number (255 decimal) of I/O operations that can be outstanding at one time for the caller's job. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete the call. |
| E$NOT$CON-<br>FIGURED | 0008H | This system call is not part of the present configuration. |
| E$PARAM | 8004H | At least one of the following is true:<br>• The number representing the file driver is not valid.<br>• A value of SELECTOR$OF(NIL) was specified for the response mailbox. |
| E$TYPE | 8002H | The resp$mbox parameter in the call is a token for an object that is not a mailbox. |

## Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$ALREADY$-<br>ATTACHED | 0038H | The specified device is already attached. |
| E$DEVFD | 0022H | The specified device is not compatible with the specified file driver. |
| E$FNEXIST | 0021H | The device specified by the device$name parameter does not exist. |

| | | |
|---|---|---|
| E$ILLVOL | 002DH | At least one of the following is true:<br>• The specified device is a disk volume not properly formatted for use with the named file driver. The volume being attached must have a valid iRMX volume label with FD$NAMED in the FILE$DRIVER field.<br>• The device could not be attached because the fnode for the root directory of the device is invalid. |
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the IORS for more information. |
| E$IO$MEM | 0042H | The memory available to the Basic I/O System job is not sufficient to complete the call. |
| E$LIMIT | 0004H | Processing this call would deplete the remote server's resources. For a list of remote server resources, refer to the *iRMX® Networking Software User's Guide*. |
| E$PROTOCOL | 02E9H | The iNA 960 version on the local system does not have the iNA R1.0 to R3.0 compatibility code and the server to be attached has iNA R1.0 loaded. |

# A$PHYSICAL$DETACH$DEVICE

The A$PHYSICAL$DETACH$DEVICE system call detaches a device from the Basic I/O System.

## CAUTION

**Any task that uses this system call loses its device independence. To maintain as much device independence as possible in your application, a few selected tasks should perform all attaching and detaching of devices, passing tokens for the devices to other tasks as necessary.**

```
        CALL RQ$A$PHYSICAL$DETACH$DEVICE(connection, hard, resp$mbox,
                                         except$ptr);
```

## Input Parameters

| | |
|---|---|
| connection | A TOKEN for the connection object for the device that is to be detached. |
| hard | A BYTE containing a value that specifies whether (0FFH) or not (0) a hard detach of the device is desired. |
| resp$mbox | A TOKEN for the mailbox to which the IORS is sent when the operation has finished (for details on the IORS, see Appendix A). A value of SELECTOR$OF(NIL) indicates that no response is desired. |

## Output Parameter

| | |
|---|---|
| except$ptr | A POINTER to a WORD where the sequential condition code will be returned. |

## Description

The A$PHYSICAL$DETACH$DEVICE system call breaks connections established by calls to A$PHYSICAL$ATTACH$DEVICE. It also deletes the file connection objects associated with those device connections. Devices that are detached in this manner must be reattached before any files on the device can be attached or reattached.

When detaching a device, you can choose to detach all attached files on the device. A hard detach deletes the connection objects for all such files on the device. To specify a hard detach, assign the value 0FFH to the hard parameter.

If you choose not to request a hard detach, there must not be any attached files on the device. To specify that you do not want a hard detach, assign the value 0 to the hard parameter.

Note that, whether you specify a hard detach or not, there will be no attached files on the device after the device is detached.

## Condition Codes

A$PHYSICAL$DETACH$DEVICE can return condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide*.

The following list is divided into two parts--one for sequential codes and one for concurrent codes.

### Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | One or more of the following parameters is not a token for an existing object:<br>• The connection parameter<br>• The resp$mbox parameter<br>• The connection for a remote driver is no longer active. |
| E$LIMIT | 0004H | The calling task's job has already reached its object limit. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete the call. |
| E$NOT$CON-FIGURED | 0008H | This system call is not part of the present configuration. |
| E$NOT$DEVICE$-CONN | 0033H | The specified connection parameter is not a device connection. |
| E$SUPPORT | 0023H | The specified connection was not created by this job. |

| | | |
|---|---|---|
| E$TYPE | 8002H | At least one of the following is true:<br>• The connection parameter is a token for an object that is not a connection.<br>• The resp$mbox parameter is a token for an object that is not a mailbox. |

### Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$FNEXIST | 0021H | The device specified by the connection parameter is already being detached. |
| E$IO | 002BH | An I/O error occurred during the operation, but the operation was successful anyway. |
| E$OUTSTANDING$-CONNS | 0037H | The call attempted a soft detach, but connections to the device still existed. |

A$READ reads the requested number of bytes, starting with the current position of the pointer for the specified file connection.

```
CALL RQ$A$READ(connection, buff$ptr, count, resp$mbox, except$ptr);
```

## Input Parameters

| | |
|---|---|
| connection | A TOKEN for the open file connection to be read. |
| buff$ptr | A POINTER to the buffer that receives the data. |
| count | A WORD giving the number of bytes to be read. |

## Output Parameters

resp$mbox

A TOKEN for the mailbox that receives an I/O request/result segment (IORS) indicating the result of the call. (For details on the IORS, see Appendix A.) A value of SELECTOR$OF(NIL) means that you do not want to receive an IORS (you do not want to check the status of the READ operation).

If your task receives an IORS, it should call DELETE$SEGMENT to delete the segment after examining it.

If you use the (Basic I/O SYSTEM) RQ$WAIT$IO system call to check the concurrent condition code after A$READ executes, the IORS is deleted for you.

The number of bytes read is in the "actual" field of the IORS. If a read operation is requested with the file pointer set at or beyond the end of the file, an actual value of zero is returned.

If all the connections to a stream file are requesting read operations, an actual value of zero is returned along with an E$FLUSHING condition code.

except$ptr

A POINTER to a WORD where the sequential condition code will be returned.

## Description

The A$READ system call initiates a read operation on an open connection. The data is read as a string of bytes, starting at the current location of the connection's file pointer. Any number of bytes can be requested. Some efficiency may be gained by starting reads on device block boundaries. After the read operation is finished, the file pointer points just past the last byte read.

The buffer specified by the "buff$ptr" parameter can be in a segment allocated by the Nucleus, but this is not a requirement.

### NOTE

A call to A$READ will not be successful unless the mode of the open connection permits reading (see A$OPEN).

## Special Considerations for iRMX®-NET

iRMX-NET's Remote File Driver does not perform fragmentation and reassembly. For optimal performance, reading and writing should begin at offsets that are integral multiples of the remote server's buffer size. The device$gran parameter returned by the A$GET$FILE$STATUS system call indicates the buffer size of a remote server.

## Condition Codes

A$READ returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide.*

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

## Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$BAD$BUFF | 8023H | **This condition code is returned only in the iRMX II Operating System.**<br>At least one of the following is true:<br>• The target memory buffer is not a writeable segment.<br>• The target memory buffer crosses a segment boundary. |
| E$BUFFERED$CONN | 0036H | The connection parameter you supplied was opened with an Extended I/O System call. You cannot use it with the A$READ system call. |
| E$EXIST | 0006H | At least one of the following is true:<br>• One or more of the following parameters is not a token for an existing object:<br>- The connection parameter<br>- The resp$mbox parameter<br>• The connection is being deleted. |
| E$LIMIT | 0004H | The calling task's job has already reached its object limit. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete the call. |
| E$NOT$CON-<br>FIGURED | 0008H | This system call is not part of the present configuration. |
| E$SUPPORT | 0023H | The specified connection was not created by this job. |
| E$TYPE | 8002H | At least one of the following is true:<br>• The connection parameter is a token for an object that is not a connection.<br>• The resp$mbox parameter is a token for an object that is not a mailbox. |

### Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$CONN$NOT$OPEN | 0034H | This connection is not open for reading or updating. |
| E$FLUSHING | 002CH | At least one of the following is true:<br>• The specified connection was closed before the read operation was completed.<br>• The file is a stream file and all other connections to the file are also attempting to read the file. |
| E$IDDR | 002AH | This request is invalid for the device driver. For example, it is not valid to use this call with a line printer. |
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the IORS for more information. |

A$RENAME$FILE changes the path name of a named data or directory file.

```
CALL RQ$A$RENAME$FILE(connection, user, prefix, subpath$ptr,
                      resp$mbox, except$ptr);
```

## Input Parameters

connection — A TOKEN for a connection to the file being renamed. This connection and all other connections to the file will remain in effect after the file is renamed.

user — A TOKEN for the user object to be inspected in access checking. A SELECTOR$OF(NIL) specifies the default user for the job.

prefix — A TOKEN for the connection to be used as the starting point in a path scan. A SELECTOR$OF(NIL) specifies the default prefix for the job.

subpath$ptr — A POINTER to a STRING containing the new subpath for the file. Prefix and subpath must not lead to an already-existing file. The string pointed to by the subpath parameter cannot be a null string.

## Output Parameters

resp$mbox — A TOKEN for the mailbox that receives an I/O request/result segment (IORS) indicating the result of the call. (For details on the IORS, see Appendix A.) A value of SELECTOR$OF(NIL) means that you do not want to receive an IORS.

If it receives an IORS, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

## Description

The A$RENAME$FILE system call applies to named files only. It is called to change the path name of a file. For named data or directory files, A$RENAME$FILE can be used to recatalog files in different parent directories, as long as the new directory is on the same volume as the file's original parent directory.

There is one restriction concerning the manner in which a directory can be renamed. Any attempt to rename a directory as its own parent causes the Basic I/O System to return an exception code. Also, be aware that renaming a directory changes the paths of any files contained in the directory.

### NOTE

In order to rename a file, the caller must have delete access to the file and must have add-entry access to the file's parent directory.

## Special Considerations for iRMX®-NET

The A$RENAME$FILE system call cannot rename the following files and directories on a remote server:

- a virtual root directory
- a file in a virtual root directory
- a public directory

If an attempt is made to rename any of these files and directories, an E$FACCESS exceptional condition is returned.

Also, unlike a local named file, the access rights of a remote named file are not checked when a connection to the file is created. Instead, the remote named file's access rights are checked during operations on the connection.

The above discrepancy won't affect your programs if you do the following:

- Open, delete, and rename files prior to changing their access lists.
- Establish connections to files after changing their access lists.

## Condition Codes

A$RENAME$FILE returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide.*

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

## Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

E$OK 0000H No exceptional conditions.

E$DEV$OFF$LINE 002EH The prefix parameter in this system call refers to a logical connection. One of the following is true of the device that is associated with the connection:

- It has been physically attached but is now offline.
- It has never been physically attached. (See the *iRMX® Device Drivers User's Guide* for a more detailed explanation.)

E$EXIST 0006H At least one of the following is true:

- One or more of the following parameters is not a token for an existing object:
    - The connection parameter
    - The user parameter
    - The prefix parameter
    - The resp$mbox parameter
- One or more of the following is being deleted:
    - The connection specified by the prefix.
    - The connection specified by the connection parameter.
- The connection for a remote driver is no longer active.

E$IFDR 002FH This system call applies only to named files, but the connection parameter specifies some other type of file.

E$LIMIT 0004H Processing this call would cause one or more of the following limits to be exceeded:

- The object limit for this job.
- The number of I/O operations that can be outstanding at one time for the user object specified in the call (255 decimal).
- The number of outstanding I/O operations for a remote connection has been exceeded.

| | | |
|---|---|---|
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete the call. |
| E$NOPREFIX | 8022H | The call specified a default prefix (prefix argument equals SELECTOR$OF(NIL)), but no default prefix can be found because of one of the following:<br>• When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default prefix.<br>• The job's directory can have entries but a default prefix is not cataloged there. |
| E$NOUSER | 8021H | If the user parameter in this call is not SELECTOR$OF(NIL) then the parameter is not a user object.<br>If the user parameter is SELECTOR$OF(NIL), it specifies a default user object, but no default user object can be found because of one of the following:<br>• When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default user object.<br>• The job's directory can have entries but a default user object is not cataloged there.<br>• The object cataloged with the name R?IOUSER is not a user object. The name R?IOUSER should be treated as a reserved word. |
| E$NOT$CONFIGURED | 0008H | This system call is not part of the present configuration. |
| E$NOT$SAME$DEVICE | 003AH | The connection and the prefix in the call refer to different devices. You cannot simultaneously rename a file and move it to another device. |
| E$PATHNAME$SYNTAX | 003EH | One or more of the following conditions caused this exception:<br>• The specified path name contains invalid characters or has a length of zero. The path name can include any printable ASCII character except the slash (/), up-arrow (↑), and circumflex (^).<br>• The subpath of the specified remote file exceeds 127 bytes in length. |

| | | |
|---|---|---|
| E$SUPPORT | 0023H | The specified connection was not created by this job. |
| E$TYPE | 8002H | At least one of the following is true:<br>• The connection parameter is a token for an object that is not a connection object.<br>• The prefix parameter is a token for an object that is not of the correct type. It must be either a connection object or a logical device object (Logical device objects are created by the Extended I/O System.)<br>• The resp$mbox parameter in the call is a token for an object that is not a mailbox. |

### Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$DEV$DETACHING | 0039H | The file specified is on a device that the system is detaching. |
| E$FACCESS | 0026H | At least one of the following is true:<br>• The specified file does not have "add entry" access to the parent directory.<br>• The specified connection does not have "delete" access to the file.<br>• The call is attempting to rename the root directory or a bit-map file. |
| E$FEXIST | 0020H | A file with the specified path name already exists. |
| E$FNEXIST | 0021H | A file in the specified path does not exist or is marked for deletion. |
| E$FTYPE | 0027H | The string pointed to by the subpath$ptr parameter contains a file that should be the name of a directory, but is not. (Except for the last file, each file listed in a pathname must be a named directory.) |
| E$ILLOGICAL$-<br>RENAME | 003BH | The call is attempting to rename the directory to a new path containing itself. |
| E$INVALID$FNODE | 003DH | The fnode for the specified file (or for a directory in the file's path) is invalid. The file with the invalid fnode cannot be accessed. |

| | | |
|---|---|---|
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the IORS for more information. |
| E$IO$MEM | 0042H | The memory available to the Basic I/O System job is not sufficient to complete the call. |
| E$LIMIT | 0004H | Processing this call would deplete the remote server's resources. For a list of remote server resources, refer to the *iRMX® Networking Software User's Guide*. |
| E$NAME$NEXIST | 0049H | The user object does not represent a verified user or the user object is not properly defined in the remote server's User Definition File (UDF). |
| E$NOT$FILE$CONN | 0032H | The subpath$ptr parameter is a null pointer and the prefix parameter is not a file connection. |
| E$PASSWORD$-MISMATCH | 004BH | The password of the user object does not match the password of the corresponding user defined on the remote server. |
| E$PATHNAME$-SYNTAX | 003EH | The syntax of the specified remote file path name is illegal. Path names for remote files must follow the naming conventions of the server. |
| E$UDF$IO | 02D0H | An error occurred while accessing the remote server's User Definition File (UDF). |
| E$SPACE | 0029H | At least one of the following is true:<br>• The volume is full.<br>• No more files can be created on the remote server's volume. The Remote File Driver cannot distinguish between an E$FNODE$LIMIT and an E$SPACE condition code. |
| E$SUPPORT | 0023H | As configured, the Basic I/O System does not allow allocation of space on volumes. |

A$SEEK moves the file pointer of an open connection.

```
CALL RQ$A$SEEK(connection, mode, move$size, resp$mbox, except$ptr);
```

## Input Parameters

connection — A TOKEN for the open file connection whose file pointer is to be moved.

mode — A BYTE describing the movement of the file pointer. Possible values are:

1 Move pointer back by move$size bytes; if this action moves the pointer past the beginning of the file, the pointer is set to zero (first byte).

2 Set the pointer to the location specified by move$size.

3 Move the file pointer forward by move$size bytes.

4 Move the pointer to the end of the file, minus move$size bytes.

move$size — A DWORD giving the number of bytes involved in the seek. The interpretation of move$size depends on the mode setting, as just explained.

## Output Parameters

resp$mbox — A TOKEN for the mailbox that receives an I/O request/result segment (IORS) indicating the result of the call. (For details on the IORS, see Appendix A.) A value of SELECTOR$OF(NIL) means that you do not want to receive an IORS (you do not want to check the status of the SEEK operation).

If your task receives an IORS, it should call DELETE$SEGMENT to delete the segment after examining it. If you use the RQ$WAIT$IO BIOS system call to check the concurrent condition code after A$SEEK executes, the IORS is deleted for you.

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

## Description

The A$SEEK system call applies to physical and named files only. This call moves the file pointer for an open connection, allowing file contents to be accessed randomly. The file pointer can be moved to any byte position in the file; the first byte is byte zero.

It is legitimate to position the file pointer beyond the end-of-file for a named file. If your task does this and then invokes the A$READ system call, the Basic I/O System behaves as though the reading operation began at the end-of-file.

Also, it is possible to invoke the A$WRITE system call with the file pointer beyond the end of the file. If your task does this, the Basic I/O System attempts to expand the file. If the Basic I/O System does expand your file in this manner, the file contains random information between the old end-of-file and the point in the file where the write begins.

## Condition Codes

A$SEEK returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide.*

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

### Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$BUFFERED$CONN | 0036H | The connection parameter was produced by the Extended I/O System. You cannot use this parameter with Basic I/O System calls. |
| E$EXIST | 0006H | At least one of the following is true:<br>• One or more of the following parameters is not a token for an existing object:<br>- The connection parameter<br>- The resp$mbox parameter<br>• The connection is being deleted.<br>• The connection for a remote driver is no longer active. |

| | | |
|---|---|---|
| E$IFDR | 002FH | This system call applies only to named and physical files, but the connection is to a stream file. |
| E$LIMIT | 0004H | At least one of the following is true:<br>• The calling task's job has already reached its object limit.<br>• The number of outstanding I/O operations for a remote connection has been exceeded. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete the call. |
| E$NOT$CON-FIGURED | 0008H | This system call is not part of the present configuration. |
| E$PARAM | 8004H | The mode parameter value is out of the valid range (1 to 4). |
| E$SUPPORT | 0023H | The specified connection was not created by this job. |
| E$TYPE | 8002H | At least one of the following is true:<br>• The connection parameter is a token for an object that is not a connection.<br>• The resp$mbox parameter is a token for an object that is not a mailbox. |

### Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$CONN$NOT$OPEN | 0034H | The connection is not open. |
| E$FLUSHING | 002CH | The specified connection was closed before the seek operation could complete. |
| E$IDDR | 002AH | This request is invalid for the device driver. For example, it is not valid to use this call with a line printer. |
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the IORS for more information. |
| E$PARAM | 8004H | This call attempted to seek beyond the end of the physical device. This applies only to physical files. |

# A$SET$EXTENSION$DATA

The A$SET$EXTENSION$DATA system call writes the extension data for a Basic I/O System file.

```
        CALL RQ$A$SET$EXTENSION$DATA(connection, data$ptr, resp$mbox,
                                      except$ptr);
```

## Input Parameters

connection — A TOKEN for a connection to a file whose extension data is to be set.

data$ptr — A POINTER to a structure of the following form:

```
DECLARE ext$data$seg  STRUCTURE(
                         count   BYTE,
                         info(*) BYTE);
```

where

count — Number (up to 255) of bytes of extension data being written. For remote files, this field must be set to 0.

info(*) — The extension data.

resp$mbox — A TOKEN for the mailbox that receives an I/O request/result segment (IORS) indicating the result of the call. (For details on the IORS, see Appendix A.) A value of SELECTOR$OF(NIL) means that you do not want to receive an IORS.

If it receives an IORS, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

## Output Parameter

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

## Description

Associated with each file created through the Basic I/O System is a file descriptor containing information about the file. Some of that information is used by the Basic I/O System and can be accessed by tasks through the A$GET$FILE$STATUS system call. Up to 255 additional bytes of the file descriptor, known as extension data, are available for use by Operating System extensions, depending upon how the volumes were formatted. For named volumes, the first three bytes of this extension data is reserved for use by the Basic I/O System. OS extensions can write extension data by using A$SET$EXTENSION$DATA and they can read extension data by using A$GET$EXTENSION$DATA. The maximum number of bytes of extension data may be less than 255 since the limit is specified when the secondary storage devices are formatted.

After the new extension data is set, an IORS returns to the response mailbox.

A$SET$EXTENSION$DATA can only be applied to asynchronous connections created via the named file driver.

## Condition Codes

A$SET$EXTENSION$DATA returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide.*

The following list is divided into two parts--one for sequential codes and one for concurrent codes.

### Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | At least one of the following is true:<br>• One or more of the following parameters is not a token for an existing object:<br>- The connection parameter.<br>- The resp$mbox parameter.<br>• The connection is being deleted. |
| E$IFDR | 002FH | This system call applies only to named files, but the connection parameter specifies another type of file. |

| | | |
|---|---|---|
| E$LIMIT | 0004H | The calling task's job has already reached its object limit. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete the call. |
| E$NOT$CON-FIGURED | 0008H | This call is not part of the present configuration. |
| E$SUPPORT | 0023H | The specified connection was not created by this job. |
| E$TYPE | 8002H | At least one of the following is true:<br>• The connection parameter is a token for an object that is not a connection.<br>• The resp$mbox parameter is a token for an object that is not a mailbox. |

### Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the IORS for more information. |
| E$PARAM | 8004H | At least one of the following is true:<br>• The count field in the ext$data$seg data structure contains a value greater than the value specified when the disk was formatted.<br>• The connection parameter references a remote file and the count field does not contain a value of 0. |

A$SPECIAL enables tasks to perform a variety of special functions.

```
CALL RQ$A$SPECIAL(connection, spec$func, ioparm$ptr, resp$mbox,
                  except$ptr);
```

## Input Parameters

connection — A TOKEN for a connection to the file or device for which the special function is to be performed. To access a remote server, this parameter must be a connection to the server's virtual root directory.

spec$func — An encoded WORD that, with the connection argument, specifies the function being requested. Only function value 2 (Notify) is supported for remote servers. The functions are described under the heading Description and are summarized as follows:

| File driver<br>for connection | Spec$func<br>value | Function |
|---|---|---|
| Physical | 0 | Format track |
| Stream | 0 | Query |
| Stream | 1 | Satisfy |
| Physical or Named | 2 | Notify |
| Physical | 3 | Get disk/tape data |
| Physical | 4 | Get terminal data |
| Physical | 5 | Set terminal data |
| Physical | 6 | Set signal |
| Physical | 7 | Rewind tape |
| Physical | 8 | Read tape file mark |
| Physical | 9 | Write tape file mark |
| Physical | 10 | Retension tape |
| Physical | 11 | Set Character Font |
| Physical | 12 | Set bad track/sector information |
| Physical | 13 | Get bad track/sector information |
|  | 14, 15 | Reserved |
| Physical | 16 | Get terminal status |
| Physical | 17 | Cancel terminal I/O |
| Physical | 18 | Resume terminal I/O |
|  | 19-32767 | Reserved for other Intel products |
|  | 32768-65555 | Reserved for user devices |

ioparm$ptr — A POINTER to a parameter block. The contents of the parameter block depends upon the requirements of the special function being requested and are described fully under the heading Description. Enter a NIL value if the special function you request does not require a parameter block.

## Output Parameters

resp$mbox — A TOKEN for the mailbox that receives an I/O request/result segment (IORS) indicating the result of the call. (For details on the IORS, see Appendix A.) A value of SELECTOR$OF(NIL) means that you do not want to receive an IORS.

If it receives an IORS, the calling task should call DELETE$SEGMENT to delete the segment.

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

## Description

The A$SPECIAL system call enables tasks to perform a variety of special functions.

Tasks define their requests by means of the spec$func and ioparm$ptr parameters. Spec$func is a code which, when combined with the file driver associated with the connection argument, specifies the function the Basic I/O System is to perform. When more information is needed to define a request, ioparm$ptr points to a parameter block containing the additional data. Descriptions of the available functions follow.

## Special Considerations for iRMX®-NET

iRMX-NET only supports function value 2 (Notify) for remote servers. The calling task is notified of a communication failure immediately after an unsuccessful attempt to access a remote file or if the device connection to the remote server is physically detached. Communication failures can result from resetting the server, faults in the consumer or server, or line transmission errors. The Remote File Driver returns an E$IO status and an IO$PRINT unit status to requests that attempt to access a file on an unavailable remote server.

To restore the availability of a remote server, perform the following steps:

1. Fix the communication problem.
2. Call A$PHYSICAL$DETACH$DEVICE to detach the server's device connection.
3. Call A$PHYSICAL$ATTACH$DEVICE to reattach the server.

### Formatting a Track (Function Code 0)

This function applies to physical files only. To format a track on a mass storage device, call A$SPECIAL with an open file connection, with spec$func equal to 0, and with ioparm$ptr pointing to a structure of the form:

```
DECLARE format$track STRUCTURE(
            track$number    WORD,
            interleave      WORD,
            track$offset    WORD,
            fill$char       WORD);
```

In this structure, the fields are defined as follows:

track$number — The number of the track to be formatted. Acceptable values are 0 to one less than the number of tracks on the volume. Other values cause an E$SPACE exceptional condition. When formatting a RAM-disk or a tape, you must place a zero value in this field.

interleave — The interleave factor for the track. (That is, the number of physical sectors to advance when locating the next logical sector.) An interleave factor of zero or one skips no physical sectors between logical sectors. If the specified interleave factor is greater than the number of physical sectors on a track, the Operating System divides the specified value by the number of physical sectors and uses the remainder as the actual interleave value.

This field does not apply to tapes.

track$offset — The number of physical sectors to advance when locating the first logical sector. This field does not apply to tapes.

fill$char — A byte value with which each sector is to be filled. Some drivers ignore this value and fill the sector with a character they establish.

### Obtaining Information On Stream File Operations (Function Code 0)

Occasionally, a task using a stream file needs to know what is being requested by the other task using the same stream file. For example, the task doing a read operation on a stream file might need to know how many bytes are being sent by the task doing a write operation on the same file. Tasks can obtain this kind of information by calling A$SPECIAL, using the connection for the stream file, with spec$func set to 0 (query). The ioparm$ptr argument is ignored.

If a read or write request is queued at the file, the information requested is returned in the IORS for the call to A$SPECIAL. In the IORS, the ACTUAL field contains the number of bytes being sent, the COUNT field contains the number of bytes still remaining in the buffer, and the BUFF$P field points to the buffer. For details on the IORS fields, refer to the *iRMX® Device Driver User's Guide.*

If no read or write request is queued at the file, the calling task's request for information is queued at the file. If a second request for information is made before the first one is satisfied, the IORSs for both requests are returned with E$STREAM$SPECIAL in the status field.

### Satisfying Stream File Transactions (Function Code 1)

When a task tries to read or write to a stream file, the request is not satisfied until the other task makes a request that matches the first request.

For example, if task A wants to read 512 bytes, but task B only wants to write 256 bytes, only 256 bytes are transferred. Task A continues to wait for the other 256 bytes, even though Task B may never write them.

By using A$SPECIAL, with a stream file connection and with spec$func set to 1 (ioparm$ptr is ignored), either task can force the data transfer request to be satisfied, even though the reading task is requesting more bytes than the writing task is providing. After the transfer, the tasks can ascertain the number of bytes sent by checking the actual field in their respective IORSs. A task trying to satisfy an I/O request in this way will receive an E$STREAM$SPECIAL exceptional condition if no request is queued at the stream file or if a request for information is queued. In the latter case, the task that submitted the request for information also receives an E$STREAM$SPECIAL condition.

### Requesting Notification that a Volume is Unavailable (Function Code 2)

This function applies to named and physical files only. When a person opens a door to a flexible disk drive or presses the online/offline button on other mass storage drives, the volume mounted on that drive becomes unavailable. A task can request notification of such an event by calling A$SPECIAL. For some 5-1/4" flexible disk drives, notification occurs when the Basic I/O System first tries to perform an operation on the unavailable volume. For most other drives, notification occurs immediately. The reason for this difference is that controller/drive combinations that include some 5-1/4" drives cannot generate an interrupt when the drive ceases to be ready. In contrast, most other controller/drive combinations do.

On those drives where no notification occurs until the Basic I/O System attempts to access the drive, a dangerous situation occurs whenever you change a volume without first detaching the device. If you do not first detach the device and then reattach it, the Basic I/O System accesses the device using the directory information from the old volume. Unless the new volume is write-protected, this process corrupts the entire volume, rendering it useless. The correct sequence of events when changing volumes on one of these devices is as follows:

1. Detach the unit (via A$PHYSICAL$DETACH$DEVICE).
2. Remove the old volume.
3. Install the new volume.
4. Reattach the unit (via A$PHYSICAL$ATTACH$DEVICE).

For devices that can perform notification, a task requests notification by calling A$SPECIAL with a token for a device connection, with spec$func set to 2, and with ioparm$ptr pointing to a structure of the form:

```
DECLARE notify STRUCTURE(
                 mailbox      TOKEN,
                 object       TOKEN);
```

where

mailbox — A TOKEN for a mailbox.

object — A TOKEN for an object. When the Basic I/O System detects that the implied volume is unavailable, the object is sent to the mailbox.

After a task has made a request for notification, the Basic I/O System remembers the object and mailbox tokens until either the volume is detected as being unavailable or until the device is detached by the A$PHYSICAL$DETACH$DEVICE system call. When the volume becomes unavailable, the object is sent to the mailbox. Note that this implies that some task should be dedicated to waiting at the mailbox.

If the volume is detected as being unavailable, the Basic I/O System will not execute I/O requests to the device on which the volume was mounted. Such requests are returned with the status field of the IORS set to E$IO and the unit$status field set to IO$OPRINT (value = 3). The latter code means that operator intervention is required.

If any task issues a subsequent notification request for the same device connection, the Basic I/O System replaces the old mailbox and object values with the new ones specified. It does not return an exception code.

To restore the availability of a volume, perform the following steps:

1. Close the door of the diskette drive or restart the hard disk drive.
2. Call A$PHYSICAL$DETACH$DEVICE. It may be necessary to do a "hard" detach of the device.
3. Call A$PHYSICAL$ATTACH$DEVICE and reattach the device.
4. Create a new file connection.

To cancel a request for notification, make a dummy request using the same connection with a SELECTOR$OF(NIL) value in the mailbox parameter.

### Obtaining Information About Winchesters and Certain Other Disks (Function Code 3)

This function applies only to physical files. If your device is a Winchester drive with an iSBC® 214/215G/221 disk controller or a drive with an iSBC 220 SMD controller, you can obtain specification information about it by calling A$SPECIAL with a token for a device connection, with spec$func set to 3, and with ioparm$ptr pointing to a structure of the form:

```
DECLARE disk$drive$data STRUCTURE(
                cylinders   WORD,
                fixed       BYTE,
                removable   BYTE,
                sectors     BYTE,
                sector$size WORD,
                alternates  BYTE);
```

A$SPECIAL returns information to the fields of this structure, as follows:

cylinders — The total number of cylinders on the drive.

fixed — The number of heads on the fixed disk or Winchester disk.

removable — The number of heads on the removable disk cartridge.

sectors — The number of sectors in a track.

sector$size — The number of bytes in a sector.

alternates — The number of alternate cylinders on the drive.

### Obtaining Information About Tapes (Function Code 3)

This function applies only to physical files. Function code 3 can be used if your device is a tape drive connected to an iSBC 214 controller, an iSBC 221 controller, or an iSBX® 217C board mounted on an iSBC 215G controller. You can obtain specific information about the tape drive by calling A$SPECIAL with a token for the device connection, with spec$func set to 3, and with ioparm$ptr pointing to a structure of the form:

```
DECLARE  tape$drive$data STRUCTURE(
                            tape         BYTE,
                            reserved(7)  BYTE);
```

A$SPECIAL returns information to the tape field of this structure. The information in the tape field is encoded as follows (bit 0 is the low-order bit):

| Bits | Value and Meaning |
|---|---|
| 0 | Indicates whether the unit is present.<br>0 = The unit is not present.<br>1 = The unit is present. |
| 1-3 | Reserved bits. |
| 4-7 | Number of tracks on the tape. |

### Getting or Setting Attributes of a Terminal (Function Codes 4 and 5)

These functions apply only to physical files. You can get (receive) or set the characteristics of a terminal that is being driven by a terminal device driver by issuing a call to A$SPECIAL. In each case, you supply a token for a connection to a terminal. To get the data, set spec$func equal to 4, and to set the data, set spec$func equal to 5.

Before setting the terminal characteristics, you should invoke A$SPECIAL with function code 4 to get the current characteristics. Then, modify the returned structure to reflect your desired changes. Finally, invoke A$SPECIAL with function code 5 to set the characteristics, using your modified structure as input.

In this section, certain terms unique to terminal devices (for example, line editing, OSC sequences, translation) are described only briefly. If you are unfamiliar with these terms, refer to the *iRMX® Device Drivers User's Guide*.

For both getting and setting terminal characteristics, ioparm$ptr should point to a structure of the form:

```
DECLARE terminal$attributes STRUCTURE(
                num$words           WORD,
                num$used            WORD,
                connection$flags    WORD,
                terminal$flags      WORD,
                in$baud$rate        WORD,
                out$baud$rate       WORD,
                scroll$lines        WORD,
                x$y$size            WORD,
                x$y$offset          WORD,
                special$modes       WORD,
                high$water$mark     WORD,
                low$water$mark      WORD,
                fc$on$char          WORD,
                fc$off$char         WORD,
                link$parameter      WORD,
                spc$hi$water$mark   WORD,
                special$char(4)     BYTE);
```

where

num$words — The number of words, not including num$words and num$used fields, that are reserved for the remainder of the terminal$attributes data structure. To access all of the information, set this field to at least 16. Intel reserves the right to expand the length of this structure in later releases.

num$used — Tells how many of the attribute words are valid. The special subfunction get$terminal$attributes fills in the structure with up to num$words of the current values and sets num$used to the number of words returned.The subfunction set$terminal$attributes sets the structures first num$used attributes to the values specified.

However, if any of the first five attributes (connection$flags through scroll$lines) is zero, the Basic I/O System skips over the zeroed field, leaving it at its previous setting. For example, if num$used is 2, while connection$flags is 0 and terminal$flags is not 0, then A$SPECIAL uses the contents of the terminal$flags field to set terminal attributes, but it ignores the contents of connection$flags field. In this way, you can set some parameters without affecting others.

For the functions represented by the remaining fields in this structure, invoking A$SPECIAL is not the only way to set the functions. You can also set them with OSC sequences. The description of each field mentions, in parentheses, the OSC characters you can use. (OSC sequences are described in the *iRMX® Device Drivers User's Guide.*) You can also use the OSC Query sequence when debugging, to ensure that your tasks invoked A$SPECIAL correctly.

connection$flags

This word applies only to this connection to the terminal. (All other parameters apply to the terminal itself and therefore to all connections to the terminal.) If you attempt to set this field to zero, the Basic I/O System ignores your entry and leaves the field set to its previous value.

**NOTE**

Changes you make with connection$flags don't take effect until a read is processed using the connection. Therefore, to ensure that the changes take effect, you should read from the connection immediately after using connection$flags to change the connection's attributes. (If you don't expect input at the terminal, set the connection to flush mode, then read 255 characters from the connection. The read will return immediately with whatever characters were available.)

The next few pages describe the bit flags in this word. (Bit 0 is the low-order bit.)

| Bits | Value and Meaning |
|---|---|
| 0-1 | Line editing control (corresponds to OSC characters C:T). Line editing refers to how the Operating System handles control characters such as those that delete characters entered at a terminal, scroll terminal, output, and others. Refer to the *iRMX® Basic I/O System User's Guide* for more information.<br>0 = Invalid Entry.<br>1 = Transparent mode (no line editing). Input is transmitted to the requesting task exactly as entered at the terminal, except for signal characters (e.g., the Human Interface CONTROL-C) set by specifying "set signal" in the spec$func parameter of A$SPECIAL, and any enabled output control characters or OSC sequences. Before being transmitted, data accumulates in a buffer until the requested number of characters has been entered.<br>2 = Normal mode (line editing). Edited data accumulates in a buffer until a carriage return is entered.<br>3 = Flush mode (no line editing). Input is transmitted to the requesting task exactly as entered at the terminal, except for signal characters (e.g., the Human Interface CONTROL-C) set by specifying "set signal" in the spec$func parameter of A$SPECIAL, and any enabled output control characters or OSC sequences. Before being transmitted, data accumulates in a buffer until an input request is received. At that time, the contents of the buffer (or the number of characters requested, if the buffer contains more than that number) is transmitted to the requesting task. If any characters remain in the buffer, they are saved for the next input request. |
| 2 | Echo control (corresponds to OSC characters C:E). |

2 Echo control (corresponds to OSC characters C:E).

0 = Echo. Characters entered into the terminal are "echoed" to the terminal's display screen.

1 = Do not echo.

3 Input parity control (corresponds to OSC characters C:R). Characters entered into the terminal have their parity bits (bit 7) set to 0 or not set by the terminal support, according to the value of the input parity control bit.

0 = Set parity bit to 0.

1 = Do not alter parity bit.

4 Output parity control (corresponds to OSC characters C:W). Characters being output to the terminal have their parity bits (bit 7) set to 0 or not set by the terminal support, according to the value of the output parity control bit.

0 = Set parity bit to 0.

1 = Do not alter parity bit.

5 Output control character control (corresponds to OSC characters C:O). This bit specifies whether output control characters are effective when entered at the terminal. The value of this bit applies only to output through this connection. Control characters are described in the *iRMX® Device Drivers User's Guide*.

0 = Accept output control characters in the input stream.

1 = Ignore output control characters in the input stream.

6-7 OSC control sequence control (corresponds to OSC characters C:C). These bits specify whether OSC control sequences should be acted upon when they appear in the input stream and, separately, when they appear in the output stream. These bits apply only to input or output through this connection. OSC control sequences are described in the *iRMX® Device Drivers User's Guide*.

0 = Act upon OSC sequences that appear in either the input or output stream.

1 = Act upon OSC sequences in the input stream only.

2 = Act upon OSC sequences in the output stream only.

3 = Do not act upon any OSC sequences.

8 Specifies whether characters in the raw input buffer are moved to the type-ahead buffer by the interrupt task or the service task. The raw input and type-ahead buffers are discussed in the *iRMX® Device Drivers User's Guide*.

0 = Characters are moved from the raw input buffer to the type-ahead buffer by the interrupt task.

1 = Characters are moved from the raw input buffer to the type-ahead buffer by the service task.

9 Specifies whether the type-ahead buffer is used to process characters in the raw input buffer.

0 = Characters are moved from the raw input buffer to the type-ahead buffer.

1 = Characters are moved directly from the raw input buffer to the application task's buffer, thus bypassing the type-ahead and line-edit buffers. This disables all Terminal Support Code features.

10-15 Reserved bits. For future compatibility, set to 0.

terminal$flags

This word applies to the terminal and therefore to all connections to the terminal. If you attempt to set this field to zero, the Basic I/O System ignores your entry and leaves the field set to its previous value. The flags in this word are encoded as follows. (Bit 0 is the low-order bit.)

| Bits | Value and Meaning |
|---|---|
| 0 | Reserved bit. Set to 1. |
| 1 | Line protocol indicator (corresponds to OSC characters T:L). Full-duplex terminals support simultaneous and independent input and output. Half-duplex terminals support independent input and output, but not simultaneously.<br>0 = Full duplex.<br>1 = Half duplex. |
| 2 | Output medium (corresponds to OSC characters T:H).<br>0 = Video display terminal (VDT).<br>1 = Printed (Hard copy). |
| 3 | Modem indicator (corresponds to OSC characters T:M).<br>0 = Not used with a modem.<br>1 = Used with a modem. |

4-5 Input parity control bit (corresponding to OSC characters T:R) determines how the terminal driver handles input parity. The parity bit (bit 7) of each input byte can be used in a variety of ways. A byte has even parity if the sum of its bits is an even number. Otherwise, the byte has odd parity.

**NOTE**

If bits 4-5 contain 2 or 3, and bits 6-8 also contain 2 or 3, then they must both contain the same value. That is, they must both reflect the same parity convention (even or odd).

0 = Terminal driver always sets parity bit to 0. (7 bits of data)

1 = Terminal driver never alters the parity bit. (8 bits of data)

2 = Even parity is expected on input (7 bits of data). The terminal driver uses the eighth bit to indicate the presence (1) or absence (0) of an error on input. That is, the driver sets the parity bit to 0 unless the received byte has odd parity or there is some other error, such as (a) the received stop bit has a value of 0 (framing error) or (b) the previous character received has not yet been fully processed (overrun error.)

3 = Odd parity is expected on input (7 bits of data). The terminal driver uses the eighth bit to indicate the presence (1) or absence (0) of an error on input. That is, the driver sets the parity bit to 0 unless the received byte has even parity or there is some other error, such as (a) the received stop bit has a value of 0 (framing error) or (b) the previous character received has not yet been fully processed (overrun error.)

6-8 Output parity control bit (corresponding to OSC characters T:W). determines how the terminal driver handles output parity. The parity bit (bit 7) of each output byte can be used in a variety of ways. A byte has even parity if the sum of its bits is an even number. Otherwise, the byte has odd parity.

## NOTE

If bits 4-5 contain 2 or 3, and bits 6-8 also contain 2 or 3, then they must both contain the same value. That is, they must both reflect the same parity convention (even or odd).

0 = Terminal driver always sets parity bit to 0 (7 bits of data).

1 = Terminal driver always sets parity bit to 1 (7 bits of data).

2 = Terminal driver sets parity bit to give the byte even parity (7 bits of data)

3 = Terminal driver sets parity bit to give the byte odd parity (7 bits of data).

4 = Terminal driver does not alter the parity bit (8 bits of data).

5-7 Invalid values.

9 Translation control (corresponds to OSC characters T:T). Translation refers to the ability to define certain control characters so that whenever these characters are entered at or written to a terminal, certain actions, usually cursor movements, take place automatically. Translation is described in the *iRMX® Device Drivers User's Guide*.

0 = Do not enable translation.

1 = Enable translation.

10 Terminal axes sequence control (corresponds to OSC characters T:F). This specifies the order in which Cartesian-like coordinates of elements on a terminal's screen are to be listed or entered.

0 = List or enter the horizontal coordinate first.

1 = List or enter the vertical coordinate first.

11 Horizontal axis orientation control (corresponds to OSC characters T:F). This specifies whether the coordinates on the terminal's horizontal axis increase or decrease as you move from left to right across the screen.

0 = Coordinates increase from left to right.

1 = Coordinates decrease from left to right.

12 Vertical axis orientation control (corresponds to OSC characters T:F). This specifies whether the coordinates on the terminal's vertical axis increase or decrease as you move from top to bottom across the screen.

0 = Coordinates increase from top to bottom.

1 = Coordinates decrease from top to bottom.

13-15 Reserved bits. For future compatibility, set to 0.

in$baud$rate The input baud rate indicator (corresponds to OSC characters T:I). If you attempt to set this field to zero, the Basic I/O System ignores your entry and leaves the field set to its previous value. The word is encoded as follows:

0 = Ignore.

1 = Perform an automatic baud rate search.

Other = Actual input baud rate, such as 9600.

out$baud$rate

The output baud rate indicator (corresponds to OSC characters T:O). If you attempt to set this field to zero, the Basic I/O System ignores your entry and leaves the field set to its previous value. The word is encoded as follows:

0 = Leave field set to the previous value.

1 = Use the input baud rate for output.

Other = Actual output baud rate, such as 9600.

Most applications require the input and output baud rates to be equal. In such cases, use in$baud$rate to set the baud rate and specify a one for out$baud$rate.

scroll$lines

An operator at a terminal can enter a control character (default is CONTROL-W) when he/she is ready for data to appear on the terminal's display screen. The scroll$lines value (corresponding to OSC characters T:S) specifies the maximum number of lines that are to be sent to the terminal each time the operator enters the control character. If you attempt to set this field to zero, the Basic I/O System ignores your entry and leaves the field set to its previous value.

x$y$size

The low-order byte of this word specifies the number of character positions on each line of the terminal's screen (and corresponds to OSC characters T:X). The high-order byte specifies the number of lines on the terminal's screen (and corresponds to OSC characters T:Y).

x$y$offset

The low-order byte of this word specifies the value that starts the numbering sequence of both the X and Y axes (and corresponds to OSC characters T:U). The high-order byte specifies the value to which the numbering of the axes must "fall back" after reaching 127 (and corresponds to OSC characters T:V).

special$modes

This and the following fields apply only to buffered devices (such as the iSBC 544A and the iSBC 188/48 boards). These devices maintain their own input and output buffers separately from the ones managed by the Basic I/O System's Terminal Support Code. If you aren't sure whether you can set these fields, invoke A$SPECIAL with function code 4 to get the terminal attributes. If bit 15 of the special$modes field is set, your board is a buffered device and you can set the bits in special$modes and the following fields.

Bits — Value and Meaning

0

Flow control mode specifies whether the communications board sends flow control characters (selected by the fc$on$char and fc$off$char fields, but usually XON and XOFF) to turn input on and off (corresponds to the OSC characters T:G). The low-order bit (bit 0) controls this option, as follows:

0 = Disable flow control.

1 = Enable flow control.

When flow control is enabled, the communication board can control the amount of data sent to it to prevent buffer overflow. This is especially important when communicating with another computer.

1

Special Character Mode (corresponds to OSC characters T:D). If your device supports special characters (currently, the iSBC 188/48/56, 546, 547, 548, and 549 boards do), the device is capable of sending an interrupt whenever a special character (defined later in the special array) is typed. When Special Character Mode is on, the device uses interrupts to inform the Terminal Support Code that special characters have been entered.

If a special character has also been defined as a signal character, the Terminal Support Code sends a unit to the appropriate signal semaphore as soon as it receives the special character interrupt.

When Special Character mode is off, the device sends special characters through the normal input stream. If the characters are signal characters, the Terminal Support Code sends units to the appropriate semaphores when the characters reach the line-edit buffer.

The setting of this bit is as follows:

0 = Disable Special Character Mode.

1 = Enable Special Character Mode.

| | |
|---|---|
| | The Special Character High Water mark (corresponds to OSC characters T:A) is used in conjunction with this field to control Special Character Mode. |
| 2-14 | Reserved bits. Set to 0. |
| 15 | Buffered Device Control. This bit is set by the terminal support to show if a device is buffered. If invoking the A$SPECIAL system call to get terminal attributes shows that this bit is set, then the special$modes bits and the data fields following are valid.<br>0 = Not a buffered device.<br>1 = Buffered device. |

The remaining fields in the structure apply only to buffered devices.

high$water$mark When the communication board's buffer fills to contain the number of bytes represented by this field, the board's firmware sends the flow control "off" character to stop input. (This field corresponds to the OSC characters T:J.)

The high-water mark of the iSBC 544A board is not configurable; therefore, setting this field has no effect on that board.

low$water$mark When the number of bytes in the communication board's buffer drops to the number represented by this field, the board's firmware sends the flow control "on" character to start input. (This field corresponds to the OSC characters T:K.)

The low-water mark of the iSBC 544A board is not configurable; therefore, setting this field has no effect on that board.

fc$on$char An ASCII character that the communication board sends to the connecting device when the number of bytes in its buffer drops to the low-water mark. Normally this character tells the connecting device to resume sending data. (This field corresponds to the OSC characters T:P.)

The fc$on$char for the iSBC 544A board is set to the XON character and is not configurable; therefore, setting this field has no effect on that board.

fc$off$char

An ASCII character that the communication board sends to the connecting device when the number of characters in its buffer rises to the high-water mark. Normally this character tells the connecting device to stop sending data. (This field corresponds to the OSC characters T:Q.)

The fc$off$char for the iSBC 544A board is set to the XOFF character and is not configurable; therefore, setting this field has no effect on that board.

link$parameter

(Corresponds to the OSC characters T:N) This word specifies the characteristics of the physical link between the terminal and a device. Not all device drivers support link$parameter. This field is supported by those boards supported by the Terminal Communications Controller driver, including the iSBC 188/48, iSBC 188/56, iSBC 546, iSBC 547, iSBC 548, and iSBC 549 controllers.

The meaning of the bits in this field are as follows:

| Bits | Value and Meaning |
|---|---|
| 0-1 | Parity<br>0 = No parity<br>1 = Invalid value<br>2 = Even parity<br>3 = Odd parity |
| 2-3 | Character length<br>0 = 6 bits/character.<br>1 = 7 bits/character.<br>2 = 8 bits/character.<br>3 = invalid value |
| 4-5 | Number of stop bits.<br>0 = 1 stop bit.<br>1 = 1 1/2 stop bits.<br>2 = 2 stop bits.<br>3 = invalid value |
| 6-14 | Reserved |
| 15 | Check if this word is to be used<br>0 = not used<br>1 = used |

If parity is enabled, an additional bit position beyond those specified in the Character Length control is added to the transmitted data and expected in received data. The received parity bit is transferred to the CPU as part of the data unless 8 bits/character is selected. If a parity error is detected on input, the character is discarded.

In the 6 and 7 bits/character modes, unused bit positions in transmit data are ignored. Unused bits in receive data are set to 1. If a framing error is detected on input, the character is returned as an 8-bit null (00H).

Bit 15 is checked to see if this word is to be used. If set to 1, the driver passes the low-order byte to the controller, which sets the parity, character length, and stop bits. If set to 0, this word is skipped and the terminal$flags field is used.

spc$hi$water$mark — (Corresponds to the OSC characters T:A). This field is used in conjunction with the Special Characters field (corresponds to the OSC characters T:D) to control Special Character Mode. When the device's input buffer fills to contain the number of characters specified in this field, Special Character Mode is enabled (assuming the Special Character field is turned on). If the number of characters in the device's input buffer is less than the high water mark, Special Character Mode is disabled, even if the Special Character field is turned on.

If the Special Character field is turned off, this field has no effect.

special$char(4) — This array holds the characters you define as special characters (and corresponds to the OSC characters T:Z). If you define less than four special characters, then you must fill the remaining slots in the array with duplicates of the last character you define.

### Designating Characters for Signaling from a Terminal Keyboard (Function Code 6)

You can use the A$SPECIAL system call to associate a keyboard character with a semaphore, so that whenever the character is entered into the terminal, the Basic I/O System automatically sends a unit to the semaphore. Up to 12 character-semaphore pairs can be so associated simultaneously; each character being associated with a different semaphore, if desired. Character-semaphore pairs are called Signal Characters.

To set up a signal character, call A$SPECIAL with a device connection, with spec$func equal to 6, and with ioparm$ptr pointing to a structure of the form:

```
DECLARE signal$pair STRUCTURE(
                        semaphore   TOKEN,
                        character   BYTE);
```

where

semaphore — A TOKEN for the semaphore that is to be associated with the character.

character — If the character value is in the range 0 to 1FH, or is 7FH, the terminal support sends a unit to the associated semaphore when it receives the ASCII equivalent of this value.

If you add 20H to the character values in the 0 to 1FH range (making this range 20H to 3FH), or if the value is 40H, then the type ahead buffer (and the input buffer if this is a buffered device) is cleared and a unit is sent to the associated semaphore.

To delete a signal character, call A$SPECIAL with the semaphore field set to NIL and character set to the signal character to be deleted.

### Tape Drive Functions (Function Codes 7, 8, 9 and 10)

You can use the A$SPECIAL system call to perform four different functions that apply to tape drives only. These functions include rewinding a tape, searching for file marks, writing file marks, and retensioning a tape.

When your task issues the A$SPECIAL system call with spec$func set to 7, the tape drive rewinds a tape to its load point. This function also terminates tape read and write operations. If the tape drive was performing a write operation when you initiated this call, the tape drive writes a file mark before rewinding the tape.

When your task issues the A$SPECIAL system call with spec$func set to 8, the tape drive moves the tape to the next file mark on the tape. This function also terminates tape read operations.

When your task issues the A$SPECIAL system call with spec$func set to 9, the tape drive writes a file mark at the current position on the tape. This function also terminates tape write operations.

When your task issues the A$SPECIAL system call with spec$func set to 10, the tape drive fast-forwards the tape to the end and then rewinds it to the load point.

### Set Font (Function Code 11)

You can use the A$SPECIAL system call with other Intel products to specify a font to be used in graphics.

## Get and Set Bad Track or Bad Sector Information (Function Codes 12 and 13)

You can use the A$SPECIAL system call to set (write) or get (read) the bad track/sector information of a volume. To perform either of these operations set the spec$func parameter to the correct value (12 to set; 13 to get). The ioparm$ptr parameter must point to a structure of the following form:

```
DECLARE bad$track$info STRUCTURE(
            reserved            WORD,
            count               WORD,
            bad$tracks (255)    DWORD),
        badtracks (255) STRUCTURE (
            cylinder            WORD,
            head                BYTE,
            sector              BYTE)
        AT (@bad$track$info.bad$tracks);
```

In this structure, the fields are defined as:

reserved — A WORD that is reserved for use by the driver.

count — A WORD containing the number of bad tracks/sectors listed in the bad$tracks STRUCTURE, up to the maximum of 255.

bad$tracks — A STRUCTURE used to store the bad track/sector list. For each bad track/sector, the structure defines the cylinder and surface as follows:

- cylinder — A WORD that gives the cylinder number of the bad track or sector.
- head — A BYTE that gives the head (surface) number of the bad track or sector.
- sector — A BYTE that gives the sector number of the bad sector on a track. On devices that only support bad track information this value should be set to zero.

Once you have correctly set the parameters for the A$SPECIAL call, perform either the get or set operation as follows:

Set — You Set (write) the Bad Track Information by performing the A$SPECIAL call (with the correct parameters) and specifying the number of bad tracks or sectors to be entered in the Bad Track/Sector Information Block and supplying a list of bad tracks or sectors in ascending order.

Any information already existing in the volume's Bad Track/Sector Information Block will be overwritten. If you wish to modify existing information, first Get the Bad Track/Sector Information and modify it, then Set the Bad Track/Sector Information.

Get — You Get (read) the Bad Track Information by performing the A$SPECIAL call (with the correct parameters) and examining the contents of the bad$track$info structure. A value of zero in the count field indicates that no valid information is available or that there are no bad tracks.

### Getting Terminal Status (Function Code 16)

This function applies only to physical files. You can get the status of a terminal that is being driven by a terminal device driver by issuing a call to A$SPECIAL.

In this section, certain terms unique to terminal devices (for example, line-editing, OSC sequences, translation) are described only briefly. If you are unfamiliar with these terms, refer to the *iRMX® Device Drivers User's Guide.*

To get a terminal's status, call A$SPECIAL with a connection for the terminal, with spec$func equal to 16, and with ioparm$ptr pointing to a structure of the form:

```
DECLARE terminal$status STRUCTURE(
                 terminal$flags          WORD,
                 input$conn$flags        WORD,
                 input$state             WORD,
                 input$conn              TOKEN,
                 input$count             WORD,
                 input$actual            WORD,
                 raw$buf$count           WORD,
                 typeahead$count         BYTE,
                 num$input$requests      BYTE,
                 output$conn$flags       WORD,
                 output$state            WORD,
                 output$conn             TOKEN,
                 output$count            WORD,
                 output$actual           WORD,
                 out$buf$count           WORD,
                 num$output$requests     BYTE);
```

where

terminal$flags — The current attributes associated with the terminal. For the meaning of the bits in this word, see the terminal$flags parameter in the description of function codes 4 and 5 of the A$SPECIAL system call.

input$conn$flags — The current attributes associated with the terminal's active input connection. For the meaning of the bits in this word, see the connection$flags parameter in the description of function codes 4 and 5 of the A$SPECIAL system call.

input$state

The internal state of this terminal's input connection. The bits in this WORD are encoded as follows. (Bit 0 is the low-order bit.)

| Bits | Value and Meaning |
|---|---|
| 0 | Indicates whether an input request has been set up.<br>0 = An input request has not been set up.<br>1 = An input request has been set up. |
| 1 | Indicates whether the current input request has completed.<br>0 = The current input request has not completed.<br>1 = The current input request has completed. |
| 2 | Reserved |
| 3 | Indicates whether an Operating System Command (OSC) sequence is being processed.<br>0 = An OSC sequence is not being processed.<br>1 = An OSC sequence is being processed. |
| 4 | Indicates whether a complete line has been processed and is ready for transfer from the line-edit buffer to theapplication task's buffer. Only applies to terminals in line-edit mode.<br>0 = A complete line has not been processed.<br>1 = A complete line has been processed. |
| 5 | Indicates whether the current character was preceded by a CONTROL-P (quoting character) and is being interpreted as data, rather than as a line-editing character. Output control characters, such as CONTROL-S and CONTROL-Q perform their normal functions even if preceded by CONTROL-P. Only applies to terminals in line-edit mode.<br>0 = The current character was not preceded by a CONTROL-P.<br>1 = The current character was preceded by a CONTROL-P. |

6 Indicates whether an escape sequence is being processed.

0 = An escape sequence is not being processed.

1 = An escape sequence is being processed.

7 Indicates whether a CONTROL-R is being used to recall the last line. Only applies to terminals in line-edit mode.

0 = The last line is not being recalled.

1 = The last line is being recalled.

8 Indicates whether this terminal is on-line and available for use. Only applies to terminal configured for use with a modem.

0 = The terminal is not available for use.

1 = The terminal is available for use.

9 Indicates whether this terminal is waiting for a ring interrupt as a result of a modem query OSC command. Only applies to terminals configured for use with a modem.

0 = The terminal is not waiting for a ring interrupt.

1 = The terminal is waiting for a ring interrupt.

10 Indicates whether this terminal is waiting for a carrier loss interrupt as a result of a modem query OSC command. Only applies to terminals configured for use with a modem.

0 = The terminal is not waiting for a carrier loss interrupt.

1 = The terminal is waiting for a carrier loss interrupt.

11 Indicates whether this terminal has a modem query pending as a result of a modem query OSC command. Only applies to terminals configured for use with a modem.

0 = The terminal does not have a modem query pending.

1 = The terminal does have a modem query pending.

12, 13 Reserved.

14 Indicates whether the current line has been cancelled. Only applies to terminals in line-edit mode.

0 = The current line has not been cancelled.

1 = The current line has been cancelled.

15 Indicates whether the type-ahead buffer is full.

0 = The type-ahead buffer is not full.

1 = The type-ahead buffer is full.

input$conn A TOKEN for the most recently used input connection associated with this terminal.

input$count The number of characters requested by the latest input request.

input$actual The number of characters that were moved from the raw input or type-ahead buffer to the application task's buffer during the latest request.

raw$buf$count The number of characters available in the raw input buffer.

typeahead$count The number of characters available in the type-ahead buffer.

num$input$requests The number of input requests in the input queue for this terminal.

output$conn$flags The current attributes associated with the terminal's active output connection. For the meaning of the bits in this word, see the connection$flags parameter in the description of function codes 4 and 5 of the A$SPECIAL system call.

output$state The internal state of this terminal's output connection. This parameter can be used to determine if a terminal's output is hindered in some way (for example, because an XOFF was received). To check for hindered output, AND output$state with the value 1E0H. If the result is non-zero, output is hindered. You can resume terminal output by invoking A$SPECIAL with function code 18.

The bits in this WORD are encoded as follows. (Bit 0 is the low-order bit.)

| Bits | Value and Meaning |
|---|---|
| 0-1 | 0 = Output character processing is occurring normally without an escape character being encountered.<br>1 = An ESC character has been encountered in the output stream. This requires special handling because it may be part of an escape or OSC sequence or it may need to be translated.<br>2 = The previously encountered escape character is part of an OSC sequence that is being processed.<br>3 = The previously encountered escape character is part of an escape sequence that is being translated. |
| 2 | Indicates whether an output request has been set up.<br>0 = An output request has not been set up.<br>1 = An output request has been set up. |
| 3 | Indicates whether the terminal controller is transmitting characters from the current output request or is ready to transmit a character from the next output request. Only applies to non-buffered devices.<br>0 = The terminal controller is busy transmitting characters from the current request on an interrupt-driven basis.<br>1 = The terminal controller is ready to transmit a character once the next output request arrives. |
| 4 | Reserved. |

5 Indicates whether this terminal's output is being discarded (in discarding mode).

0 = Not in discarding mode.

1 = In discarding mode.

6 Indicates whether this terminal's output is blocked because an XOFF was received or a page scroll has completed (placing output into stopped mode).

0 = Output is not blocked.

1 = Output is blocked.

7 Indicates whether this terminal's output is in scroll mode.

0 = Not in scroll mode.

1 = In scroll mode.

8 Indicates whether the terminal's output is blocked because an XOFF was received.

0 = Output is not blocked.

1 = Output is blocked.

9 Indicates whether the terminal's current output request has been cancelled and is being flushed.

0 = The terminal request has not been cancelled.

1 = The terminal request has been cancelled.

10-15 Reserved.

output$conn A TOKEN for the most recently used output connection associated with this terminal.

output$count The number of characters requested by the latest output request.

output$actual The number of characters moved from the application task's buffer into the output buffer during the latest output request.

out$buf$count The number of characters still awaiting output from the output buffer of the Terminal Support Code or the buffered device.

num$output$requests — The number of output requests in the output queue for this terminal.

### Cancelling Terminal I/O (Function Code 17)

The A$SPECIAL system call allows a program to cancel all requests associated with a specified connection to a terminal.

To cancel all requests associated with a connection to a terminal, call A$SPECIAL with a connection for the terminal, with spec$func equal to 17, and with ioparm$ptr pointing to a structure of the form:

```
DECLARE cancel$io$struc STRUCTURE( cancel$conn$t   TOKEN);
```

where

cancel$conn$t — A TOKEN for the connection whose requests are to be cancelled. Setting cancel$conn$t to SELECTOR$OF(NIL) cancels all input requests associated with the connection specified by A$SPECIAL's connection parameter. To determine which connection is active and can be cancelled, invoke A$SPECIAL with spec$func equal to 16 (get terminal status) and check the TOKEN returned in the input$conn parameter.

#### NOTE

The cancel terminal I/O function cancels all requests that are using the specified connection. Therefore, unless you have a reason to do otherwise, each task using a particular terminal device should have its own connection to the device. Then the requests associated with a private connection can be cancelled without affecting other input requests on the same terminal device.

### Resuming Terminal I/O (Function Code 18)

The A$SPECIAL system call allows a program to resume an output request that is blocked because an output control character was entered at the terminal. To resume an output request that is blocked, call A$SPECIAL with any connection for the blocked terminal and with spec$func equal to 18. The ioparm$ptr parameter is ignored.

## Condition Codes

A$SPECIAL returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide*.

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

### Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

E$OK 0000H No exceptional conditions.

E$BUFFERED$CONN 0036H The connection parameter you supplied was opened with an Extended I/O System call. You cannot use it with the A$READ system call.

E$EXIST 0006H At least one of the following is true:

- One or more of the following parameters or fields is not a token for an existing object:
    - The connection parameter
    - The resp$mbox parameter
    - The mailbox field in the notify structure. (Spec$func = 2.)
    - The object field in the notify structure. (Spec$func = 2.)
    - The semaphore field in the signal$pair structure. (Spec$func = 6.)
- The connection is being deleted.
- The connection for a remote driver is no longer active.

E$IFDR 002FH The function requested (spec$func) is not valid for the type of file specified by the connection parameter.

E$LIMIT 0004H The calling task's job has already reached its object limit.

E$MEM 0002H The memory available to the calling task's job is not sufficient to complete the call.

E$NOT$CONFIGURED 0008H This system call is not part of the present configuration.

E$PARAM 8004H At least one of the following is true:

- The spec$func parameter was greater than 18 but less than 32K.
- The entire user-provided structure does not have the correct read/write accesses as described in the following list:

| Not Readable | Not Writeable |
|---|---|
| format track | get disk/tape date |
| notify | get terminal data |
| set terminal data | get bad track information |
| set signal | |
| set bad track information | |

E$SUPPORT 0023H The specified connection was not created by this job.

E$TYPE 8002H One or more of the following parameters or fields is a token for an existing object of the wrong type:

- The connection parameter.
- The resp$mbox parameter.
- The mailbox field of the notify structure. (Spec$func = 2.)
- The semaphore field of the signal$pair structure. (Spec$func = 6.)

### Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

E$OK 0000H No exceptional conditions.

E$CONN$NOT$OPEN 0034H The specified connection is not open. This applies only to stream and physical files.

E$FLUSHING 002CH The specified connection was closed before the function could be completed.

E$IDDR 002AH The specified function is not supported by the device containing the file.

| | | |
|---|---|---|
| E$IO | 002BH | An I/O error occurred that might have prevented the operation from completing. Examine the unit$status field of the IORS for more information. |
| E$IO$ALT$ASSIGNED | 0058H | A warning that an alternate has been assigned for a bad track. |
| E$IO$MEM | 0042H | The memory pool of the Basic I/O System on the server does not have a block of memory large enough to allow the system call to finish. |
| E$IO$NO$SPARES | 0057H | An attempt was made to assign an alternate track, but no more alternate tracks were available. |
| E$NOT$DEVICE$CONN | 0033H | The function code is 'notify', but the specified connection is not a device connection. This applies only to named and physical files. |
| E$SPACE | 0029H | One of the following is true:<br>• This call attempted to format a track of a physical file that is beyond the end of the volume.<br>• This call attempted to format a track of a RAM disk other than track 0. |
| E$STREAM$SPECIAL | 003CH | One of the following is true:<br>• This is a "query" request, but another query is already queued. This applies only to stream files.<br>• This is a "satisfy" request, but either a query request is queued, or no requests are queued. This applies only to stream files. (See Artificially Satisfying a Stream File I/O Request in the Description.) |

A$TRUNCATE truncates a named file at the current setting of the file pointer, freeing all allocated space beyond the pointer.

```
    CALL RQ$A$TRUNCATE(connection, resp$mbox, except$ptr);
```

## Input Parameter

connection — A TOKEN for an open connection to the file being truncated.

## Output Parameters

resp$mbox — A TOKEN for the mailbox that receives an I/O request/result segment (IORS) indicating the result of the call (for details on the IORS, see Appendix A). A value of SELECTOR$OF(NIL) means that you do not want to receive an IORS.

If it receives an IORS, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

## Description

The A$TRUNCATE system call applies to named files only. This call truncates a file at the current setting of the file pointer, freeing all allocated space beyond the pointer. A$SEEK can be called to position the pointer before A$TRUNCATE is called. If the file pointer is at or beyond the end-of-file, no operation is performed.

Truncation is performed immediately, rather than waiting until connections to the file are deleted.

### NOTE

The designated file connection must be open for writing and must have update access to the file.

File pointers for other connections to the file are not affected by the truncation operation. Thus, it is possible that file pointers for other connections to the file will be beyond the new end-of-file after the A$TRUNCATE call. If a task invokes the A$READ system call with a file pointer beyond the end-of-file, the Basic I/O System behaves as though the reading operation began at the end-of-file. If a task invokes the A$WRITE system call with a file pointer beyond the end-of-file, the Basic I/O System attempts to expand the file. If the Basic I/O System does expand your file in this manner, the file contains random information between the old end-of-file and the point in the file where the write begins.

## Condition Codes

A$TRUNCATE returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide.*

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

### Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$BUFFERED$CONN | 0036H | The connection is a connection produced by the Extended I/O System. You cannot use it with Basic I/O System calls. |
| E$EXIST | 0006H | At least one of the following is true:<br>• One or more of the following parameters is not a token for an existing object:<br>- The connection parameter<br>- The resp$mbox parameter<br>• The connection is being deleted.<br>• The connection for a remote driver is no longer active. |
| E$IFDR | 002FH | This system call applies only to named files, but the connection parameter specified some other type of file. |

| | | |
|---|---|---|
| E$LIMIT | 0004H | At least one of the following is true:<br>• The calling task's job has already reached its object limit.<br>• The number of outstanding I/O operations for a remote connection has been exceeded. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete the call. |
| E$NOT$CON-<br>FIGURED | 0008H | This system call is not part of the present configuration. |
| E$SUPPORT | 0023H | The specified connection was not created by this job. |
| E$TYPE | 8002H | At least one of the following is true:<br>• The connection parameter is a token for an object that is not a connection.<br>• The resp$mbox parameter is a token for an object that is not a mailbox. |

### Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$CONN$NOT$OPEN | 0034H | The specified file is not open for writing or updating. |
| E$FACCESS | 0026H | An attempt was made to truncate a file that was created with no update access. |
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the IORS for more information. |

# A$UPDATE

A$UPDATE updates a device by writing all partial sectors that remain in the Basic I/O System's buffers after the most recent operation that involved writing to the volume. Such operations include creating, truncating, and writing operations.

```
    CALL RQ$A$UPDATE(connection, resp$mbox, except$ptr);
```

## Input Parameters

connection — A TOKEN for a file or device connection. A$UPDATE updates all files on the device.

resp$mbox — A TOKEN for the mailbox that receives an I/O request/result segment (IORS) indicating the result of the call. (For details on the IORS, see Appendix A) A value of SELECTOR$OF(NIL) means that you do not want to receive an IORS.

If it receives an IORS, the calling task should call DELETE$SEGMENT to delete the segment after examining it.

## Output Parameter

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

## Description

When the I/O System performs an A$WRITE operation, it writes only entire sectors. If part of a sector remains to be written, the I/O System, unless requested to finish the writing operation (that is, to "update the file"), leaves the data for a partial sector in an output buffer. The next time A$WRITE is called on behalf of that file, the I/O System combines the leftover data in the buffer with the data in the new request and again begins writing entire sectors.

The A$UPDATE system call forces the Basic I/O System to finish the writing operation for a device; that is, it writes all partial buffers pertaining to files on a particular device. This ensures that files on removable volumes (such as diskettes) are updated before the operator removes the volume. However, the A$UPDATE system call has no effect on buffers that the Extended I/O System manages.

Three different events can cause the Basic I/O System to "update" a file. One, of course, is a call to A$UPDATE. The other two, called fixed updating and timeout updating, are triggered by the passing of (possibly different) amounts of time. You specify the time periods, and the devices to which they apply, when you configure the Basic I/O System.

Fixed updating occurs when an amount of time, which is specified for an entire system, passes. At that time, all devices to which updating applies are "updated". When configuring the Basic I/O System, you specify, for each I/O device, whether fixed updating applies to that device.

Timeout updating is just like fixed updating, except in two respects. First, the time period is defined separately for each device, rather than applying to the system as a whole. When configuring the Basic I/O System, you specify, for each I/O device, whether timeout updating applies to that device, and if it does, what the timeout period is to be for that device.

The second difference between timeout updating and fixed updating is that, in timeout updating, the timeout period starts at the end of each I/O operation, while fixed updating is independent of I/O activity. In I/O intensive systems, updating can be delayed if the timeout period is longer than the average time between I/O functions.

## Condition Codes

A$UPDATE returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User's Guide*.

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

## Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | At least one of the following is true:<br>• One or more of the following parameters is not a token for an existing object:<br>- The connection parameter<br>- The resp$mbox parameter<br>• The connection is being deleted.<br>• The connection for a remote driver is no longer active. |
| E$IFDR | 002FH | An attempt was made to update a stream file connection. |
| E$LIMIT | 0004H | At least one of the following is true:<br>• The calling task's job has already reached its object limit.<br>• The number of outstanding I/O operations for a remote connection has been exceeded. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete the call. |
| E$NOT$CON-FIGURED | 0008H | This system call is not part of the present configuration. |
| E$SUPPORT | 0023H | The specified connection was not created by this job. |
| E$TYPE | 8002H | At least one of the following is true:<br>• The connection parameter is a token for an object that is not a connection.<br>• The resp$mbox parameter is a token for an object that is not a mailbox. |

## Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see the Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the IORS for more information. |
| E$NOT$FILE$CONN | 0032H | The connection parameter is a device connection, not a file connection. |

# A$WRITE

A$WRITE writes data from the calling task's buffer to a connected file.

```
    CALL RQ$A$WRITE(connection, buff$ptr, count, resp$mbox,
                    except$ptr);
```

## Input Parameters

connection — A TOKEN for the open connection through which the write operation is to take place.

buff$ptr — A POINTER to the buffer that contains the data to be written.

count — A WORD giving the number of bytes to be written.

## Output Parameters

resp$mbox — A TOKEN for the mailbox that receives an I/O request/result segment (IORS) indicating the result of the call. (For details on the IORS, see Appendix A.) A value of SELECTOR$OF(NIL) means that you do not want to receive an IORS (you do not want to check the status of the WRITE operation).

If your task receives an IORS, it should call DELETE$SEGMENT to delete the segment after examining it. If you use the RQ$WAIT$IO BIOS system call to check the concurrent condition code after A$WRITE executes, the IORS is deleted for you.

If all the other connections to a stream file are requesting write operations, an actual value of zero and a status value of E$FLUSHING are returned in the IORS.

except$ptr — A POINTER to a WORD where the sequential condition code will be returned.

## Description

The A$WRITE call writes data from the caller's buffer to a connected file. The data is written starting at the current location of the connection's file pointer. After the write operation, the file pointer is positioned just after the last byte written. Some efficiency may be gained by starting writes on device block boundaries and writing an integral number of device blocks.

It is possible to use the A$SEEK system call to position the file pointer beyond the end of the file and commence writing. (This applies to named files only.) If a task does this, the Basic I/O System will extend the file to accommodate the writing operation. However, the data located between the old end of file and the beginning of the writing operation is undefined.

### NOTES

The buffer supplying the data to be written should not be modified until the write request has been acknowledged at the response mailbox.

The designated file connection must be open for writing, and it must have append or update access to the file.

Update access is used if you are writing over existing data. Append access is used if you are extending a file.

## Special Considerations for iRMX®-NET

iRMX-NET's Remote File Driver does not perform fragmentation and reassembly. For optimal performance, reading and writing should begin at offsets that are integral multiples of the remote server's buffer size. The device$gran parameter returned by the A$GET$FILE$STATUS system call indicates the buffer size of a remote server.

## Condition Codes

A$WRITE returns condition codes at two different times. The code returned to the calling task immediately after invocation of the system call is considered a sequential condition code. A code returned as a result of asynchronous processing is a concurrent condition code. A complete explanation of sequential and concurrent parts of system calls is in the *iRMX® Basic I/O System User Guide*.

The following list is divided into two parts--one for sequential codes, and one for concurrent codes.

## Sequential Condition Codes

The Basic I/O System can return the following condition codes to the word specified by the except$ptr parameter of this system call.

E$OK 0000H No exceptional conditions.

E$BAD$BUFF 8023H **This condition code is returned only in the iRMX II Operating System.**

At least one of the following is true:

- The user-provided memory buffer is not readable or crosses memory boundaries.
- The target memory buffer crosses a segment boundary.

E$BUFFERED$CONN 0036H The connection parameter you supplied was opened with an Extended I/O System call. You cannot use it with the A$READ system call.

E$EXIST 0006H At least one of the following is true:

- One or more of the following parameters is not a token for an existing object:
    - The connection parameter
    - The resp$mbox parameter
- The connection is being deleted.

E$LIMIT 0004H The calling task's job has already reached its object limit.

E$MEM 0002H The memory available to the calling task's job is not sufficient to complete the call.

E$NOT$CONFIGURED 0008H This system call is not part of the present configuration.

E$SUPPORT 0023H The specified connection was not created by this job.

E$TYPE 8002H At least one of the following is true:

- The connection parameter is a token for an object that is not a connection.
- The resp$mbox parameter is a token for an object that is not a mailbox.

## Concurrent Condition Codes

The Basic I/O System returns one of the following condition codes in an IORS at the mailbox specified by resp$mbox. (For details on the IORS, see Appendix A.) After examining this segment, you should delete it.

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$CONN$NOT$OPEN | 0034H | The connection is not open for writing or updating. |
| E$FACCESS | 0026H | The specified connection does not have "update" or "append" access to the file. |
| E$FLUSHING | 002CH | At least one of the following is true:<br>• The specified connection was closed before the write operation could be performed.<br>• The file specified by the connection parameter is a stream file, and all other connections are also requesting to write to the file. (See the description of resp$mbox.) |
| E$FRAGMENTATION | 0030H | The file is too fragmented to be extended. Try copying the file to a temporary file, deleting the original file, and renaming the temporary file to the original name. |
| E$IO | 002BH | An I/O error occurred which might have prevented the operation from completing. Examine the unit$status field of the IORS for more information. |
| E$SPACE | 0029H | At least one of the following is true:<br>• The volume is full.<br>• The operation attempted to write beyond the end of the device. This applies only to physical files. |
| E$SUPPORT | 0023H | The write operation, if carried out, would extend the file, but as the Basic I/O System is configured, extending a file is not allowed. |

# CREATE$USER

The CREATE$USER system call creates a user object.

## CAUTION

**This system call overrides the file access protection mechanism provided by the Basic I/O System. It should be used only by system programmers in charge of security management.**

```
user = RQ$CREATE$USER(ids$ptr, except$ptr);
```

## Input Parameter

ids$ptr

A POINTER to a structure of the following form:

```
DECLARE  ids STRUCTURE(
                length  WORD,
                count   WORD,
                id(*)   WORD);
```

where

length

Number of elements in the ID array.

count

Number of IDs (from the ID array) that are to be included in the user object. This number must be less than or equal to length, but greater than or equal to one.

id

Array of IDs, each of which is included in the user object. The first ID is to be used as the owner ID for any file created with reference to this user object.

## Output Parameters

user

A TOKEN where a token for the new user object will be returned.

except$ptr

A POINTER to a WORD where the condition code will be returned.

## Description

The CREATE$USER system call creates a user object. It accepts a list of IDs and returns a token for the new object.

If the number of ID slots, as specified by the length field, is greater than the number of IDs, as specified by the count field, the effect is as if length had been reduced to equal count.

## Condition Codes

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$LIMIT | 0004H | The calling task's job has already reached its object limit. |
| E$MEM | 0002H | The memory available to the calling task's job is not sufficient to complete the call. |
| E$PARAM | 8004H | The count field in the ids structure either is zero or is greater than the length field. |

The DELETE$USER system call deletes a user object.

## CAUTION

**This system call overrides the file access protection mechanism provided by the Basic I/O System. It should be used only by system programmers in charge of security management.**

```
    CALL RQ$DELETE$USER(user, except$ptr);
```

## Input Parameter

| | |
|---|---|
| user | A TOKEN for the user object to be deleted. |

## Output Parameter

| | |
|---|---|
| except$ptr | A POINTER to a WORD where the condition code will be returned. |

## Description

The DELETE$USER system call deletes a user object. Deleting a user object has no effect on connections created with the user object.

## Condition Codes

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | The user parameter is not a token for an existing object. |
| E$LIMIT | 0004H | The call cannot be processed without exceeding the number (255 decimal) of I/O operations which can be outstanding at one time for the user object specified in the call. |
| E$NOT$CON-<br>FIGURED | 0008H | This system call is not part of the present configuration. |
| E$TYPE | 8002H | The user parameter is a token that is not a user object. |

The ENCRYPT system call encodes a specified string of characters. This system call is typically used to encrypt a password supplied by a user during logon or other system access verification.

```
CALL RQ$ENCRYPT(password$ptr, key$ptr, encryption$ptr, except$ptr);
```

## Input Parameters

password$ptr
A POINTER to a STRING containing the data to be encrypted. The length of this STRING may be between one and eight bytes.

key$ptr
A POINTER to two ASCII characters that serve as an encryption key. These two characters become the second and third BYTES of the STRING pointed to by encryption$ptr. The two characters must be used in subsequent encryptions of the same (unencrypted) password to yield the same encryption.

## Output Parameters

encryption$ptr
A POINTER to a 15 character iRMX STRING where the encrypted password will be placed. The first character is the length of the STRING. The second and third characters are the key used to encrypt the password. The next 11 characters are the encrypted password. The last character is a null terminater.

except$ptr
A POINTER to a WORD where the condition code will be returned.

## Description

ENCRYPT encodes a string (typically a password) pointed to by the password$ptr parameter. It places the encrypted string in the 15 character buffer pointed to by encryption$ptr. The encryption is done using the Data Encryption Standard (DES) algorithm. See the Federal Information Processing Standard Publication #46, January 15, 1977, for more information. Note that there is no way to decrypt the encrypted string with this system call. The key$ptr parameter allows the input parameter to be encrypted to the same string each time ENCRYPT is called, provided the key$ptr parameter is identical. Using any other key will cause the input parameter to be encrypted differently. When a string is initially encrypted, the key should be randomly generated.

## Condition Codes

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$LIMIT | 0004H | The calling task's job object limit is too small. |
| E$MEM | 0002H | The memory of the calling task's job is exhausted. |
| E$NOT$CON-<br>FIGURED | 0008H | This call is not part of the present configuration. |

GET$DEFAULT$PREFIX returns the default prefix of a job.

```
connection = RQ$GET$DEFAULT$PREFIX(job, except$ptr);
```

## Input Parameter

| | |
|---|---|
| job | A TOKEN for the job whose default prefix is sought. A SELECTOR$OF(NIL) specifies the calling task's job. |

## Output Parameters

| | |
|---|---|
| connection | A TOKEN that receives a token for the connection object that is the default prefix for the designated job. |
| except$ptr | A POINTER to a WORD where the condition code will be returned. |

## Description

The GET$DEFAULT$PREFIX system call allows the caller to ascertain the default prefix for the specified job.

## Condition Codes

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$NOPREFIX | 8022H | You requested a default prefix, but no default prefix can be found for of one of the following reasons:<br>• When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default prefix.<br>• The job's directory can have entries but a default prefix is not cataloged there.<br>• The job parameter is not a token for an existing object.<br>• The prefix that is cataloged is not of the correct type. The default prefix must be a connection object or logical device object. (Logical device objects are created by the Extended I/O System.) |

| | | |
|---|---|---|
| | | • The job parameter contains a token for an object that is not a job. |
| E$NOT$CONFIGURED | 0008H | This system call is not part of the present configuration. |

GET$DEFAULT$USER returns the default user object of a job.

```
    user$id = RQ$GET$DEFAULT$USER(job, except$ptr);
```

## Input Parameter

| | |
|---|---|
| job | A TOKEN for the job whose default user object is sought. A SELECTOR$OF(NIL) specifies the calling task's job. |

## Output Parameters

| | |
|---|---|
| user$id | A TOKEN for the user object that is the default user for the designated job. |
| except$ptr | A POINTER to a WORD where the condition code will be returned. |

## Description

The GET$DEFAULT$USER system call allows the calling task to ascertain the default user object associated with the designated job.

## Condition Codes

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$NOUSER | 8021H | No default user can be found because of one of the following reasons:<br>• When this job was created, a size of zero was specified for its object directory, so the job cannot catalog a default user.<br>• The job's directory can have entries but a default user is not cataloged there.<br>• The object which is cataloged with the name R?IOUSER is not a user object. The name R?IOUSER should be treated as a reserved word.<br>• The job parameter contains a token for an object that is not a job. |

| | | |
|---|---|---|
| | | • The job parameter is not a token for an existing object. |
| E$NOT$CONFIGURED | 0008H | This system call is not part of the present configuration. |

The GET$GLOBAL$TIME system call reads the time of day from a battery-backed up hardware clock. This system call supports the Global Time of Day Clock on the iSBC 546 Terminal Communications Controller board, the MULTIBUS® II iCSM board, the iSBC 86C38 board, and the System 120.

```
CALL RQ$GET$GLOBAL$TIME (date$time$ptr, except$ptr);
```

## Input Parameters

None.

## Output Parameters

date$time$ptr — A POINTER to a structure in which the Basic I/O System returns the date and time information. The structure must have the following form:

```
DECLARE  date$time  STRUCTURE (
                         seconds     BYTE,
                         minutes     BYTE,
                         hours       BYTE,
                         days        BYTE,
                         months      BYTE,
                         years       WORD);
```

where

seconds — The current value of the seconds counter.

minutes — The current value of the minutes counter.

hours — The current value of the hours counter.

days — The current value of the days counter.

months — The current value of the month counter.

years — The current value of the year counter.

except$ptr — A POINTER to a WORD in which the Basic I/O System returns the condition code for this system call.

## Description

The GET$GLOBAL$TIME system call returns the date and time value stored in the hardware clock. The Basic I/O System accesses the appropriate registers on the hardware clock to read the global date and time values.

## Condition Codes

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | This call was made from an environment that did not contain a hardware clock. |
| E$NOT$CON-<br>FIGURED | 0008H | This system call is not part of the present configuration. |
| E$SHARE | 0028H | The hardware clock was busy (i.e., another task was accessing it). |
| E$SUPPORT | 0023H | The clock type specified in the configuration file is not a supported type. |

The GET$TIME system call returns the date/time value from the Basic I/O System's local clock.

```
    date$time = RQ$GET$TIME(except$ptr);
```

## Input Parameters

None.

## Output Parameters

date$time — A DWORD containing a date/time value expressed as the number of seconds since a fixed point in time.

except$ptr — A POINTER to a WORD where the condition code will be returned.

## Description

The GET$TIME system call returns the date/time value for the Basic I/O System. The Basic I/O System maintains the date/time value as the number of seconds since midnight, January 1, 1978. The iRMX UDI and Human Interface follow the convention that January 1, 1978 is equal to 0 seconds. When the date$time value reaches its maximum (0FFFFFFFFH), it will stop incrementing and will not roll over to start again from zero.

## Condition Codes

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$NOT$CONFIGURED | 0008H | This system call is not part of the present configuration. |

# INSPECT$USER

The INSPECT$USER system call returns a list of the IDs contained in a user object.

```
CALL RQ$INSPECT$USER(user, ids$ptr, except$ptr);
```

## Input Parameter

user — A TOKEN for the user object being inspected.

## Output Parameters

ids$ptr — A POINTER to a structure of the following form:

```
DECLARE  ids  STRUCTURE(
                 length  WORD,
                 count   WORD,
                 id(*)   WORD);
```

where

length — The upper limit on the number of IDs that are to be returned. (You must supply this when you invoke the system call. Zero values are not permitted.)

count — Actual number of IDs that are being returned.

id(*) — The IDs being returned.

except$ptr — A POINTER to a WORD where the condition code will be returned.

## Description

The INSPECT$USER system accepts a token for a user object and returns a list of the IDs in the user object.

The calling task must supply the length value in the data structure pointed to by the ids$ptr parameter. The BASIC I/O System fills in the remaining fields in that structure.

If the length value is smaller than the actual number of IDs in the user object, only the specified number of IDs will be returned.

## Condition Codes

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | The user parameter is not a token for an existing object. |
| E$NOT$CON-<br>FIGURED | 0008H | This system call is not part of the present configuration. |
| E$PARAM | 8004H | The length field contains a zero value. |
| E$TYPE | 8002H | The user parameter is a token for an object of the wrong type. |

# SET$DEFAULT$PREFIX

SET$DEFAULT$PREFIX sets the default prefix for an existing job.

```
    CALL RQ$SET$DEFAULT$PREFIX(job, prefix, except$ptr);
```

## Input Parameters

| | |
|---|---|
| job | A TOKEN for the job whose default prefix is to be set. A SELECTOR$OF(NIL) specifies the current job. |
| prefix | A TOKEN for the connection that is to become the default prefix. |

## Output Parameter

| | |
|---|---|
| except$ptr | A POINTER to a WORD where the condition code will be returned. |

## Description

The SET$DEFAULT$PREFIX system call sets the default prefix for an existing job. It does this by cataloging the connection (supplied as the prefix parameter) in the object directory of the job (supplied as the job parameter). The Basic I/O System catalogs the prefix under the name "$". If an object is already cataloged under the name "$", the Basic I/O System uncatalogs that object before cataloging the new prefix.

## Condition Codes

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$CONTEXT | 0005H | When this job was created, a size of zero was specified for the object directory, so a default prefix cannot be cataloged. |
| E$EXIST | 0006H | One or more of the following parameters is not a token for an existing object:<br>• The job parameter<br>• The prefix parameter |
| E$LIMIT | 0004H | The prefix parameter cannot be cataloged because the calling job's object directory is full. |
| E$NOT$CON-<br>FIGURED | 0008H | This system call is not part of the present configuration. |

| | | |
|---|---|---|
| E$TYPE | 8002H | At least one of the following is true:<br>• The job parameter is a token for an object that is not a job.<br>• The prefix parameter is a token for an object that is not of the correct type. It must be either a connection object or a logical device object. (Logical device objects are created by the Extended I/O System.) |

SET$DEFAULT$USER sets the default user object for a job.

## CAUTION

**This system call overrides the file access protection mechanism provided by the Basic I/O System. It should be used only by system programmers in charge of security management.**

```
CALL RQ$SET$DEFAULT$USER(job, user, except$ptr);
```

## Input Parameters

| | |
|---|---|
| job | A TOKEN for the job whose default user object is to be set. A SELECTOR$OF(NIL) designates the calling task's job. |
| user | A TOKEN for the user object that is to become the default user. |

## Output Parameter

| | |
|---|---|
| except$ptr | A POINTER to a WORD where the condition code will be returned. |

## Description

The SET$DEFAULT$USER system call sets the default user for an existing job.

## Condition Codes

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$CONTEXT | 0005H | When this job was created, a size of zero was specified for the object directory, so a default prefix cannot be cataloged. |
| E$EXIST | 0006H | One or more of the following parameters is not a token for an existing object:<br>• The job parameter<br>• The user parameter |
| E$LIMIT | 0004H | The user object cannot be cataloged because the calling job's object directory is full. |

|  |  |  |
|---|---|---|
| E$NOT$CONFIGURED | 0008H | This system call is not part of the present configuration. |
| E$TYPE | 8002H | The job or user argument is a token for an object of the wrong type. |

The SET$GLOBAL$TIME system call sets the battery backed-up hardware clock to a specified value. This system call supports the Global Time of Day Clock on the iSBC 546 Terminal Communications Controller board, the MULTIBUS® II iCSM board, the iSBC 86C38 board, and the System 120.

## CAUTION

**This system call overrides the global timing mechanism provided by the hardware clock. It should be used only by system programmers setting the initial system time.**

```
CALL RQ$SET$GLOBAL$TIME (date$time$ptr, except$ptr);
```

## Input Parameter

date$time$ptr

A POINTER to a structure that contains the date and time information to which the hardware clock is set. The structure must have the following form:

```
DECLARE date$time STRUCTURE (
                seconds BYTE,
                minutes BYTE,
                hours   BYTE,
                days    BYTE,
                months  BYTE,
                years   WORD);
```

where

seconds — The value to which the seconds counter is to be set. This value cannot exceed 59.

minutes — The value to which the minutes counter is to be set. This value cannot exceed 59.

hours — The value to which the hours counter is to be set. This value cannot exceed 23.

days — The value to which the days counter is to be set. This value cannot exceed 31.

months — The value to which the months counter is to be set. This value cannot exceed 12.

| | |
|---|---|
| years | The value to which the years counter is to be set. |

## Output Parameter

| | |
|---|---|
| except$ptr | A POINTER to a WORD in which the Basic I/O System returns the condition code for this system call. |

## Description

The SET$GLOBAL$TIME system call sets the global date and time values in the battery backed-up hardware clock. The Basic I/O System writes the new values into the appropriate registers on the clock board.

## Condition Codes

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$EXIST | 0006H | This call was made from an environment that did not contain a hardware clock. |
| E$NOT$CON-FIGURED | 0008H | This system call is not part of the present configuration. |
| E$PARAM | 8004H | One or more of the values specified in the date$time structure is illegal. |
| E$SHARE | 0028H | The global time-of-day clock was busy (i.e., another agent was accessing it). |
| E$SUPPORT | 0023H | The clock type specified in the configuration file is not a supported type. |

The SET$TIME system call sets the date and time for the Basic I/O System's local clock.

## CAUTION

**This system call overrides the timing mechanism provided by the Basic I/O System. It should be used only by system programmers setting the initial system time.**

```
CALL RQ$SET$TIME(date$time, except$ptr);
```

## Input Parameter

| | |
|---|---|
| date$time | A DWORD containing a date/time value expressed as the number of seconds since a fixed, user-determined point in time. |

## Output Parameter

| | |
|---|---|
| except$ptr | A POINTER to a WORD where the condition code will be returned. |

## Description

The SET$TIME system call sets the date/time value for the I/O system. The I/O System maintains the date/time value as a double word containing the number of seconds since a fixed point in time. Any time in the past can be used as the "beginning of time", but we recommend that you use 12:00 am (midnight), January 1, 1978. This convention is used by the Universal Development Interface and the Human Interface. When the date$time value reaches its maximum (0FFFFFFFFH), it will stop incrementing and will not roll over to start again from zero.

## Condition Codes

| | | |
|---|---|---|
| E$OK | 0000H | No exceptional conditions. |
| E$NOT$CON-FIGURED | 0008H | This system call is not part of the present configuration. |

WAIT$IO can be called following a call to A$READ, A$WRITE, or A$SEEK. When called, it returns to the calling task the concurrent condition code for the prior call. If applicable, WAIT$IO also returns the number of bytes read or written.

```
    actual = RQ$WAIT$IO(connection, resp$mbox, time$limit, except$ptr);
```

## Input Parameters

connection — A TOKEN for the connection that was specified as the connection in the prior asynchronous system call. (See Description.)

resp$mbox — A TOKEN for the mailbox that was specified as the response mailbox for the prior asynchronous system call. (See Description.)

time$limit — A WORD specifying the number of Nucleus system clock units that the task calling WAIT$IO is willing to wait for the I/O request/result segment (IORS) to arrive at the response mailbox. (For details on the IORS, see Appendix A.) A value of 0 means that the task is not willing to wait at all, and a value of 0FFFFH means that the task will wait indefinitely.

## Output Parameters

actual — A WORD to which the Basic I/O System returns the number of bytes read or written in the prior asynchronous system call. This value is undefined if the prior call was to A$SEEK. (See Description.)

except$ptr — A POINTER to a WORD where either the (concurrent) condition code for the prior asynchronous system call or the (sequential) condition code for the WAIT$IO system call is to be returned. (See Description.)

## Description

There are two ways in which a task calling A$READ, A$WRITE, or A$SEEK can receive the result of the concurrent portion of the call from the designated response mailbox. One way is for the task to wait at the mailbox, receive an IORS there, and extract the information from the segment. It is then incumbent upon the task to delete the segment, so that memory reserves are not needlessly depleted.

The other way for the task to receive this information is to call WAIT$IO. After the concurrent portion of the previous I/O call has been completed, the WAIT$IO system call returns the result of that call as follows:

- To the actual word, the number of bytes read or written, depending upon whether the previous call was to A$READ or A$WRITE, respectively. If the previous call was to A$SEEK, the value in the actual word is undefined.
- To the word pointed to by the except$ptr parameter, the concurrent condition code from the previous I/O call or the sequential condition code from the call to WAIT$IO. That is, if either of these condition codes is not E$OK, then that code is returned; if both of the condition codes are not E$OK, then the code that is returned is the code from the call to WAIT$IO. You should take note of the following:
    - There are three condition codes--E$LIMIT, E$MEM, and E$SUPPORT--that can be returned by either the sequential or the concurrent portion of a system call. However, WAIT$IO does not return any of these codes, so if one of them is returned, it came from the previous I/O call.
    - If the concurrent portion of the previous I/O call caused an E$IO exceptional condition, WAIT$IO does not return this code. Instead (in this case only), WAIT$IO determines the error by examining the lower four bits of the unit$status field of the IORS for the previous I/O call. Based on the content of the unit$status field, WAIT$IO returns one of the following condition codes (described under Condition Codes):

| Mnemonic | Value |
|---|---|
| E$IO$UNCLASS | 50H |
| E$IO$SOFT | 51H |
| E$IO$HARD | 52H |
| E$IO$OPRINT | 53H |
| E$IO$WRPROT | 54H |
| E$IO$NO$DATA | 55H |
| E$IO$MODE | 56H |
| E$IO$NO$SPARES | 57H |
| E$IO$ALT$ASSIGNED | 58H |

The benefit of WAIT$IO is that, in applications that use it, tasks do not always have to deal directly with IORSs. In particular, those tasks do not have to delete IORSs. Because of this, the Basic I/O System, in applications using WAIT$IO, maintains a supply of IORSs that can be used repeatedly. This means that performance is enhanced because the Basic I/O System does not have to create a segment every time an IORS is needed. This provides a significant advantage because A$READ, A$WRITE, and A$SEEK are typically the most commonly invoked Basic I/O System calls.

## Condition Codes

E$OK 0000H No exceptional conditions.

E$EXIST 0006H At least one of the following is true:

- The connection parameter or the resp$mbox parameter (or both) did not contain a token for an existing object.
- The specified connection or response mailbox (or both) was deleted.
- The token returned to the specified mailbox was for an object that had been deleted.

E$IO$HARD 0052H A hard I/O error occurred. Another retry is probably useless.

E$IO$MODE 0055H At least one of the following is true:

- A tape drive attempted to perform a read operation before the previous write operation completed.
- A tape drive attempted to perform a write operation before the previous read operation completed.

E$IO$NO$DATA 0056H A tape drive attempted to read the next record, but it found no data.

E$IO$OPRINT 0053H The device was off-line. Operator intervention is required.

E$IO$SOFT 0051H A soft I/O error occurred. The Basic I/O System tried to perform the operation a number of times (the number is configurable for each device). All attempts failed. If the configurable value specifying the number of retries is a reasonable value (for example, 9), another retry probably won't be successful either.

E$IO$UNCLASS 0050H An unknown type of I/O error occurred.

E$IO$WRPROT 0054H The asynchronous operation was A$WRITE and the volume was write-protected.

E$NOT$CONFIGURED 0008H This system call is not part of the present configuration.

| | | |
|---|---|---|
| E$TIME | 0001H | One of the following is true:<br>• The calling task was not willing to wait, and there was no IORS at the specified mailbox.<br>• The specified waiting period elapsed before the response mailbox received an IORS. |
| E$TYPE | 8002H | At least one of the following is true:<br>• The connection parameter is a token for an object that is not a file connection.<br>• The resp$mbox parameter is a token for an object that is not a mailbox.<br>• The object received at the response mailbox is not a segment or is a segment that is not an IORS. |

# I/O REQUEST/RESULT SEGMENT A

## A.1 OVERVIEW

Certain asynchronous Basic I/O System calls return a data structure called an I/O Request/Result Segment (IORS) to the mailbox specified by the resp$mbox parameter. The following system calls can return such a segment:

A$ATTACH$FILE
A$CHANGE$ACCESS
A$CLOSE
A$CREATE$DIRECTORY
A$CREATE$FILE
A$DELETE$CONNECTION
A$DELETE$FILE
A$OPEN
A$PHYSICAL$ATTACH$DEVICE
A$PHYSICAL$DETACH$DEVICE
A$READ
A$RENAME$FILE
A$SEEK
A$SPECIAL
A$TRUNCATE
A$UPDATE
A$WRITE

Before waiting at the response mailbox to receive the IORS, your application task should examine the condition code returned in the word pointed to by the except$ptr parameter. If this code is E$OK, the task can wait at the mailbox. However, if the code is not E$OK, an exceptional condition exists and nothing is sent to the mailbox.

Immediately after receiving the IORS, the task should examine the status field. This field contains an E$OK if the system call was completed successfully or an exceptional condition code if an error occurred. The result segment also contains the actual number of bytes read or written, if appropriate.

## A.2 STRUCTURE OF I/O REQUEST/RESULT SEGMENT

The IORS is structured as follows:

```
DECLARE   iors         STRUCTURE(
    status             WORD,
    unit$status        WORD,
    actual             WORD);
```

where

status — Condition code indicating the outcome of the call.

unit$status — The lower four bits of this field contain device-dependent error code information that is meaningful only if the status is E$IO. Certain devices also use the upper 12 bits of unit$status to provide more information about the error (for details, see Appendix E in the *iRMX® Device Drivers User's Guide*). The codes, meanings, and associated mnemonics for the lower four bits are as follows:

| Code | Mnemonic | Meaning |
|---|---|---|
| 0 | IO$UNCLASS | An error occurred for which it was impossible to ascertain the cause. |
| 1 | IO$SOFT | Soft error; the I/O system has retried the operation and failed; another retry is not possible. |
| 2 | IO$HARD | Hard error; a retry is not possible. |
| 3 | IO$OPRINT | Operator intervention is required. |
| 4 | IO$WRPROT | Write-protected volume. |
| 5 | IO$NO$DATA | No data on the next tape record. |
| 6 | IO$MODE | A read (or write) was attempted before the previous write (or read) completed. |
| 7 | IO$NO$SPARES | An I/O error occurred during disk formatting; no alternate tracks were available. |
| 8 | IO$ALT$AS-<br>SIGNED | An I/O error occurred during disk formatting; an alternate track was assigned. |

actual — The actual number of bytes transferred.

The IORS contains other fields which are of interest only to the designer of a device driver. Refer to the *iRMX® Device Drivers User's Guide* for information about the remaining fields in the IORS.

# INDEX

## A

A$ATTACH$FILE 1-6
  iRMX®-NET considerations 1-7
A$CHANGE$ACCESS 1-11
  iRMX-NET considerations 1-14
A$CLOSE 1-19
A$CREATE$DIRECTORY 1-22
  iRMX-NET considerations 1-23
A$CREATE$FILE 1-28
  iRMX-NET considerations 1-31
A$DELETE$CONNECTION 1-36
A$DELETE$FILE 1-39
  iRMX-NET considerations 1-40
A$GET$CONNECTION$STATUS 1-45
  iRMX-NET considerations 1-47
A$GET$DIRECTORY$ENTRY 1-49
  iRMX-NET considerations 1-50
A$GET$EXTENSION$DATA 1-52
A$GET$FILE$STATUS 1-55
  device functions supported 1-58
  diskette drive flags 1-58
  file types supported on a device 1-57
  named file portion of status structure 1-56
  status information structure 1-55
  volume information flags 1-61
A$GET$PATH$COMPONENT 1-64
  file name segment structure 1-64
A$GET$PHYSICAL$ATTACH$DEVICE 1-72
A$OPEN 1-68
  iRMX-NET considerations 1-69
  mode values 1-68
  share mode values 1-68
A$PHYSICAL$ATTACH$DEVICE
  file$driver field values 1-72
A$PHYSICAL$DETACH$DEVICE 1-76

## A (continued)

A$READ 1-79
  iRMX-NET considerations 1-80
A$RENAME$FILE 1-83
  iRMX-NET considerations 1-84
A$SEEK 1-89
  mode field values 1-89
A$SET$EXTENSION$DATA 1-92
  extension data structure 1-92
A$SPECIAL 1-95
  bad track or sector information 1-117
  buffered device fields 1-113
  changing volumes 1-99
  connection flags for terminal attributes 1-104
  formatting a track structure 1-97
  functions 1-95
  hard disk information structure 1-100
  iRMX-NET considerations 1-96
  iSBC® 188/48 1-111
  iSBC 544A 1-111
  set font 1-116
  signal characters 1-115
  signal semaphore structure 1-115
  stream file information 1-97
  stream file transactions 1-98
  tape drive functions 1-116
  tape information 1-101
  tape information structure 1-101
  TCC-supported device fields 1-114
  terminal attributes 1-101
  terminal flags field 1-107
  terminal information structure 1-102
  terminal status 1-118
  terminal status structure 1-119
  volume unavailable notification 1-98
  volume unavailable structure 1-99
A$TRUNCATE 1-129
A$UPDATE 1-132
A$WRITE 1-136
  iRMX-NET considerations 1-137

## B

Bad track or sector information 1-117

C

Changing volumes 1-99
Connection access byte 1-46
Connection mode values 1-46
Connection status structure 1-45
CREATE$USER 1-140

D

Data Encryption Standard (DES) 1-143
Data file access rights 1-12
DELETE$USER 1-142
Deleting a file 1-36
Directory access rights 1-13, 1-22
Directory entry segment structure 1-49

E

ENCRYPT 1-143
Encrypted password
  length 1-143
Encryption standard 1-143

F

Federal Information Processing Standard Publication #46 1-143
File access mask 1-29
File driver values 1-45
File types supported on a device 1-57

G

GET$DEFAULT$PREFIX 1-145
GET$DEFAULT$USER 1-147
GET$GLOBAL$TIME 1-149
GET$TIME 1-151

I

I/O Request/Result Segment (IORS) A-1
INSPECT$USER 1-152
IORS A-1
iRMX-NET considerations
  A$ATTACH$FILE 1-7
  A$CHANGE$ACCESS 1-14
  A$CREATE$DIRECTORY 1-23
  A$CREATE$FILE 1-31

**I (continued)**

iRMX-NET considerations (continued)
- A$DELETE$FILE 1-40
- A$GET$CONNECTION$STATUS 1-47
- A$GET$DIRECTORY$ENTRY 1-50
- A$OPEN 1-69
- A$READ 1-80
- A$RENAME$FILE 1-84
- A$SPECIAL 1-96
- A$WRITE 1-137

**R**

Restoring volume availability 1-100

**S**

Set font function 1-116
SET$DEFAULT$PREFIX 1-154
SET$DEFAULT$USER 1-156
SET$GLOBAL$TIME 1-158
SET$TIME 1-160
Share mode values 1-46
Signal characters 1-115
Stream file information 1-97
Stream file transactions 1-98
Structure
- bad track or sector information 1-117
- connection status 1-45
- date and time for GET$GLOBAL$TIME 1-149
- date and time for SET$GLOBAL$TIME 1-158
- directory entry segment 1-49
- extension data 1-92
- file name segment 1-64
- file status information 1-55
- formatting a track 1-97
- hard disk information 1-100
- named file portion of status structure 1-56
- signal semaphore for keyboard characters 1-115
- terminal information 1-102
- terminal status 1-119
- user ID 1-140
- user object for INSPECT$USER 1-152
- volume unavailable 1-99

System call dictionary 1-3

T

Tape drive functions 1-116
Tape information 1-101
Terminal attributes 1-101
Terminal status 1-118
Time
  beginning convention 1-160
  getting 1-151
  getting from battery backed-up clock 1-149
  setting 1-160
  setting battery backed-up clock 1-158

U

Updating files
  calling A$UPDATE 1-132
  differences 1-133
  fixed updating 1-133
  timeout updating 1-133
User ID access mask 1-11

V

Volume unavailable notification 1-98

W

WAIT$IO 1-161
Winchester disks, obtaining information 1-100
Writing bad track/sector information 1-118

intel®

iRMX® Basic I/O System Cal
Reference Manu
462915-00

# REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative.

1. Please describe any errors you found in this publication (include page number).

______________________________
______________________________
______________________________
______________________________

2. Does this publication cover the information you expected or required? Please make suggestions for improvement.

______________________________
______________________________
______________________________
______________________________

3. Is this the right type of publication for your needs? Is it at the right level? What other types of publications are needed?

______________________________
______________________________
______________________________
______________________________

4. Did you have any difficulty understanding descriptions or wording? Where?

______________________________
______________________________
______________________________

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating). ____________

NAME ______________________________ DATE ____________
TITLE ______________________________
COMPANY NAME/DEPARTMENT ______________________________
ADDRESS ______________________________ PHONE ( ) ____________
CITY ______________ STATE ______________ ZIP CODE ____________
(COUNTRY)

Please check here if you require a written reply. □

## VE'D LIKE YOUR COMMENTS . . .

his document is one of a series describing Intel products. Your comments on the back of this form will elp us produce better manuals. Each reply will be carefully reviewed by the responsible person. All omments and suggestions become the property of Intel Corporation.

you are in the United States, use the preprinted address provided on this form to return your omments. No postage is required. If you are not in the United States, return your comments to the Intel ales office in your country. For your convenience, international sales office addresses are printed on ie last page of this document.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO. 79 HILLSBORO, OR

POSTAGE WILL BE PAID BY ADDRESSEE

**Intel Corporation**
**OMSO Technical Publications, MS: HF3-72**
**5200 N.E. Elam Young Parkway**
**Hillsboro, OR 97124-9978**

# INTERNATIONAL SALES OFFICES

INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, California 95051

BELGIUM
Intel Corporation SA
Rue des Cottages 65
B-1180 Brussels

DENMARK
Intel Denmark A/S
Glentevej 61-3rd Floor
dk-2400 Copenhagen

ENGLAND
Intel Corporation (U.K.) LTD.
Piper's Way
Swindon, Wiltshire SN3 1RJ

FINLAND
Intel Finland OY
Ruosilante 2
00390 Helsinki

FRANCE
Intel Paris
1 Rue Edison-BP 303
78054 St.-Quentin-en-Yvelines Cedex

ISRAEL
Intel Semiconductors LTD.
Atidim Industrial Park
Neve Sharet
P.O. Box 43202
Tel-Aviv 61430

ITALY
Intel Corporation S.P.A.
Milandfiori, Palazzo E/4
20090 Assago (Milano)

JAPAN
Intel Japan K.K.
Flower-Hill Shin-machi
1-23-9, Shinmachi
Setagaya-ku, Tokyo 15

NETHERLANDS
Intel Semiconductor (Netherland B.V.)
Alexanderpoort Building
Marten Meesweg 93
3068 Rotterdam

NORWAY
Intel Norway A/S
P.O. Box 92
Hvamveien 4
N-2013, Skjetten

SPAIN
Intel Iberia
Calle Zurbaran 28-IZQDA
28010 Madrid

SWEDEN
Intel Sweden A.B.
Dalvaegen 24
S-171 36 Solna

SWITZERLAND
Intel Semiconductor A.G.
Talackerstrasse 17
8125 Glattbrugg
CH-8065 Zurich

WEST GERMANY
Intel Semiconductor G.N.B.H.
Seidlestrasse 27
D-8000 Munchen

INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, California 95051
(408) 987-8080