iRMX®
System Call Reference

Order Number: 469157-004

In the United States, additional copies of this manual or other Intel literature may be obtained by writing:

Literature Distribution Center
Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

Or you can call the following toll-free number: 1-800-548-4725

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 printed on the last page of this document. Contact your local sales office to obtain the latest specifications before placing your order.

Intel Corporation (Intel) 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 assumes no responsibility for any errors that may appear in this document. Intel makes no commitment to update nor to keep current the information contained in this document. No part of this document may be copied or reproduced in any form or by any means without prior written consent of Intel. Intel retains the right to make changes to these specifications at any time, without notice.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's Software License Agreement.
U.S. GOVERNMENT RESTRICTED RIGHTS: These software products and documentation were developed at private expense and are provided with "RESTRICTED RIGHTS." Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR 52.227-14 and DFAR 252.227-7013 et seq. or its successor.

The Intel logo, i960, Pentium, and iRMX are registered trademarks of Intel Corporation, registered in the United States of America and other countries. Above, i287, i386, i387, i486, Intel287, Intel386, Intel387, Intel486, Intel487 and EtherExpress are trademarks of Intel Corporation.

Adaptec is a registered trademark of Adaptec, Inc. AT, IBM and PS/2 are registered trademarks and PC/XT is a trademark of International Business Machines Corporation. All Borland products are trademarks or registered trademarks of Borland International, Inc. CodeView, Microsoft, MS, MS-DOS and XENIX are registered trademarks of Microsoft Corporation. Comtrol is a registered trademark and HOSTESS is a trademark of Comtrol Corporation. DT2806 is a trademark of Data Translation, Inc. Ethernet is a registered trademark of Xerox Corporation. Hayes is a registered trademark of Hayes Microcomputer Products. Hazeltine and Executive 80 are trademarks of Hazeltine Corporation. Hewlett-Packard is a registered trademark of Hewlett-Packard Co. Maxtor is a registered trademark of Maxtor Corporation. MIX® is a registered trademark of MIX Software, Incorporated. MIX is an acronym for Modular Interface eXtension. MPI is a trademark of Centralp Automatismes (S.A.). NetWare and Novell are registered trademarks of Novell Corp. NFS is a trademark of Sun Microsystems, Inc. Phar Lap is a trademark of Phar Lap Software, Inc. Soft-Scope is a registered trademark of Concurrent Sciences, inc. TeleVideo is a trademark of TeleVideo Systems, Inc. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited. VAX is a registered trademark and VMS is a trademark of Digital Equipment Corporation. Visual Basic and Visual C++ are trademarks of Microsoft Corporation. All Watcom products are trademarks or registered trademarks of Watcom International Corp. Windows, Windows 95 and Windows for Workgroups are registered trademarks and Windows NT is a trademark of Microsoft in the U.S. and other countries. Wyse is a registered trademark of Wyse Technology. Zentec is a trademark of Zentec Corporation. Other trademarks and brands are the property of their respective owners.

Copyright © 1991, 1992, 1993 and 1995 Intel Corporation, All Rights Reserved

REVISION HISTORY		**DATE**
-001	Original Issue	12/91
-002	Revision One	08/92
-003	Update for Release 2.1 of the OS	12/93
-004	Update for Release 2.2 of the OS	11/95

Quick Contents

Notational Conventions

Most of the references to system calls in the text and graphics use C syntax instead of PL/M (for example, the system call **send_message** instead of **send$message**). If you are working in C, you must use the C header files, *rmx_c.h*, *udi_c.h* and *rmx_err.h*. If you are working in PL/M, you must use dollar signs ($) and use the *rmxplm.ext* and *error.lit* header files.

This manual uses the following conventions:

- Syntax strings, data types, and data structures are provided for PL/M and C respectively.
- All numbers are decimal unless otherwise stated. Hexadecimal numbers include the `H` radix character (for example, `0FFH`). Binary numbers include the `B` radix character (for example, `11011000B`).
- Bit 0 is the low-order bit. If a bit is set to 1, the associated description is true unless otherwise stated.
- `Data structures and syntax strings appear in this font.`
- **System call names and command names appear in this font.**
- PL/M data types such as BYTE and SELECTOR, and iRMX data types such as STRING and SOCKET are capitalized. All C data types are lower case except those that represent data structures.
- The following OS layer abbreviations are used. The Nucleus layer is unabbreviated.

AL	Application Loader
BIOS	Basic I/O System
EIOS	Extended I/O System
HI	Human Interface
UDI	Universal Development Interface

- Whenever this manual describes I/O operations, it assumes that tasks use BIOS calls (such as **rq_a_read**, **rq_a_write**, and **rq_a_special**). Although not mentioned, tasks can also use the equivalent EIOS calls (such as **rq_s_read**, **rq_s_write**, and **rq_s_special**) or UDI calls (**dq_read** or **dq_write**) to do the same operations.

Contents

1 Introduction

2 Application Loader System Calls

3 Basic I/O System Calls

4 Extended I/O System Calls

5 Human Interface System Calls

6 Nucleus System Calls

6 Nucleus System Calls (continued)

7 UDI System Calls

8 Windows- and DOS-Specific System Calls

9 Kernel System Calls and Handlers

10 Virtual Memory System Calls

A Application Loader Examples

B Nucleus Examples

C UDI Examples

D Condition Codes

Index

Service Information

Tables

Figures

Introduction 1

This manual is a reference to the system calls for the iRMX® III Operating System, iRMX for PCs, and iRMX for Windows. It provides a detailed description of each system call and syntax in both PL/M and C languages. System calls can also be invoked from other languages.

See also: Specific language information, *Programming Techniques*

This chapter provides general information that applies to the system calls:

- Definitions of data types for PL/M and C
- Header files (include files)
- Interface libraries for system calls and the C library functions
- Layer-specific information for the Application Loader, BIOS, EIOS, and Kernel
- Tables summarizing the calls in each Operating System (OS) layer

Reader Level

This manual assumes that you are familiar with:

- Terms and concepts of the iRMX OS

  See also: *Introducing the iRMX Operating Systems,*
  System Concepts

- The PL/M or C programming language

  See also: *PL/M 386 Programmer's Guide,*
  iC-386 Compiler User's Guide

The Human Interface chapter also assumes that you are familiar with:

- Human Interface command parsing

  See also: *System Concepts*

- Human Interface command format

  See also: *Command Reference*

Call Prefixes For Various Layers

A number of prefixes are used with iRMX system calls to designate functions or OS layers. This list presents the prefix designations, examples of system call names using those prefixes, and the use of the prefix.

Prefix	**Example**	**Prefix Usage**
rq_	rq_delete_job	Basic label for Nucleus, BIOS, EIOS, AL
rqe_	rqe_offspring	Basic label for extended system calls
rqv_	rqv_allocate	Basic label for virtual memory system calls
a_	rq_a_load	Label for asynchronous (rq_a, rqe_a) calls
s_	rq_s_overlay	Label for synchronous (rq_s, rqe_s) calls
c_	rq_c_get_char	Label for Human Interface (rq_c) calls
dq_	dq_allocate	Basic label for UDI system calls
cq_	cq_comm_rb	Basic label for iNA 960 network calls
KN_	KN_delete_alarm	Basic label for Kernel calls
KNE_	KNE_get_time	Basic label for extended Kernel calls

Modified Alphabetical Listing of Calls

This manual uses a shorthand notation that omits the basic **rq_** prefix. For example, the call **rq_s_create_file** is shown as **s_create_file**. You must use the full name in application programs.

Extended system calls begin with the prefix **rqe**. For extended calls, this manual spells out the complete names, including the **rqe** prefix, for example **rqe_create_io_job**. The **dq**, **cq**, **KN**, and **KNE** prefixes are also spelled out.

Within their OS layer, system call descriptions are presented in alphabetical order according to their basic names, without regard to the standard **rq_** prefix. For example, **rq_create_io_job** is listed alphabetically as **create_io_job**. Extended system calls are also arranged by their basic names but the **rqe** prefix is retained for uniqueness. For example, **rqe_create_io_job** (including the **rqe** prefix) follows **create_io_job**. The same is true for the **dq**, **cq**, **KN**, and **KNE** prefixes.

Condition Codes

Except for Kernel calls, which do not perform error checking, each system call returns a condition code whenever it is invoked. If the call executes without error, it returns the condition code E_OK. (Some iNA 960 cq_* calls can return a value other than E_OK to indicate success.) If an error occurs, the call returns a condition code that describes the error. Your application can handle the condition code directly (in-line) or with an exception handler.

See also: Condition codes, exception handlers, *System Concepts*

The typical condition codes returned by each call are listed in each system call description. However, be aware that:

- PL/M programs use a $ instead of an _ (underscore) in the condition code mnemonic.
- Condition codes can percolate up to outer layers of the OS from inner layers. For example, an HI call can produce exception codes from the BIOS or EIOS. In that case, the condition code is not listed in the HI call description.

See also:	Condition code master list, Appendix D

Data Types

Except for Kernel calls, each system call description lists PL/M and C data types for each call parameter. The data types, unless otherwise stated, define the acceptable range of values for a parameter. Table 1-1 lists the data types used in this manual. Data types such as WORD_16, WORD_32, and NATIVE_WORD are iRMX data types, not native to PL/M or C; they are defined in the include files provided with the OS.

See also: *rmxtypes.h* and *rmx_c.h* files in the *intel/include* directory

⚠ **CAUTION**

Compiler controls (such as `long64` in iC-386) allow certain data types to be larger than specified here. Use only the compiler option that provides data types conforming with the table below.

See also: `long64`, *iC-386 Compiler User's Guide*

Table 1-1. Data Types in System Calls

<table>
<tr><th>C Data Type</th><th>PL/M</th><th colspan="3">Description</th></tr>
<tr><td>UINT_8</td><td>BYTE</td><td colspan="3">An unsigned 8-bit binary number or character in the range of 0 to 255, contained in 1 byte of memory.</td></tr>
<tr><td>UINT_16</td><td>WORD_16</td><td colspan="3">An unsigned 16-bit binary number in the range of 0 to 65535, contained in 2 contiguous bytes of memory.</td></tr>
<tr><td>UINT_32</td><td>WORD_32</td><td colspan="3">An unsigned 32-bit binary number in the range of 0 to 4,294,967,295, contained in four contiguous bytes of memory.</td></tr>
<tr><td>SELECTOR</td><td>SELECTOR</td><td colspan="3">A 16-bit index identifying a particular memory segment in a descriptor table (segmented application) or page tables (flat model). The selector is the data type for a token, which is a value that the OS assigns to an object.</td></tr>
<tr><td>data_type far *</td><td>POINTER</td><td colspan="3">In C, the data_type can be any data type in this table, or a data structure defined in the call description, or void. The asterisk (*) is part of the name. Pointer types and sizes are:</td></tr>
<tr><td></td><td></td><td><b>Compiler Type</b></td><td><b>Pointer Type</b></td><td><b>Pointer Size</b></td></tr>
<tr><td></td><td></td><td>16-bit compact/large segmented</td><td>segment:offset</td><td>16:16
(32 bits total length)</td></tr>
<tr><td></td><td></td><td>32-bit compact segmented
(e.g., PL/M-386, iC-386)</td><td>segment:offset</td><td>16:32
(48 bits total length)</td></tr>
<tr><td></td><td></td><td>32-bit flat
(non-Intel C compilers)</td><td>offset only</td><td>32 bits (near pointer even if declared far)</td></tr>
</table>

continued

Table 1-1. Data Types in System Calls (continued)

C Data Type	PL/M Data Type	Description
SOCKET_STRUCT	SOCKET$STRUCT	Combination of a host ID and port ID for use in message passing.
RMX_STRING or PLM_STRING_STRUCT	PLM_STRING_STRUCT	An array of consecutive characters with the first character defining the length of the string.
STRING_TABLE_STRUCT	PLM_STRINGTABLE_STRUCT	An array of consecutive RMX_STRINGs or PLM_STRING_STRUCTs.
BOOLEAN	BYTE	This data type corresponds to BOOLEAN logic (true or false). It is an unsigned 8-bit binary number that can take on the values FALSE (0) and TRUE (not 0 or any value greater than 0). In PL/M, TRUE must have bit 0 set to 1.
KN_TOKEN	WORD_32	An unsigned 32-bit binary number in the range of 0 to 4,294,967,295, contained in four contiguous bytes of memory.
KN_STATUS	WORD_32	An unsigned 32-bit binary number in the range of 0 to 4,294,967,295, contained in four contiguous bytes of memory.
KN_FLAGS	WORD_32	An unsigned 32-bit binary number in the range 0 to 4,294,967,295, contained in four contiguous bytes of memory.
NATIVE_WORD	(no equivalent)	In C, expands type definitions of variables from 16 bits to 32 bits when using 32-bit code. The NATIVE_WORD type can be either an unsigned 16-bit or unsigned 32-bit binary number. In PL/M, you include either the 16-bit or 32-bit version of header (.ext and .lit) files to get the correct data type.

Constants

Among others, these constant values are defined:

Value	Defined as
0	FALSE
0FFH	TRUE

SOCKET Definition

The SOCKET$STRUCT data type is defined in PL/M as:

```
DECLARE SOCKET$STRUCTURE(
host_id                  WORD_16,
port_id                  WORD_16);
```

For C, it is structured as:

```
struct {
   UINT_16               host_id;
   UINT_16               port_id;
} SOCKET_STRUCT;
```

Where:

`host_id` A number from 0 to 19, which is the slot number of a Multibus II board, identifying a message-passing host.

`port_id` A number that uniquely identifies a port on the host.

See also: Nucleus call **create_port**, in this manual,
Sockets and ports, *System Concepts*

Strings and String Table Format

The iRMX OS uses structures called strings to store groups of ASCII characters, such as pathnames. The OS assumes a string to be a series of consecutive bytes broken into two portions: a count byte and text bytes. The first byte in the string is the count byte; its value is set to the number of bytes in the text portion of the string. The text bytes contain the substance of the string. The maximum number of characters in the STRING data type is 255.

Note
When you call C functions, as in the C Library or the TCP/IP socket calls, you use the null-terminated string that is typical of the C language. When you make iRMX system calls from C (or any language), you must use the OS string type described here.

The OS also uses another structure called a string table. A string table consists of a count byte and a series of consecutive strings. As with the string, the first byte in the string table is the count byte; its value is set to the number of strings in the string table. Figure 1-1 shows the string table format.

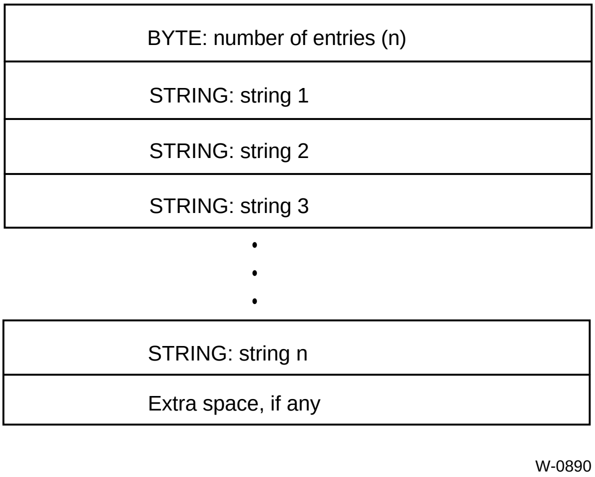

Figure 1-1. String Table Format

STRING Definition

The iRMX OS STRING data type is not the same as the null-terminated string commonly used in C programs. The STRING data type is defined in PL/M as:

```
STRING                    LITERALLY 'STRUCTURE(
length                    BYTE,
char (STRING$MAX)         BYTE)';
```

```
DECLARE  PLM_STRING_STRUCT(
   length          BYTE,
   char(*)         BYTE);
```

or in C:

```
typedef struct {
   UINT_8          length;
   UINT_8          text [_MAX_STRING];
} RMX_STRING;
```

Where:

`length` Specifies the length of the string. This equals the index of the character array. 0 specifies a null string.

`text [_MAX_STRING]`
The character array. In C, adjust the index for _MAX_STRING from 255 to fit the maximum value of `length` in actual use (≤ 255).

String Table Definition

The STRINGTABLE data type is defined in PL/M as:

```
DECLARE STRINGTABLE STRUCTURE(
   count                 BYTE,
   strings(_NUM_STRINGS)  STRING)
```

Where:

`count` Specifies the number of entries in the STRINGTABLE.

`strings[_NUM_STRINGS]`
The number of strings in the table.

or in C:

```
typedef struct {
   UINT_8                numentry;
   PLM_STRING_STRUCT     strings[_NUM_STRING];
} STRING_TABLE_STRUCT;
```

Where:

`numentry` Specifies the number of entries in the STRINGTABLE.

`strings[_NUM_STRING]`
The number of strings (of type `PLM_STRING_STRUCT`) in the table.

Underscores in Calls, Structures, and Data Types

This manual refers to all calls and data types such as structure definitions with names that include underscores (_) separating the parts of the name. (In PL/M, dollar signs ($) separate the parts of system call names.) In some cases, you can refer to the same system call or structure definition with or without the underscore separator. For example, you can call **rq_send_message** or **rqsendmessage**, depending on the include file. The OS defines such calls and data types both ways; the versions without underscores are provided for backwards compatibility with existing code.

As a general rule, the data types with underscores are defined in header files that have underscores in the names. For example, *rmx_c.h* defines system calls with underscores; it also includes *rmxc.h*, which defines system calls without underscores. Similarly, *rmx_err.h* defines condition code names with underscores, while *rmxerr.h* defines the same names without underscores. (The header files, or include files, are described later in this chapter.)

However, some of the latest OS type definitions are defined only with underscore separators, as shown in this manual. In your application program, include the underscore version of include files and use the underscores as shown in this manual.

⚠ **CAUTION**

Not all type definitions that include underscores are exactly the same as their counterparts that don't include underscores. For example, the `STRING_TABLE_STRUCT` structure (see page 24) is not defined the same way as its counterpart, `STRINGTABLESTRUCT`.

Furthermore, not all of the PL/M structure definitions listed in this manual are actually defined in PL/M header files (.lit and .ext files). If you program in C and include the correct header files, you can use the type definitions listed in this manual without defining them yourself. But if you program in PL/M, you may need to declare some of the literal structures listed here.

Header Files to Include for System Calls

The header files to include in your application programs are located in the directories listed below. These files define the prototypes for system calls, data types shown in this manual, and mnemonics for condition codes.

Compiler Type	**Directory**
C	*/intel/include*
PL/M 32-bit	*/rmx386/inc*
PL/M 16-bit	*/rmx386/inc16*

Most references to system calls in this manual use C syntax instead of PL/M (for example, the system call **rq_send_message** instead of **rq$send$message**). The header file you include determines whether system calls and data types are defined with an underscore (_) as shown in this manual. In PL/M, use dollar signs ($) in system calls and condition code mnemonics.

Table 1-2 lists the general include files and files that are specific to layers of the OS. The general include files include most of the layer-specific files, so you don't have to specifically include all these files in your application.

Table 1-2. Include Files for System Calls and Data Types

General Include Files	**C, Underscores**	**C, No Underscores**	**PL/M**
Most layers	rmx_c.h	rmxc.h	rmxplm.ext rmxplm.lit
Condition codes	rmx_err.h	rmxerr.h	error.lit
Layer-Specific Files	**C, Underscores**	**C, No Underscores**	**PL/M**
OS data types, constants	common.h	common.h	common.lit
Application Loader			loader.ext
BIOS			bios.ext
EIOS			eios.ext
Human Interface			hi.ext
Nucleus	nucleus.h	nucleus.h	nuclus.ext
Kernel [1]	rmk.h		rmk_base.ext rmk_base.lit
UDI [1]	udi_c.h	udi.h	udi.ext
iNA 960 cq_ calls [1]	cqcomm.h cq*.h	cqcomm.h	cqcomm.ext [2] cq*.lit [2]

[1] Most layer-specific files are already included in the general include files, but you must specifically include these files.

[2] These PL/M files are for 16-bit applications only; they are in the /rmx386/inc16 directory but not in /rmx386/inc.

The directories listed above contain other include files for specific purposes. The include files for C Library functions are in the */intel/include* directory.

See also: Header files, *C Library Reference*

Interface Libraries for System Calls and C Library

Libraries supplied with the OS provide a standard interface to the system calls. These are the libraries to which you bind (link) your application. Procedures in the interface libraries perform the operations needed to invoke the actual system call, depending on the compiler you use.

Tables 1-3 and 1-4 list the system call interface libraries for the various supported compiler models. These interface libraries are located in the */rmx386/lib* directory. The libraries in Table 1-3 are interfaces to these layers of the OS:

Application Loader
BIOS
EIOS
Human Interface
Nucleus
Paging Subsystem
iNA 960 cq_* calls
Kernel

⚠ **CAUTION**

Interfaces to the iNA 960 calls were formerly in separate libraries: */rmx386/rmxnet/cq*.lib*. As of release 2.2 of the OS, interfaces for iNA 960 calls are defined in the libraries listed in Table 1-3. The old libraries and the directory they were in no longer exist. You must relink your existing applications that make cq_ calls to one of the libraries in Table 1-3.

The libraries in Table 1-3 for non-Intel compilers include an interface for Kernel calls. For 32-bit compact applications that make Kernel calls using Intel compilers, you must also link to the *kn_call.lib* library.

Table 1-3. Interface Libraries for All Calls Except UDI

Interface Type	Intel iC-386 and PL/M	Borland C Compiler	Microsoft C Compiler	Watcom C Compiler
16-bit compact	rmxifc.lib [1]	rmxifcb.lib	rmxifcm.lib	
16-bit large	rmxifl.lib [1]	rmxiflb.lib	rmxiflm.lib	
32-bit compact	rmxifc32.lib [2] kn_call.lib			rmxifc3w.lib
32-bit flat		rmxiff3b.lib	rmxiff3m.lib	N/A

[1] These libraries do not include an interface to Kernel calls.

[2] This library does not include an interface to Kernel calls. You must also link to kn_call.lib, which supports only 32-bit compact applications.

Table 1-4. Interface Libraries for UDI Calls

Interface Type	Intel iC386 and PL/M	Borland C Compiler	Microsoft C Compiler	Watcom C Compiler
16-bit compact	udiifc.lib	udiifcb.lib	udiifcm.lib	
16-bit large	udiifl.lib	udiiflb.lib	udiiflm.lib	
32-bit compact	udiifc32.lib			udiifc3w.lib
32-bit flat		udiiff3b.lib	udiiff3m.lib	

Table 1-5 lists the interface libraries for C library functions. These libraries are located in the \intel\lib directory Link your application to the appropriate library according the compiler you use. There is also a PL/M-specific library, */intel/lib/plm386.lib*, for any application written in PL/M.

See also: *C Library Reference* for information on the C functions

Table 1-5. Interface Libraries for C Library Functions

Interface Type	Intel iC386 Compiler	Borland C Compiler	Microsoft C Compiler	Watcom C Compiler
16-bit compact		cifcb.lib	cifcm.lib	
16-bit large		ciflb.lib	ciflm.lib	
32-bit compact	cifc32.lib			cifc32w.lib
32-bit large	cifl32.lib			
32-bit flat		ciff3b.lib	ciff3m.lib	

Layer-specific Information

This section presents information that applies specifically to the AL, BIOS, EIOS, and Kernel layers:

- AL-specific information relates to synchronous and asynchronous condition codes, file access requirements, mailboxes, and Loader Result Segments
- BIOS-specific information relates to call types, sequential and concurrent condition codes, and I/O Request/Result Segments
- EIOS-specific information relates to file and call types
- Kernel-specific information relates to syntax, description types, and parameters

Application Loader Layer-specific Information

There are three types of AL calls, as indicated by these prefixes:

Prefix	**Meaning**
rq_a_	Asynchronous call. The calling task continues running while the loading operation is in process.
rq_s_	Synchronous call. The calling task is suspended during the loading operation.
rqe_	Extended call. This call involves addressability of more than 1 Mbyte. It can be designated with the asynchronous or synchronous prefix.

Condition Codes For Synchronous System Calls

For system calls that are synchronous (**s_load_io_job**, **rqe_s_load_io_job**, and **s_overlay**), the AL returns a single condition code each time the call is invoked. Your system's exception handler receives this code when an exceptional condition occurs.

Condition Codes For Asynchronous System Calls

For system calls that are asynchronous (**a_load**, **a_load_io_job**, **rqe_a_load_io_job**), the AL returns two condition codes each time the call is invoked. Your task must process these two condition codes separately:

- One code is returned after the sequential part of the system call is executed.
- The other code is returned after the concurrent part of the call is executed.

See also: Sequential and concurrent portions of asynchronous system calls, *System Concepts*

File Access Requirements

The AL does not need exclusive access to the file being loaded. However, other tasks sharing the file are affected:

- The other tasks must not share the connection passed to the AL, but must obtain their own connections to the file.
- The AL specifies share with readers only when opening the connection; during the loading operation, other tasks can access the file only for reading.

Mailboxes and Loader Result Segments

Your task must specify a mailbox when invoking an asynchronous system call in order to receive a Loader Result Segment (LRS). Three AL system calls described in this manual are asynchronous: **a_load**, **a_load_io_job**, and **rqe_a_load_io_job**.

Do not use the same response mailbox for more than one concurrent invocation of asynchronous system calls, because the AL can return LRSs in an order different from the order of invocation. It is safe to use the same mailbox for multiple invocations of asynchronous system calls if only one task invokes the calls and that task always obtains the result of one call (using the **receive_message** Nucleus call) before making the next call.

The LRS indicates the result of the loading operation, but the LRS format depends on which system call was invoked and whether the calling task is 16 or 32 bit. Individual system calls contain LRS details.

The AL uses memory from the pool of the calling task's job to create the LRS. The calling task should delete the segment after it is no longer needed. Creating multiple segments without deleting them can result in an E_MEM or E_SLOT condition code.

See also: AL calls **a_load**, **a_load_io_job**, and **rqe_a_load_io_job**

BIOS Layer-specific Information

The case-sensitivity of filenames and pathnames in the BIOS depends on the file driver. For example, iRMX file names are not case sensitive; file *xyz* is equal to file *XYZ*. However, files accessed through NFS may be case-sensitive.

File Types

Each BIOS system call may be used with one or more of these types of files, as specified in the call descriptions:

File Type	Description
Physical	Enables the OS to access an entire I/O device as single file. This is useful for accessing devices such as line printers, formatting secondary storage devices, and accessing backup volumes.
Stream	Enables two programs to communicate with each other: One program writes to the stream file while the other program reads from it.
Named	Divides data on storage devices into a hierarchical file structure specific to the iRMX OS. Named files include data files and directory files.
DOS	Provides access to standard DOS-formatted media from the iRMX III OS and iRMX For PCs.
EDOS	Encapsulated DOS (EDOS) makes DOS files accessible to iRMX for Windows applications using EIOS, BIOS, and UDI system calls.
Remote	Refers to iRMX named files accessed through the Remote File Driver of iRMX-NET or to files on any OS accessed through the NFS file driver.

System Call Types

There are two types of BIOS calls indicated by these prefixes:

Prefix	Meaning
rq_	Synchronous system calls. These calls begin running as soon as your application invokes them, continue running until they detect an error or finish their task, and then return control to your application.
rq_a	Asynchronous system calls. These calls run concurrently with your application, which can continue working while the BIOS deals with devices such as disk drives and tape drives.

Sequential and Concurrent Condition Codes

The asynchronous system calls return condition codes at two different times:

- Sequential codes return immediately after invocation of the system call
- Concurrent codes return as a result of asynchronous processing

See also: Sequential and concurrent parts of system calls, condition codes, *System Concepts*

I/O Request/Result Segments

Certain asynchronous BIOS calls return a data structure called an I/O Request/Result Segment (IORS).

See also: EIOS Layer-specific Information for the EIOS IORS structure

The synchronous portion of the I/O system creates an IORS when an application task requests an I/O operation. The IORS contains information about the request and about the unit on which the operation is to be performed. The asynchronous portion of the I/O system processes the request. After performing the requested operation, the I/O system modifies the IORS to indicate the results of what it has done. It then sends the IORS back to the mailbox specified by the `resp_mbox` parameter of the system call.

These system calls can return an IORS:

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_set_file_status
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 indicated by the `except_ptr` parameter of any call listed above. 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 IORS also contains the actual number of bytes read or written, if appropriate.

See also: Accessing the IORS, *Programming Techniques*

The fields of general interest in the IORS have this structure. The IORS also contains other fields which are of interest only to the designer of a device driver.

See also: IORS, *Driver Programming Concepts*

```
DECLARE IORS STRUCTURE(
   status                WORD_16,
   unit_status           WORD_16,
   actual                WORD_32);
```

or

```
typedef struct {
   UINT_16               status;
   UINT_16               unit_status;
   NATIVE_WORD           actual;
} IORS;
```

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. These are the codes, meanings, and associated mnemonics for the lower four bits:

Code	**Mnemonic**	**Meaning**
0	IO_UNCLASS	An error occurred but 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_ASSIGNED	An I/O error occurred during disk formatting; an alternate track was assigned

`actual` The actual number of bytes transferred.

EIOS Layer-specific Information

The case-sensitivity of filenames and pathnames in the EIOS depends on the file driver. For example, iRMX file names are not case sensitive; file *xyz* is equal to file *XYZ.* However, files accessed through NFS may be case-sensitive.

Colon characters are required in logical names such as :sd: when used in EIOS pathnames.

See also: Logical names, *System Concepts*,
Logical Names screen, *ICU User's Guide and Quick Reference*

Several EIOS system calls may be used with one or more types of files, as specified in the call descriptions.

See also: File types, BIOS Layer-specific Information for file type definitions

System Call Types

There are three types of EIOS calls, as indicated by these prefixes:

Prefix	**Meaning**
rq_s	Synchronous system calls that have asynchronous equivalents in the BIOS.
rq_	Synchronous system calls that do not have asynchronous equivalents in the BIOS.
rqe_	System calls that involve addressability of greater than 1 Mbyte.

I/O Request/Result Segments

Some EIOS calls return an abbreviated version of an IORS:

```
typedef struct {
   NATIVE_WORD             actual;
#ifdef __INT16__
   UINT_16                 actualfill;
#endif
   UINT_16                 device;
   UINT_8                  unit;
   UINT_8                  funct;
   UINT_16                 subfunct;
   UINT_32                 deviceloc;
   UINT_8 far *            buff;
   NATIVE_WORD             count;
#ifdef __INT16__
   UINT_16                 countfill;
#endif
   void far *              aux;
} IORS_DATA_STRUCT;
```

iRMK Kernel-specific Information

The iRMX OS includes the iRMK Kernel embedded within the iRMX Nucleus. This Kernel and its associated user interfaces give additional capabilities to the iRMX OS. Unless otherwise specified, when this manual refers to the Kernel, it means the iRMK Kernel.

See also: *System Concepts* for more information on Kernel capabilities

Syntax

In the call descriptions, the system calls, data structures, and data types are specified using the C language syntax. If you write your programs in C, you can access the system calls using this syntax.

The Kernel also provides support for PL/M and assembly language programs. The PL/M interface requires that you use a different set of include files in the compilation of your programs and possibly linking to a different interface library. The assembly language interface is a register interface; you must set up a group of registers with parameter values before calling the system calls.

See also: Developing Applications in Assembly Language, *Programming Techniques*

Data Types

The Kernel defines the UINT 64 type as a long integer type for use in some system calls. Write modules that use these system calls in PL/M or Assembly language. In iC-386, the default long is 32 bits. Keep 64-bit operations isolated in a separate module where the `long64` switch is enabled. This is necessary because `long64` changes the definition of long.

Scheduling Category

The descriptions of Kernel calls contain a scheduling category. This category indicates what effect a system call may have on task scheduling and whether a scheduling lock changes that effect. It also indicates whether the system call can be safely used by interrupt handlers, which should not lose control of the CPU when they run. There are four types:

Non-scheduling (Safe)
The system call does not cause rescheduling, and interrupt handlers can safely use it.

Signalling
The system call could put other tasks in the ready state. If those tasks are higher priority, rescheduling would occur, pre-empting the calling task. If this system call is called from an interrupt handler, the handler could lose control. A scheduling lock will prevent rescheduling when using such a system call. Any task state change caused by a signalling system call takes place immediately, but the running task is not switched until scheduling is started again.

Blocking
The system call could put the running task to sleep causing rescheduling. An interrupt handler should not call this system call unless it knows that the running task will not be put to sleep as a result; the system call will complete its operation without blocking the calling task. A scheduling lock does not prevent a blocking system call from causing rescheduling.

Rescheduling (Unsafe)
This system call always causes rescheduling. An interrupt handler should never call this system call. A scheduling lock does not prevent rescheduling for this system call.

See also: Chapter 6 for Nucleus calls that can be made from interrupt handlers

Parameters

The Kernel header files declare literal values to define many of the data structures and parameter values needed in programming the system calls. To use the Kernel-defined values when setting up data structures and calling the system calls, include the appropriate header files in your programs.

➾ **Note**

Some system calls include parameters that are actually status return values. Those system calls will include a Return Value subsection.

Flags Parameters

Masks typically refer to a single bit field in the flag. A mask is used to isolate a value in the flags field when you examine a flag. To set a flag, choose one literal value for each mask listed. Then OR the values together to form the flags value.

For example, these are the masks for the `flags` parameter of the **KN_create_semaphore** system call.

KN_EXCH_TYPE_MASK

Specifies the type of semaphore. Choose one of these literals:

Literal	**Meaning**
KN_FIFO_QUEUEING	The semaphore uses FIFO queueing
KN_PRIORITY_QUEUEING	The semaphore uses priority queueing
KN_REGION	The exchange is a single-unit region

KN_INITIAL_SEM_STATE_MASK

Specifies the number of initial units the semaphore receives. Choose one of these literals:

Literal	**Meaning**
KN_ZERO_UNITS	The semaphore is created with no units
KN_ONE_UNIT	The semaphore is created with one unit

To set up a semaphore that uses FIFO queueing and has one unit, specify these literal values for the `flags` parameter:

```
KN_FIFO_QUEUEING | KN_ONE_UNIT
```

Kn_Task_State Structure

KN_TASK_STATE is a structure describing the state of a task. It is used in the Kernel handler procedures **create_task_handler**, **delete_task_handler**, and **task_switch_handler**. Only some parts of this structure are visible. None of it should be modified.

```
typedef struct {
   UINT_8                 reserved1 [112];
   UINT_16                dynamic_priority;
   UINT_16                static_priority;
   UINT_8                 reserved2 [116];
   UINT_16                rmx_task_token;
} KN_TASK_STATE;
```

Where:

`reserved1` Private to the Kernel.

`dynamic_priority`
The current dynamic priority of the task. This field is equal to the static priority field unless the task's priority has been adjusted because of region ownership, in which case it is equal to the adjusted priority. The dynamic priority of tasks is used in scheduling the processor.

`static_priority`
The current static priority of the task. This field gives the priority of the task if priority adjustment due to regions is ignored.

`reserved2` Private to the Kernel.

`rmx_task_token`
This is the iRMX token corresponding to this task.

Configuring the Kernel Tick Interval

You can specify the Kernel Tick Ratio (KTR) in the *rmx.ini* file or by using the ICU, but do not assume that a Nucleus tick is equivalent to a Kernel tick, especially for KTR values that are less than the 10 millisecond default. You should write code that adapts to the KTR values.

See also: Getting System Information, in this chapter
KTR, *System Configuration and Administration,*
KTR, *ICU User's Guide and Quick Reference*

Getting System Information

The OS catalogs several items of information about the system, including the Kernel Tick Ratio (KTR), in an abject called RQSYSINFO. To get the information, first invoke the Nucleus **rq_get_task_tokens** system call to get the token for the root job (where the RQSYSINFO object is cataloged). Then call **rq_lookup_object**, specifying the token for the root job and the string RQSYSINFO.

The token returned by **rq_lookup_object** is a SELECTOR for a memory segment where the information is stored. Use a structure such as the following to get the information at that segment, in PL/M:

```
DECLARE sysinfo_type STRUCTURE(
   boot_dev(15)        BYTE,
   file_driver         BYTE,
   boot_file(30)       BYTE,
   reserved1(11)       BYTE,
   nuc_tick_interval   WORD_16,
   kn_tick_ratio       WORD_16,
   reserved2(29)       BYTE,
   bustype             BYTE,
   reserved3(6)        BYTE,
   physical_memory     WORD_32,
   reserved4(27)       BYTE,
   user_reserved(32)   BYTE);
```

or in C:

```
struct sysinfo_type {
   UINT_8       boot_dev[15];
   UINT_8       file_driver;
   UINT_8       boot_file[30];
   UINT_8       reserved1[11];
   UINT_16      nuc_tick_interval;
   UINT_16      kn_tick_ratio;
   UINT_8       reserved2[29];
   UINT_8       bustype;
   UINT_8       reserved3[6];
   UINT_32      physical_memory;
   UINT_8       reserved4[27];
   UINT_8       user_reserved[32];
};
```

Where:

`boot_dev[15]`
An RMX_STRING containing the name of the boot device.

`file_driver`
The file driver type used by the boot device:

Value	File Driver
3	DOS
4	Named
5	Remote
6	EDOS

`boot_file[30]`
An RMX_STRING containing the name of the boot file.

`nuc_tick_interval`
The number of milliseconds from one Nucleus clock tick to the next.

`kn_tick_ratio`
The KTR value. Divide `nuc_tick_interval` by this value to get the number of milliseconds in the Kernel tick interval.

`bustype` 1 is Multibus I, 2 is Multibus II, 3 is PC.

`physical_memory`
The top of physical memory as known by the iRMX Free Space Manager.

`user_reserved[32]`
Available for your application's use.

System Call Summary Tables

The following tables summarize the iRMX system calls by OS layer and by functional group within each layer. The calls are listed alphabetically within each functional group, without regard to their various prefixes (rq_, rqe_, etc.).

Application Loader System Calls Summary

This table summarizes the AL system calls by functional groups.

Table 1-6. Application Loader System Calls

FILE AND MODULE LOADING	
Call Name	**Description**
a_load	Loads an object file from secondary storage into memory.
s_overlay	Loads an overlay module into memory.
JOB AND TASK CREATION WITH FILE LOADING	
a_load_io_job	obsolete; it is provided for compatibility with older versions of the iRMX OS.
rqe_a_load_io_job	Creates an I/O job with a memory pool of up to 4 Gbytes, loads a specified object file, and creates a task to execute the loaded code.
s_load_io_job	obsolete; it is provided for compatibility with older versions of the iRMX OS.
rqe_s_load_io_job	Creates an I/O job with a memory pool of up to 4 Gbytes, loads a specified object file, and creates a task to execute the loaded code.

BIOS System Calls Summary

This table summarizes the BIOS calls by functional groups.

Table 1-7. BIOS System Calls

JOB-LEVEL SYSTEM CALLS	
Call Name	**Description**
encrypt	Encrypts a specified string of characters.
get_default_prefix	Returns the default prefix of a specified job.
get_default_user	Returns the default user object of a specified job.
set_default_prefix	Sets the default prefix for a specified existing job.
set_default_user	Sets the default user object for a specified existing job.
DEVICE-LEVEL SYSTEM CALLS	
a_physical_attach_device	Attaches the specified device to the BIOS.
a_physical_detach_device	Detaches a device that was attached using **a_physical_attach_device**.
rq_install_duibs	Installs a cluster of Device Unit Information Blocks (DUIBs) into the BIOS.
a_special	Enables tasks to perform a variety of device-level functions.
FILE/CONNECTION-LEVEL SYSTEM CALLS	
a_attach_file	Creates a connection to an existing file of any type.
a_create_directory	Creates a directory file.
a_create_file	Creates a file and returns a token for the new file connection.
a_delete_connection	Deletes a file connection created by **a_create_file**, **a_create_directory**, or **a_attach_file**.
a_delete_file	Marks a stream, named data or named directory file for deletion.
install_file_driver	Installs a loadable file driver into the BIOS.

continued

Table 1-7. BIOS System Calls (continued)

FILE-MODIFICATION SYSTEM CALLS	
Call Name	**Description**
a_change_access	Changes the access rights to a named data or directory file.
a_rename_file	Changes the pathname of a named data or directory file.
a_set_file_status	Changes the owner and/or time stamps of a file.
a_truncate	Truncates a named data file at the current setting of the file pointer.
FILE INPUT/OUTPUT SYSTEM CALLS	
a_close	Closes an open file connection for any type of file.
a_open	Opens an asynchronous file connection for I/O operations for any type of file.
a_read	Reads the requested number of bytes on an open connection for any type of file.
a_seek	Moves the file pointer of an open file connection.
a_update	Updates a device by writing all buffered partial sectors.
wait_io	Returns the concurrent condition code for the prior call to the calling task.
wait_iors	Waits for an IORS and copies it to a user-provided buffer.
a_write	Writes data from the calling task's buffer to a connected physical, stream, or named data file.
GET STATUS/ATTRIBUTE SYSTEM CALLS	
a_get_connection_status	Returns information about the connection status of a specified file.
a_get_directory_entry	Returns the filename associated with an entry number in a named, DOS, or EDOS directory.
get_file_driver_status	Returns information on a specified file driver.
a_get_file_status	Returns status and attribute information about a specified file.
a_get_path_component	Returns the name of a data or directory file, as cataloged in its parent directory.

continued

Table 1-7. BIOS System Calls (continued)

USER OBJECT SYSTEM CALLS	
create_user	Creates a user object, accepts a list of IDs, and returns a token for the new object.
delete_user	Deletes a user object.
inspect_user	Accepts a token for a user object and returns a list of the IDs contained in the user object.
EXTENSION DATA SYSTEM CALLS	
a_get_extension_data	Writes the extension data for a named data or directory file; not valid for DOS files.
a_set_extension_data	Stores a named file's extension data; not valid for DOS files.
TIME/DATE SYSTEM CALLS	
get_global_time	Reads the time of day from the battery-backed-up hardware clock.
set_global_time	Sets the battery-backed-up hardware clock to a specified time.

EIOS System Calls Summary

This table summarizes the EIOS calls by functional groups.

Table 1-8. EIOS System Calls

I/O JOBS	
Call Name	**Description**
create_io_job	Obsolete; it is provided for compatibility with earlier versions of the OS.
rqe_create_io_job	Creates an I/O job containing one task with a memory pool of up to 4 Gbytes.
exit_io_job	Sends a message to a previously designated mailbox and deletes the calling task.
start_io_job	Starts the initial task in an I/O job.
LOGICAL NAMES	
s_catalog_connection	Creates a logical name for a connection by cataloging the connection in the object directory of a job.
s_get_directory_entry	Returns a directory entry filename to the caller.
s_get_path_component	Returns the name of a named file as the file is known in its parent directory.
hybrid_detach_device	Temporarily removes the correspondence between a logical name and a physical device.
logical_attach_device	Assigns a logical name to a physical device.
logical_detach_device	Removes the correspondence between a logical name and a physical device, and removes the logical name from the root object directory.
s_lookup_connection	Returns a token for the connection associated with the specified logical name.
s_uncatalog_connection	Deletes a logical name from the object directory of a job.
FILES AND CONNECTIONS	
s_attach_file	Creates a connection to an existing file.
s_create_directory	Creates a new directory file and automatically adds a new entry to the parent directory.
s_create_file	Creates a new physical, stream, or named data file.
s_change_access	Changes the access list for a named file.
s_rename_file	Changes the pathname of a directory or data file.

continued

Table 1-8. EIOS System Calls (continued)

FILES AND CONNECTIONS (continued)	
Call Name	**Description**
s_close	Closes an open connection to a named, physical, or stream file.
s_open	Opens a file connection.
s_read_move	Reads a number of contiguous bytes from a file associated with a connection to a buffer specified by the calling task.
s_seek	Moves the file pointer for any open physical or named file connection.
s_truncate_file	Removes information from the end of a named data file.
s_write_move	Writes a collection of bytes from a buffer to a file.
s_delete_connection	Deletes a file connection, not a device connection.
s_delete_file	Deletes a stream, named data, or named directory file created by the BIOS or the EIOS.
DEVICES	
s_special	Enables tasks to communicate with devices, device drivers, and the stream file driver to perform various operations.
OBTAINING OR CHANGING STATUS	
s_get_connection_status	Provides status information about file and device connections that were created by the BIOS or the EIOS.
s_get_file_status	Obtains information about a physical, stream, or named file created by the BIOS or the EIOS.
get_logical_device_status	Provides status information about logical names that represent devices.
s_set_file_status	Changes the owner and/or time stamps of a file.
USERS	
get_user_ids	Returns the user ID(s) associated with a user defined in the User Definition File (UDF).
verify_user	Verifies a user's name and password.

Human Interface System Calls Summary

This table summarizes the HI calls by functional groups.

Table 1-9. Human Interface System Calls

Call Name	Description
c_get_input_connection	Returns an EIOS connection object for the specified input file.
c_get_output_connection	Returns an EIOS connection object for the specified output file.
	COMMAND PARSING
c_backup_char	Moves the parsing buffer pointer back one character for each occurrence of the call.
c_get_char	Gets a character from the parsing buffer and moves the parsing buffer pointer to the next character.
c_get_input_pathname	Gets a pathname from the list of input pathnames in the parsing buffer.
c_get_output_pathname	Gets a pathname from the list of output pathnames in the parsing buffer.
c_get_parameter	Retrieves one parameter from the parsing buffer and moves the parsing pointer to the next parameter.
c_set_parse_buffer	Permits parsing the contents of a buffer other than the command line buffer whenever the parsing system calls are used.
c_get_command_name	Obtains the pathname of the command entered by the operator.
	MESSAGE PROCESSING
c_format_exception	Creates a default message for a given exception code and writes that message into a user-provided string.
c_send_co_response	Sends a message to *:co:* and reads a response from *:ci:*.
c_send_eo_response	Sends a message to and reads a response from the operator's terminal.

continued

Table 1-9. Human Interface System Calls (continued)

COMMAND PROCESSING	
Call Name	**Description**
c_create_command_connection	Returns a token for a command connection object required to invoke commands programmatically instead of interactively.
c_delete_command_connection	Deletes a command connection object previously defined in a **c_create_command_connection** call and frees the memory used by the command connection's data structures.
c_send_command	Stores a command line in the command connection created by the **c_create_command_connection** call, concatenates the command line with any others already stored there, and (if the command invocation is complete) invokes the command.
PROGRAM CONTROL	
c_set_control_c	Changes the default response to a <Ctrl-C> entry to a response that meets the needs of your task.

Nucleus System Calls Summary

This table summarizes the Nucleus system calls by functional group.

Table 1-10. Nucleus System Calls

JOBS	
Call Name	**Description**
create_job	Obsolete; provided for compatibility.
rqe_create_job	Creates a job containing one task with a memory pool of up to 4 Gbytes and returns a token for the job.
delete_job	Deletes a specific job.
offspring	Returns a token for the segment containing tokens of the child jobs of the specified job.
rqe_offspring	Fills the specified data structure with tokens of the child jobs of the specified job.
rqe_set_max_priority	Dynamically changes the maximum priority of tasks in a job.
TASKS	
create_task	Creates a task and returns a token for it.
delete_task	Deletes a specific non-interrupt task.
get_priority	Returns the static priority of a specific task.
get_task_accounting	Returns task creation time and amount of execution time.
get_task_info	Returns high-level information about a task, including priority and execution state.
get_task_state	Returns low-level information about a task, including state of the CPU registers for ready tasks.
get_task_tokens	Returns a token for either itself, its job, its job's parameter object, or the root job.
resume_task	Decreases a task's suspension depth by one.
set_priority	Changes the priority of a non-interrupt task.
sleep	Places the calling task in the asleep state for a specified amount of time.
suspend_task	Increases a task's suspension depth by one.
system_accounting	Enables or disables tracking of CPU use for task accounting

continued

Table 1-10. Nucleus System Calls (continued)

INTERRUPT LEVELS, INTERRUPT HANDLERS, and INTERRUPT TASKS	
Call Name	**Description**
disable	Disables a specific interrupt level.
enable	Enables a specific interrupt level.
end_init_task	Informs the root task that a synchronous initialization process has completed. Will not affect loaded jobs.
enter_interrupt	Sets up a previously-specified data segment base address for the calling interrupt handler.
exit_interrupt	Used by interrupt handlers to send an end-of-interrupt (EOI) to hardware.
rqe_exit_interrupt	A high performance version of the **exit_interrupt** call.
get_level	Returns the interrupt level of the highest priority interrupt that an interrupt handler is currently processing.
reset_interrupt	Cancels the assignment of an interrupt handler to a level.
set_interrupt	Assigns an interrupt handler and, if desired, an interrupt task to an interrupt level.
signal_interrupt	Used by interrupt handlers to invoke interrupt tasks.
rqe_timed_interrupt	Puts the calling interrupt task to sleep until either it is called into service by an interrupt handler or a specified time period elapses.
wait_interrupt	Puts the calling interrupt task to sleep until it is called into service by an interrupt handler.
MAILBOXES	
add_reconfig_mailbox	Specifies a mailbox that will receive failure and reset messages generated by the Multibus II watchdog timer.
create_mailbox	Creates a mailbox and returns a token for it.
delete_mailbox	Deletes a specific mailbox.
receive_data	Receives a data message from a data mailbox.
receive_message	Receives a signal message from an object mailbox.
send_data	Sends a data message of up to 80H characters to a data mailbox.
send_message	Sends a signal object to an object mailbox.

continued

Table 1-10. Nucleus System Calls (continued)

SEMAPHORES	
Call Name	**Description**
create_semaphore	Creates a semaphore and returns a token for it.
delete_semaphore	Deletes a specific semaphore.
receive_units	Requests a specific number of units from a semaphore.
send_units	Sends a specific number of units to a semaphore.
REGIONS	
accept_control	Provides access to data protected by a region only if access is immediately available.
create_region	Creates a region and returns a token for it.
delete_region	Deletes a specific region.
receive_control	Enables the calling task to gain access to data protected by a region.
send_control	Relinquishes control to the next task waiting at the region.
SEGMENTS and MEMORY POOLS	
create_segment	Creates a segment and returns a token for it.
delete_segment	Returns a segment to the memory pool from which it was allocated or deletes a descriptor from the Global Descriptor Table (GDT).
get_buffer_limit	Returns the maximum size of a buffer starting from a pointer within a regular or virtual iRMX segment.
get_pool_attrib	Returns the memory pool attributes of the calling task's job.
get_size	Returns the size, in bytes, of a regular or virtual iRMX segment.
move_data	Copies bytes from one buffer to another.
rqe_get_pool_attrib	Returns the same information as **get_pool_attributes** for any job, plus the amount of memory borrowed and the token of the parent job.
set_pool_min	Sets the minimum attribute of the memory pool of the caller's job.
validate_buffer	Verifies that a buffer pointer is a valid pointer to physical memory and that it has access rights to the memory.

continued

Table 1-10. Nucleus System Calls (continued)

Call Name	Description
	DELETION CONTROL
disable_deletion	Makes an object immune to ordinary deletion.
enable_deletion	Makes an object susceptible to ordinary deletion.
force_delete	Deletes objects whose disabling depths are 0 or 1.
	BUFFER POOLS
create_buffer_pool	Creates a buffer pool object that can be associated with one or more ports.
delete_buffer_pool	Deletes a buffer pool object.
release_buffer	Returns previously allocated buffer space to the specified buffer pool.
request_buffer	Gets a buffer from an existing buffer pool.
	OBJECTS
catalog_object	Places an entry for an object in an object directory.
rqe_change_object_access	Changes the access rights of iRMX segments or composite objects.
rqe_get_address	Returns the physical address of an object.
rqe_get_object_access	Returns the access type of an object whose token is specified.
get_type	Returns the type code for the specified object.
lookup_object	Returns a token for the specified cataloged object name.
uncatalog_object	Removes an entry for an object from an object directory.
	DESCRIPTORS
rqe_change_descriptor	Changes the base physical address and size of a descriptor in the GDT.
rqe_create_descriptor	Builds a descriptor for a memory segment, places the descriptor in the GDT, and returns a token for that descriptor.
rqe_delete_descriptor	Removes a descriptor entry from the GDT.

continued

Table 1-10. Nucleus System Calls (continued)

EXCEPTION HANDLERS	
Call Name	**Description**
get_exception_handler	Returns the address and exception mode of the calling task's exception handler.
rqe_get_exception_handler	Returns the address and exception mode of the exception handler for the current task or job, for the system default, or for the system hardware trap handler.
set_exception_handler	Assigns an exception handler and exception mode attributes for the calling task.
rqe set_exception_handler	Assigns an exception handler and exception mode attributes for the calling task, its job, or the system default; or sets values for hardware trap handlers.
COMPOSITE OBJECTS	
alter_composite	Replaces components of composite objects.
create_composite	Creates a composite object and returns a token for it.
delete_composite	Deletes a composite object but not its component objects.
inspect_composite	Returns a list of the component tokens contained in a composite object.
EXTENSION OBJECTS	
create_extension	Creates a new object type and returns a token for it.
delete_extension	Deletes an extension object and all composites of that type.
OS EXTENSIONS	
rqe_set_os_extension	Attaches or deletes the entry-point address of a user-written OS extension to a call gate.
signal_exception	Used by OS extensions to signal the occurrence of an exceptional condition.
MULTIBUS II INTERCONNECT CALLS	
get_interconnect	Retrieves the contents of the specified interconnect register.
set_interconnect	Alters the contents of an interconnect register to a specified value.

continued

Table 1-10. Nucleus System Calls (continued)

COMMUNICATION SERVICE CALLS	
Call Name	**Description**
attach_buffer_pool	Associates a buffer pool with one or more ports.
attach_port	Forwards all messages sent to the port that issued the call to a sink port.
broadcast	Sends a control message to every message passing host.
cancel	Performs synchronous cancellation of RSVP message transmission.
connect	Creates a connection between the sending task and a remote task.
create_port	Creates a port object that can be used in message passing.
delete_port	Deletes a specific port.
detach_buffer_pool	Ends the association between a buffer pool and a port.
detach_port	Ends message forwarding from the source port to the sink port.
get_host_id	Returns the host ID of the board that the task is running on.
get_port_attributes	Returns information about the specified port.
receive	Accepts a message at a port.
receive_fragment	Accepts a fragment of an RSVP data message.
receive_reply	Accepts a message that is a reply to an earlier request.
receive_signal	Receives a signal from a remote host at a specified port.
send	Sends a data message from a port to a port on another board.
send_reply	Sent in response to the **rq_send_rsvp** system call.
send_rsvp	Initiates a request/response message exchange.
send_signal	Sends a signal message to a remote host through the specified port.
TIME/DATE CALLS	
get_time	Returns the date and time from the local Nucleus clock.
set_time	Sets the local Nucleus clock to a specified time.

UDI System Calls Summary

This table summarizes the UDI system calls by functional group.

Table 1-11. UDI System Calls

PROGRAM CONTROL CALLS	
Call Name	**Description**
dq_exit	Exits from the current application job.
dq_overlay	Loads an overlay module.
dq_trap_cc	Designates an interrupt procedure that takes control when <Ctrl-C> is entered.
FILE-HANDLING CALLS	
dq_attach	Creates a connection to a file.
dq_change_access	Changes access rights to a file or directory.
dq_change_extension	Changes the extension of a file name in memory.
dq_close	Closes the specified file connection.
dq_create	Creates a file.
dq_delete	Deletes a file.
dq_detach	Closes a file and deletes its connection.
dq_file_info	Returns data about directory and data files.
dq_get_connection_status	Returns information about a file connection.
dq_open	Opens a file for a particular type of access.
dq_read	Reads bytes from a file.
dq_rename	Renames a file.
dq_seek	Moves the file pointer of a file.
dq_special	Sets the mode of a console input device.
dq_truncate	Truncates a file at the position specified by the file pointer.
dq_write	Writes data to a file.

continued

Table 1-11. UDI System Calls (continued)

MEMORY MANAGEMENT CALLS	
Call Name	**Description**
dq_allocate	Requests a memory segment.
dq_free	Returns a memory segment to the system.
dq_get_msize	Returns the size of a segment allocated by **dq_mallocate**.
dq_get_size	Returns the size of a specified segment.
dq_mallocate	Requests a logically contiguous memory segment of a specified size.
dq_mfree	Returns memory allocated by **dq_mallocate** to the Free Space Pool.
dq_reserve_io_memory	Sets aside memory for I/O operations.
EXCEPTION-HANDLING CALLS	
dq_decode_exception	Converts a condition code into its equivalent mnemonic.
dq_get_exception_handler	Returns the address of the current exception handler.
dq_trap_exception	Substitutes an alternate exception handler.
UTILITY AND COMMAND PARSING	
dq_decode_time	Decodes the specified binary date/time value to ASCII characters.
dq_get_argument	Returns an argument from the command line.
dq_get_system_id	Returns the identity of the OS environment.
dq_get_time	Obsolete: included for compatibility.
dq_switch_buffer	Selects a new command line buffer.

Windows- and DOS-Specific System Calls Summary

This table summarizes the Windows- and DOS-specific system calls. The first two calls are used by DOS applications only; they are not supported in the iRMX OS.

Table 1-12. Windows- and DOS-Specific System Calls

DATA TRANSFER	
Call Name	**Description**
rqe_read_segment	Enables a DOS application program to transfer data from a Protected Virtual Address Mode (PVAM) segment to a Real Mode segment.
rqe_write_segment	Enables a DOS application program to transfer data from a Real Mode segment to a PVAM segment.
EXTENSIONS AND INTERRUPTS	
rqe_set_vm86_extension	Installs and removes a Virtual 8086 Mode (VM86) extension at the specified interrupt level.
rqe_dos_request	Makes DOS/ROM BIOS requests and other software interrupts handled by DOS applications.
iRMX STATUS	
RqeGetRmxStatus	Gets the current status of the iRMX environment.

Kernel System Calls Summary

This table summarizes the Kernel system calls.

⚠ **CAUTION**
iRMK functions do not validate objects or object areas. Be careful to pass correct values.

Table 1-13. Kernel System Calls and Handlers

COMMUNICATION AND SYNCHRONIZATION	
Call Name	**Description**
KN_create_mailbox	Creates a mailbox.
KN_create_semaphore	Creates a semaphore.
KN_delete_mailbox	Deletes a mailbox.
KN_delete_semaphore	Deletes a semaphore.
KN_receive_data	Requests a message from a mailbox.
KN_receive_unit	Requests a unit from a semaphore.
KN_send_data	Sends data to a mailbox.
KN_send_priority_data	Places a priority message at head of mailbox queue.
KN_send_unit	Adds a unit to a semaphore.
MEMORY MANAGEMENT	
KN_create_area	Allocates memory from a pool.
KN_create_pool	Creates a memory pool.
KN_delete_area	Returns a memory area to the memory pool.
KN_delete_pool	Deletes a memory pool.
KN_get_pool_attributes	Gets a memory pool's attributes.
TASK MANAGEMENT	
KN_reset_handler *	Removes previously set task handler.
KN_set_handler *	Dynamically sets task handler.
KN_start_scheduling	Cancels one scheduling lock.
KN_stop_scheduling	Temporarily locks the scheduling mechanism.

* You cannot make these calls in a flat model application

continued

Table 1-13. Kernel System Calls and Handlers (continued)

TIME MANAGEMENT	
Call Name	**Description**
KN_create_alarm *	Creates and starts a virtual alarm clock.
KN_delete_alarm *	Deletes an alarm.
KN_get_time	Gets the current value of the Kernel clock timer.
KNE_get_time	Gets the current value of the Kernel clock timer from a structure that allows the use of 32-bit data types.
KN_reset_alarm *	Resets an existing alarm.
KN_set_time	Sets the Kernel clock timer.
KNE_set_time	Sets the Kernel clock timer in a structure that allows the use of 32-bit data types.
KN_sleep	Puts the calling task to sleep.
HANDLERS	
create_task_handler *	Creates a task.
delete_task_handler *	Deletes a task.
task_switch_handler *	Executes when a task switch occurs.

* You cannot make these calls or write these handlers in a flat model application

Virtual Memory System Calls Summary

This table summarizes the virtual memory system calls.

Table 1-14. Virtual Memory System Calls

Call Name	Description
rqv_allocate	Allocates physical memory to a virtual segment.
rqv_allocate_at	Allocates physical memory to a virtual segment at a specific offset.
rqv_change_access	Changes the access rights for physical memory within a virtual segment.
rqv_create_segment	Creates a virtual segment with no physical memory allocated to it.
rqv_free	Frees physical memory associated with a virtual segment.
rqv_map_physical	Maps physical memory into the address space within a virtual segment.

Networking System Calls Summary

This table summarizes the system calls you can use to communicate with iNA 960 and the Name Server. The network system calls begin with a **cq_** prefix rather than **rq_**. This manual does not describe these calls.

See also: *Network User's Guide and Reference* for the full description of these calls

Table 1-15. System Calls that Access iNA 960 Network Software

ADDRESS/POINTER CONVERSION	
Call Name	**Description**
cq_comm_ptr_to_dword	Converts a pointer to the corresponding 32-bit absolute address.
PROCESSING	
cq_comm_rb	Delivers a request block to iNA or to the Name Server for processing.
STATUS	
cq_comm_multi_status	Returns NIC and iNA 960 status information from a specified NIC.
cq_comm_status	Returns NIC and iNA 960 status information.
USERS	
cq_create_comm_user	Creates a user ID for programmatic access to iNA 960.
cq_create_multi_comm_user	Creates a unique user ID for programmatic access to a specified NIC and iNA 960 job.
cq_delete_comm_user	Releases all resources and returns all request blocks held on behalf of a specified user ID.

□ □ □

Application Loader System Calls 2

a_load

Asynchronously loads an object file from secondary storage into memory.

Syntax, PL/M and C

```
CALL rq$a$load (connection, response_mbox, except_ptr);
```

```
rq_a_load (connection, response_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection`	SELECTOR	SELECTOR
`response_mbox`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection`

A token for a connection to the file to be loaded. The user object specified when the connection was created must have had read access. The connection must have been created in the calling task's job, be to a named file, and be closed.

`response_mbox`

A token for the mailbox to which the AL sends the Loader Result Segment (LRS) after the concurrent part of the system call runs.

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

The object code to be loaded must be of the Single Task Loadable (STL) type with LODFIX records.

A_load cannot automatically cause the code to be executed as a task; the calling task must explicitly do this using the Nucleus call **create_task**.

Once the loaded program has finished, delete all the segments allocated for this program to free the memory for use by other tasks or jobs. To find tokens for the segments to delete, check the token array in `more_slots`.

Loader Result Segment

This call returns this LRS. Use the contents of the LRS to create a task or job to start the loaded code. The LRS provides the values to specify for the initial address, stack pointer, stack size, and data segment.

```
DECLARE a_load_lrs STRUCTURE(
   except_code           WORD_16,
   reserved_word1        WORD_16,
   reserved_byte         BYTE,
   reserved_word2        WORD_16,
   code_seg_offset       WORD_32,
   code_seg_base         SELECTOR,
   stack_offset          WORD_32,
   stack_seg_base        SELECTOR,
   stack_size            WORD_32,
   data_seg_base         SELECTOR,
   num_more_slots        BYTE,
   more_slots(*)         SELECTOR);
```

or

```
typedef struct {
   UINT_16                 except_code;
   UINT_16                 reserved_1;
   UINT_8                  reserved_2;
   UINT_16                 reserved_3;
   NATIVE_WORD             code_seg_offset;
   SELECTOR                code_seg_base;
   NATIVE_WORD             stack_offset;
   SELECTOR                stack_seg_base;
   NATIVE_WORD             stack_size;
   SELECTOR                data_seg_base;
   UINT_8                  num_more_slots;
   SELECTOR                more_slots[255];
                           /* adjust 255 as necessary */
} A_LOAD_LRS_STRUCT
```

Where:

`except_code`
: The condition code for the concurrent part of the system call.

`code_seg_offset`
: The initial value for the loaded program's instruction pointer (IP) register taken from the Task State Segment (TSS) of the object file.

`code_seg_base`
: A token for the initial value of the code segment selector.

`stack_offset`
: The initial value of the stack pointer, taken from the TSS of the object file.

`stack_seg_base`
: A token for the initial value of the stack segment selector.

`stack_size`
: Specifies the number of bytes required for the loaded program's stack. The AL sets this value to 0 whenever `stack_offset` is 0 and `stack_seg_base` is a null selector.

`data_seg_base`
: A token for the initial value of the data segment selector taken from the TSS of the object file. The AL sets this value to a null selector if the target file contains no initial data segment selector.

`num_more_slots`

Indicates how many Global Descriptor Table (GDT) or Local Descriptor Table (LDT) slots were allocated (from 0 to 255), including the initial code, data, and stack segments. If greater than 255, the value returned is set to 255.

`more_slots`

A token array that lists the selectors of all the segments that were allocated for the loaded program. The length of this array is contained in `num_more_slots`.

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_BAD_HEADER	0062H	The object file contains an invalid header record.
E_CONN_NOT_OPEN	0034H	The AL opened the connection but some other task closed the connection before the loading operation began.
E_CONN_OPEN	0035H	The calling task specified a connection that was already open.
E_DEV_DETACHING	0039H	The device containing the specified file is being detached. If the device is remote, a retry may succeed.
E_EOF	0065H	The AL encountered an unexpected EOF while reading a record.
E_EXIST	0006H	Either the connection or msg_mbox parameter did not refer to an existing object.
E_FACCESS	0026H	The specified connection did not have read access to the file.
E_FLUSHING	002CH	The device containing the target file is being detached.
E_IO_HARD	0052H	A hard I/O error occurred. A retry is probably useless because secondary storage is not functioning.
E_IO_OPRINT	0053H	The device containing the target file was off-line. Operator intervention is required.

E_IO_SOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation and failed; a retry may succeed.
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its object limit. • Either the calling task's job, or the job's default user object, is already involved in 255 I/O operations.
E_LOADER_SUPPORT	006FH	Loading the target file requires capabilities not configured into the AL.
E_MEM	0002H	The memory available to the calling task's job or the BIOS is not sufficient to complete the call.
E_NOT_FILE_CONN	0032H	The calling task specified a connection to a device rather than to a named file.
E_SHARE	0028H	The calling task tried to open a connection to a file already being used by some other task, and the file's sharing attribute is not compatible with the open request.
E_SUPPORT	0023H	The specified connection was not created by the calling task's job.
E_TYPE	8002H	The connection parameter is not a token for a connection.

Concurrent Condition Codes: returned to except_code in the LRS after loading attempt

E_OK	0000H	No exceptional conditions occurred.
E_DEV_DETACHING	0039H	The device containing the specified file is being detached. If the device is a remote device, a retry may succeed.
E_EOF	0065H	The call encountered an unexpected EOF.

E_EXIST	0006H	At least one of these is true: • The specified mailbox was deleted before the loading operation completed. • The device containing the file to be loaded was detached before the loading operation completed.
E_FLUSHING	002CH	The device containing the target file is being detached.
E_IO_HARD	0052H	A hard I/O error occurred. A retry is probably useless.
E_IO_OPRINT	0053H	The device containing the target file was off-line. Operator intervention is required.
E_IO_SOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation and failed; a retry may succeed.
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_LIMIT	0004H	The calling task's job has already reached its object limit.
E_NO_LOADER_MEM	0067H	The memory pool of the newly created I/O job does not currently have a block of memory large enough to allow the AL to run.
E_PARAM	8004H	The target file has a stack smaller than 16 bytes.

a_load_io_job

Obsolete. Asynchronously creates an I/O job with a memory pool of up to 1 Mbyte, loads a specified object file, and creates a task to execute the loaded code. Only tasks running within I/O jobs should invoke this call. It is provided for compatibility with earlier versions of the OS.

See also: **rqe_a_load_io_job**

Syntax, PL/M and C

```
job = rq$a$load$io$job (connection, pool_min, pool_max,
      except_handler, job_flags, task_priority, task_flags,
      msg_mbox, except_ptr);
```

```
job = rq_a_load_io_job (connection, pool_min, pool_max,
      except_handler, job_flags, task_priority, task_flags,
      msg_mbox, except_ptr);
```

rqe_a_load_io_job

Asynchronously creates an I/O job with a memory pool of up to 4 Gbytes, loads a specified object file, and creates a task to execute the loaded code. For segmented applications, only tasks running within I/O jobs should invoke this call. However, you must use this call to load standalone, linked flat model applications instead of creating an I/O job in flat model.

Syntax, PL/M and C

```
job = rqe$a$load$io$job (connection, pool_min, pool_max,
      except_handler, job_flags, task_priority, task_flags,
      msg_mbox, except_ptr);
```

```
job = rqe_a_load_io_job (connection, pool_min, pool_max,
      except_handler, job_flags, task_priority, task_flags,
      msg_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
job	SELECTOR	SELECTOR
connection	SELECTOR	SELECTOR
pool_min	WORD_32	UINT_32
pool_max	WORD_32	UINT_32
except_handler	POINTER	EXCEPTION_STRUCT far *
job_flags	WORD_16	UINT_16
task_priority	BYTE	UINT_8
task_flags	WORD_16	UINT_16
msg_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

job A token for the newly created I/O job, only valid if E_OK returns.

Parameters

connection

A token for a connection to the file to be loaded. The user object specified when the connection was created must have had read access. The connection must have been created in the calling task's job, be to a named file, and be closed.

pool_min

Specifies the minimum size of the new job's memory pool in 16-byte paragraphs. The upper limit is 4 Gbytes.

`pool_max`

Specifies the maximum allowable size of the new job's memory pool in 16-byte paragraphs. The upper limit is 4 Gbytes.

`except_handler`

A pointer to this structure:

```
DECLARE exception STRUCTURE (
   exception_handler_ptr              POINTER,
   exception_mode                     BYTE);
```

or

```
typedef struct exception_struct {
   void far *             exception_handler_ptr;
   UINT_8                 exception_mode;
} EXCEPTION_STRUCT;
```

Where:

`exception_handler_ptr`

If not null, references the first instruction of the new job's own exception handler. If null, the new job's exception handler is the system default exception handler. The exception handler for the new task becomes the default exception handler for the job.

`exception_mode`

Indicates when control is to be passed to the exception handler. It is encoded as:

Value	**When Control Passes To Exception Handler**
0	Never
1	On programmer errors only
2	On environmental conditions only
3	On all exceptional conditions

See also: Exception handlers, exception mode, *System Concepts*

`job_flags`

Specifies whether the Nucleus checks the validity of objects used as parameters in system calls.

Bits	**Meaning**
15-2	Must be set to 0
1	If 0, the Nucleus checks the validity of objects
0	Must be set to 0

`task_priority`

Specifies the priority of the new job's initial task.

Value	Meaning
0	Priority equals the maximum priority of the EIOS initial job.
not 0	The priority of the new job's initial task. If this priority is higher (numerically lower) than the maximum priority of the EIOS initial job, an E_PARAM error occurs.

`task_flags`

Indicates:

Bits	Value	Meaning
15-2	0	Reserved, set to 0
1	0	The task starts immediately.
	1	The task is suspended until **start_io_job** occurs.
0	0	The task does not use floating point instructions.
	1	The task uses floating-point instructions

`msg_mbox`

A token for a mailbox that receives the LRS after the loading operation completes. Each call to **rqe_a_load_io_job** requires a unique and valid mailbox; do not use a null selector.

This parameter also receives an exit message from the newly created I/O job.

See also: `msg_mbox` parameter, EIOS call **rqe_create_io_job**

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

Rqe_a_load_io_job creates a new job using **rqe_create_io_job** and loads the specified object file. The loaded file's code becomes the initial task of the new job. The calling task continues to run during the loading operation. If the `task_flags` parameter specifies delayed start, use **start_io_job** to start the new task. Otherwise, the task becomes ready at the end of the loading operation.

During the sequential part of this call the AL:

- Checks the validity of the target file's header record.
- Creates an I/O job. This I/O job is a child of the calling task's job.
- Returns a condition code reflecting the success or failure of the first phase.

The concurrent part of this call runs as the initial task in the new job, and:

- Loads the file designated by the `connection` parameter from secondary storage into main memory.
- Creates the initial task. If there are no errors while the file is loaded, the task can start running.
- Sends an LRS to the mailbox specified by the `msg_mbox` parameter.
- Deletes itself.

See also: Sequential and concurrent parts of an asynchronous system call, *System Concepts*

Loader Result Segment

The LRS has this structure:

```
DECLARE io_job_lrs STRUCTURE(
   termination_code       WORD_16,
   except_code            WORD_16,
   job_token              SELECTOR,
   return_data_len        BYTE,
   reserved_word1         WORD_16,
   reserved_byte          BYTE,
   reserved_word2         WORD_16,
   mem_requested          WORD_16,
   mem_received           WORD_16);
```

or

```
typedef struct {
   UINT_16                termination_code;
   UINT_16                except_code;
   SELECTOR               job_token;
   UINT_8                 return_data_len;
   UINT_16                reserved_word1;
   UINT_8                 reserved_byte;
   UINT_16                reserved_word2;
   UINT_16                mem_requested;
   UINT_16                mem_received;
} IO_JOB_LRS_STRUCT
```

Where:

`termination_code`

Indicates the success or failure of the loading operation. If failure is shown, delete the newly created I/O job; the AL doesn't do so.

Value	**Meaning**
100H	Success
002H	Failure

`except_code`

The concurrent condition code.

`job_token` A token for the newly created I/O job.

`return_data_len`

Indicates the length of the remainder of the data structure, minus 13 bytes.

`mem_requested`

Indicates the number of 16-byte paragraphs the target file requested for the new job, including the memory needed for all segments and the job's memory pool. If more than 1 megabyte was requested, this field will contain 0FFFFH.

`mem_received`

Indicates the number of 16-byte paragraphs actually allocated to the new job. If more than 1 megabyte was allocated, this field will contain 0FFFFH.

See also: Exit messages, *Systems Concepts*

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_CONN_NOT_OPEN	0034H	The AL opened the connection, but some other task closed the connection before the loading operation began.
E_CONN_OPEN	0035H	The specified connection was already open.
E_CONTEXT	0005H	The calling task's job is not an I/O job.
E_DEV_DETACHING	0039H	The device containing the specified file is being detached. If the device is a remote device, a retry may succeed.
E_EXIST	0006H	At least one of these is true: • The `connection` parameter is not a token for an existing object. • The calling task's job has no global job. See also: Global job, *System Concepts* • The msg_mbox parameter does not refer to an existing object.
E_FACCESS	0026H	The specified connection does not have read access to the file.
E_FLUSHING	002CH	The device containing the target file is being detached.
E_IO_HARD	0052H	A hard I/O error occurred. A retry is probably useless.

E_IO_OPRINT	0053H	The device containing the target file is off-line. Operator intervention is required.
E_IO_SOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation and failed; a retry may succeed.
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_JOB_PARAM	8060H	The pool_max parameter is both non-0 and smaller than the pool_min parameter.
E_JOB_SIZE	006DH	The pool_max parameter is non-0 and too small for the target file.
E_LOADER_SUPPORT	006FH	The target file requires capabilities not configured into the AL.
E_MEM	0002H	The memory available to the calling task's job or the BIOS is not sufficient to complete the call.
E_NO_LOADER_MEM	0067H	The memory pool of the newly created I/O job does not currently have a block of memory large enough to allow the AL to run.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NOT_FILE_CONN	0032H	The specified connection is to a device rather than to a named file.
E_PARAM	8004H	Either the task_priority is invalid (higher than the maximum priority of the EIOS initial job) or the value of the exception_mode field in the exception handler structure is outside the range 0-3.
E_SHARE	0028H	The calling task tried to open a connection to a file already being used by some other task, and the file's sharing attribute is not compatible with the open request.
E_SUPPORT	0023H	The specified connection was not created in this job.

E_TIME	0001H	The calling task's job is not an I/O job.
E_TYPE	8002H	The connection parameter is not a token for a connection.
E_SLOT	000CH	The GDT has no available slots.

Concurrent Condition Codes: returned to except_code in the LRS after loading attempt

E_OK	0000H	No exceptional conditions occurred.
E_DEV_DETACHING	0039H	The device containing the specified file is being detached. If the device is a remote device, a retry may succeed.
E_EOF	0065H	The call encountered an unexpected EOF.
E_EXIST	0006H	At least one of these is true: • The mailbox specified was deleted before the loading operation completed. • The device containing the target file was detached before the loading operation completed.
E_FACCESS	0026H	The default user of the newly created I/O job does not have read access to the target file.
E_FLUSHING	002CH	The device containing the target file is being detached.
E_IO_HARD	0052H	A hard I/O error occurred. A retry is probably useless.
E_IO_OPRINT	0053H	The device containing the target file is off-line. Operator intervention is required.
E_IO_SOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation and failed; a retry may succeed.
E_IO_UNCLASS	0050H	An unknown I/O error occurred.

E_LIMIT	0004H	At least one of these is true: • The `task_priority` parameter is higher (numerically lower) than the newly created I/O job's maximum priority. See also: For ICU-configurable systems, I/O jobs, *ICU User's Guide and Quick Reference* • Either the newly created I/O job, or its default user, is already involved in 255 I/O operations. • The calling task's object directory is full. • The root object directory is full.
E_NO_LOADER_MEM	0067H	There is not enough memory available to the newly created I/O job or the BIOS to allow the AL to run.
E_NO_START	006CH	The target file does not specify the entry point for the program being loaded.
E_PARAM	8004H	The target file has a stack smaller than 16 bytes.

s_load_io_job

Obsolete. Synchronously loads an object file and creates an I/O job for it. This call description is identical to **rqe_s_load_io_job**; **s_load_io_job** is provided for compatibility with older versions of the iRMX OS.

See also: **rqe_s_load_io_job**

Syntax, PL/M and C

```
job = rq$s$load$io$job (path_ptr, pool_min, pool_max,
      except_handler, job_flags, task_priority, task_flags,
      msg_mbox, except_ptr);
```

```
job = rq_s_load_io_job (path_ptr, pool_min, pool_max,
      except_handler, job_flags, task_priority, task_flags,
      msg_mbox, except_ptr);
```

rqe_s_load_io_job

Synchronously creates an I/O job containing the AL task, which loads the code for the user task from secondary storage. For segmented applications, only tasks running within I/O jobs should invoke this call. However, you must use this call to load standalone, linked flat model applications instead of creating an I/O job in flat model.

Syntax, PL/M and C

```
job = rqe$s$load$io$job (path_ptr, pool_min, pool_max,
      except_handler, job_flags, task_priority, task_flags,
      msg_mbox, except_ptr);
```

```
job = rqe_s_load_io_job (path_ptr, pool_min, pool_max,
      except_handler, job_flags, task_priority, task_flags,
      msg_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
job	SELECTOR	SELECTOR
path_ptr	POINTER	STRING far *
pool_min	WORD_32	UINT_32
pool_max	WORD_32	UINT_32
except_handler	POINTER	EXCEPTION_STRUCT far *
job_flags	WORD_16	UINT_16
task_priority	BYTE	UINT_8
task_flags	WORD_16	UINT_16
msg_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

`job` A token for the newly created I/O job, only valid if E_OK returns.

Parameters

`path_ptr`

A pointer to a STRING containing a pathname for the named file with the object code to be loaded. The pathname must conform to the EIOS pathname syntax for named files.

See also: Pathname syntax, *System Concepts* `pool_min`

Specifies the minimum size of the new job's memory pool in 16-byte paragraphs. The upper limit is 4 Gbytes.

`pool_max`

Specifies the maximum allowable size of the new job's memory pool in 16-byte paragraphs. The upper limit is 4 Gbytes.

`except_handler`

A pointer to this structure:

```
DECLARE exception STRUCTURE (
   exception_handler_ptr              POINTER,
   exception_mode                     BYTE);
```

or

```
typedef struct {
   void far *           exception_handler_ptr;
   UINT_8               exception_mode;
} EXCEPTION_STRUCT;
```

Where:

`exception_handler_ptr`

If not null, references the first instruction of the new job's own exception handler. If null, the new job's exception handler is the system default exception handler. The exception handler for the new task becomes the default exception handler for the job.

`exception_mode`

Indicates when control is to be passed to the exception handler. It is encoded as:

Value	**When Control Passes To Exception Handler**
0	Never
1	On programmer errors only
2	On environmental conditions only
3	On all exceptional conditions

See also: Exception handlers, exception mode, *System Concepts*

`job_flags`

Specifies whether the Nucleus checks the validity of objects used as parameters in system calls.

Bits	**Meaning**
15-2	Must be set to 0
1	If 0, the Nucleus checks the validity of objects
0	Must be set to 0

`task_priority`

Specifies the priority of the new job's initial task.

Value	**Meaning**
0	Priority equals the maximum priority of the EIOS initial job.
not 0	The priority of the new job's initial task. If this priority is higher (numerically lower) than the maximum priority of the EIOS initial job, an E_PARAM error occurs.

`task_flags`

Indicates:

Bits	**Value**	**Meaning**
15-2	0	Reserved, set to 0
1	0	The task starts immediately.
	1	The task is suspended until **start_io_job** occurs.
0	0	The task does not use floating point instructions.
	1	The task uses floating-point instructions

`msg_mbox`

A token for a mailbox that receives an exit message from the newly created I/O job. Each call to **rqe_s_load_io_job** requires a unique and valid mailbox; do not use a null selector.

See also: `msg_mbox` parameter, EIOS call **create_io_job**

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The calling task's job is not an I/O job.
E_DEV_DETACHING	0039H	The device containing the specified file is being detached. If the device is a remote device, a retry may succeed.
E_EOF	0065H	The call encountered an unexpected EOF.
E_EXIST	0006H	At least one of these is true: • The `msg_mbox` parameter is not a token for an existing object. • The calling task's job has no global job. See also: Global job, *System Concepts* • The device containing the target file was detached.

E_FACCESS	0026H	The default user object for the new I/O job does not have read access to the specified file.
E_FNEXIST	0021H	The specified target file, or some file in the specified path, does not exist or is marked for deletion.
E_FLUSHING	002CH	The device containing the target file is being detached.
E_INVALID_FNODE	003DH	The fnode for the specified file is invalid, so the file must be deleted.
E_IO_HARD	0052H	A hard I/O error occurred. A retry is probably useless.
E_IO_JOB	0047H	The EIOS could not create an I/O job because the default directory size (DDS) configuration parameter is too small.
E_IO_OPRINT	0053H	The device containing the target file is off-line. Operator intervention is required.
E_IO_SOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation and failed; a retry may succeed.
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_JOB_PARAM	8060H	The pool_max parameter is not 0 and smaller than the pool_min parameter.
E_JOB_SIZE	006DH	The pool_max parameter is not 0 and too small for the target file.
E_LIMIT	0004H	At least one of these is true: • The `task_priority` parameter is higher (numerically lower) than the newly created I/O job's maximum priority. See also: For ICU-configurable systems, I/O jobs, *ICU User's Guide and Quick Reference* • Either the newly created I/O job or its default user object is already involved in 255 I/O operations.
E_LOADER_SUPPORT	006FH	The target file requires capabilities not configured into the AL.

E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E_NO_LOADER_MEM	0067H	The memory pool of the newly created I/O job does not currently have a block of memory large enough to run the AL.
E_SLOT	000CH	The GDT has no available slots.
E_NO_START	006CH	The target file does not specify the entry point for the program being loaded.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	At least one of these is true: • The value of the `exception_mode` field in the `except_handler` structure is outside the range of 0 to 3. • The task_priority is higher than the maximum priority of the EIOS initial job. • The target file requested a stack smaller than 16 bytes.
E_PATHNAME_SYNTAX	003EH	The specified pathname contains one or more invalid characters.
E_SUPPORT	0023H	The specified connection is not in this job.
E_TIME	0001H	The calling task's job is not an I/O job.
E_TYPE	8002H	The connection parameter is not a token for a connection.

s_overlay

Synchronously loads overlay modules for 16-bit (OMF286) programs. Not valid for 32-bit programs.

Syntax, PL/M and C

```
CALL rq$s$overlay (name_ptr, except_ptr);

rq_s_overlay (name_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
name_ptr	POINTER	STRING far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

name_ptr

A pointer to a STRING containing the name of an overlay. Use only uppercase letters, both here and in the overlay definition file.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Root modules issue this system call when they want to load an overlay module. The root module must be loaded using one of the system calls that create an I/O job.

The condition code is returned to the calling task.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_DEV_DETACHING	0039H	The device containing the specified file is being detached. If the device is a remote device, a retry may succeed.
E_EOF	0065H	The call encountered an unexpected EOF.
E_EXIST	0006H	The specified device does not exist.
E_FLUSHING	002CH	The device containing the target file is being detached.

E_IO_HARD	0052H	A hard I/O error occurred. A retry is probably useless.
E_IO_OPRINT	0053H	The device containing the target overlay is off-line. Operator intervention is required.
E_IO_SOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation and failed; a retry may succeed.
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_LIMIT	0004H	Either the calling task's job, or its default user object, is already involved in 255 I/O operations.
E_NOMEM	0068H	The memory pool of the new I/O job does not have a block of memory large enough for the AL to load the overlay module.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_OVERLAY	006EH	The overlay name indicated by the name_ptr parameter does not match any overlay module name in the overlay definition file.
E_SUPPORT	0023H	At least one of these is true: • The specified connection is not in this job. • The calling task is a 16-bit task attempting to load a 32-bit object which contains either a code or stack offset larger than 64 Kbytes.

□ □ □

Basic I/O System Calls 3

a_attach_file

Creates a connection to an existing file of any type.

Syntax, PL/M and C

```
CALL rq$a$attach$file (user, prefix, subpath_ptr, resp_mbox,
      except_ptr);
```

```
rq_a_attach_file (user, prefix, subpath_ptr, resp_mbox,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
user	SELECTOR	SELECTOR
prefix	SELECTOR	SELECTOR
subpath_ptr	POINTER	STRING far *
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

`user` A token for the user object to be inspected during access checking of named files. A null selector specifies the default user object. The BIOS ignores this parameter when you attach physical, stream, or DOS files because the user is always World.

`prefix`

A token for the connection object to be used as the path prefix. A null selector specifies the default prefix.

`subpath_ptr`

A pointer to a STRING containing the subpath of the named 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. The BIOS ignores the `subpath_ptr` parameter for physical and stream files.

`resp_mbox`

A token for the mailbox where the BIOS places the result object of the call. This result object is a token for a new file connection if the call succeeds, or an I/O Result Segment (IORS). To determine the type of object returned, use the Nucleus system call **get_type**.

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

Once the connection is established, it remains in effect until the program deletes the connection object or the creating job. Once attached, the file can be opened, closed, read, or written to multiple times. **A_attach_file** has no effect on the owner ID or the access list for the file.

The BIOS does not check the access rights of an iRMX-NET remote file when you create a connection to the file, but checks during operations on the connection. This won't affect your programs if you do this:

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

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_DEV_OFFLINE	002EH	The prefix parameter references a logical connection to a device. One of these is true of this device: • It has been physically attached but is now off-line. • It has been logically attached but never physically attached. See also: Connections, *System Concepts* • An unspecified DOS error occurred.

E_EXIST	0006H	One of these is true: • One or more of the `user`, `prefix`, or `resp_mbox` parameters is not a token for an existing object. • The `prefix` connection is being deleted. • The connection to a remote driver is no longer active.
E_LIMIT	0004H	Processing this call would exceed one or more of these limits: • The object limit for this job • 255 outstanding I/O operations for the specified user object • 255 outstanding I/O operations for the caller's job • 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 the call.
E_NOPREFIX	8022H	The calling task specified a default prefix using a null selector, but a default prefix cannot be found for one of these reasons: • When this job was created, a 0 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	The user parameter is not a null selector, and is not a token for a user object. Otherwise, it specifies a default user, but no default user can be found for one of these reasons: • When this job was created, a 0 was specified for its object directory, so the job cannot catalog a default user. • The job's directory can have entries but a default user is not cataloged there. • The cataloged object *r?iouser* is not a user object. Treat *r?iouser* as a reserved word.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

E_PATHNAME_SYNTAX	003EH	One or more of these is true: • The specified pathname contains invalid characters or has 0 length. • The subpath of the specified remote file exceeds 127 bytes.
E_TYPE	8002H	Either the prefix parameter is not a connection or logical device object created by the EIOS, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

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 subpath_ptr contains a filename that is not the name of a directory. 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; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*
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. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*
E_IO_MEM	0042H	The memory available to the BIOS job is not sufficient to complete the call.
E_LIMIT	0004H	Processing this call would deplete the remote server's resources.
E_NAMEN_EXIST	0049H	The user object is not for a verified user or is not in the remote server's User Definition File (UDF).

E_PASSWORD_MISMATCH	004BH	The user object password does not match the password of the user defined on the remote server.
E_PATHNAME_SYNTAX	003EH	The syntax of the specified remote file pathname is illegal; it must follow the naming conventions of the server.
E_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF.

a_change_access

Changes the access rights to a named data or directory file.

Syntax, PL/M and C

```
CALL rq$a$change$access (user, prefix, subpath_ptr, ID, access,
      resp_mbox, except_ptr);
```

```
rq_a_change_access (user, prefix, subpath_ptr, ID, access,
      resp_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
user	SELECTOR	SELECTOR
prefix	SELECTOR	SELECTOR
subpath_ptr	POINTER	STRING far *
ID	WORD_16	UINT_16
access	BYTE	UINT_8
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

`user` A token for the user object to be inspected in access checking. A null selector specifies the default user object. For DOS files, the BIOS ignores this parameter because the user is always World.

`prefix`

A token for the connection object to be used as the path prefix. A null selector specifies the default prefix.

`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 designates the desired file.

`ID` 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 up to three ID-access pairs. For DOS files and directories, since the user is always World, no IDs can be added or deleted. For NFS files, user IDs may be mapped differently between different OSs.

See also: Accessing NFS Files, Chapter 17, *System Concepts*

access

The new access rights for the ID. Setting all the bits to 0 removes the specified ID from the access list of the file. If not 0, the meaning of the various bit settings depends upon whether the file is a data file or a directory file. The following tables show the access rights for data and directory files. Setting a bit to 1 enables access, 0 denies access. For NFS files, access rights may be mapped differently between different OSs. The World user always has read (list) access to DOS files and directories; write (delete, append, and update) access is optional.

Bits	**Data File Access Rights**
7-4	Reserved; set to 0.
3	Update: permission to write over any information in the file by using **a_write** or **s_write_move**, and permission to truncate the file using **a_truncate** or **s_truncate_file**. This does not include permission to add information to the EOF. Set to the same value as bit 2 for remote files.
2	Append: permission to write information only at the EOF by using **a_write** or **s_write_move**. Set to the same value as bit 3 for remote files.
1	Read: permission to read data from the file by using **a_read** or **s_read_move**.
0	Delete: permission to delete the entire file by using **a_delete_file** or **s_delete_file**. Also enables changing the name of the file by using **a_rename_file** or **s_rename_file**. The BIOS ignores this bit for remote files.

Bits	**Directory File Access Rights**
7-4	Reserved; set to 0.
3	Change entry: permission to change the access list associated with a file in the directory, using **a_change_access** or **s_change_access**. This does not include permission to change the access list of the directory itself. The BIOS ignores this bit for remote directories.
2	Add-entry: permission to add files to the directory by using **a_create_file, a_create_directory, a_rename_file, s_create_file, s_create_directory**, or **s_rename_file**. This does not include permission to change existing files in the directory.
1	List: permission to read information from the directory by using **a_read**, **a_get_directory_entry**, or **s_read_move**.
0	Delete: permission to delete the directory by using **s_delete_file** or **a_delete_file**. Also enables changing the name of the directory by using **s_rename_file** or **a_rename_file**. The BIOS ignores this bit for remote directories.

`resp_mbox`

A token for the mailbox that receives an IORS indicating the result of the call. A null selector means that you do not want to receive an IORS.

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

If the owner is World (0FFFFH), any task may change the access mask of the file. Otherwise, the caller must be the owner of the file or must have change-entry access to the file's parent directory. If this system has system manager support configured by the ICU, user 0 may change the access rights of any file regardless of which user is the owner.

See also: System manager ID, *ICU User's Guide and Quick Reference*

This call has no effect on existing connections to the file. Depending on the contents of the `ID` and `access` parameters, users may be added to or deleted from an iRMX file's ID-access mask list, or the access privileges granted to a particular user may be changed.

You cannot change the access rights of a virtual root directory, because a virtual root directory has no assigned owner. Otherwise, an E_FACCESS condition code returns.

For DOS files, the World user cannot be changed, and list (read) access is automatic. Only write access is optional.

For NFS files on DOS or Unix, access rights are mapped differently than on iRMX systems.

See also: Accessing NFS files, Chapter 17, System Concepts

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_DEV_OFFLINE	002EH	The prefix parameter references a logical connection to a device. One of these is true of this device: • It has been physically attached but is off-line. • It has been logically attached but never physically attached. • An unspecified DOS error occurred.
E_EXIST	0006H	At least one of these is true: • One or more of the user, prefix, or resp_mbox parameters is not a token for an existing object. • The prefix connection is being deleted. • The remote driver connection is no longer active.
E_IFDR	002FH	The prefix and subpath_ptr parameters specify a type of file other than a named file.
E_LIMIT	0004H	Processing this call would exceed one or more of these limits: • The object limit for this job • 255 outstanding I/O operations for the specified user object • 255 outstanding I/O operations for the caller's job • The number of outstanding I/O operations for a remote file
E_MEM	0002H	The memory available to the calling task's job is insufficient to complete this call.

E_NOPREFIX	8022H	The calling task specified a default prefix using a null selector, but a default prefix cannot be found for one of these reasons: • When this job was created, a 0 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 is not a null selector, the parameter is not a token for a user object. Otherwise it specifies a default user, but no default user can be found for one of these reasons: • When this job was created, a 0 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 which is cataloged with the name *r?iouser* is not a user object. Treat *r?iouser* as a reserved word.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PATHNAME_SYNTAX	003EH	One or more of these is true: • The specified pathname contains invalid characters. • The subpath of the specified remote file exceeds 127 bytes.
E_SUPPORT	0023H	The connection was not created by this job.
E_TYPE	8002H	One or more of these is true: • The user token designates a connection of the wrong type. • The `prefix` parameter is not a token for a connection object or a logical device object created by the EIOS. • The resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
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 that is not a directory. 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; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*
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. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*
E_IO_MEM	0042H	The memory available is not sufficient to complete this call.
E_LIMIT	0004H	Processing this call would deplete the remote server's resources.
E_NAME_NEXIST	0049H	The user object does not represent a verified user or is not properly defined in the remote server's UDF.
E_NOT_FILE_CONN	0032H	The subpath_ptr parameter = NIL and the prefix parameter is not a file connection.
E_PASSWORD_MISMATCH	004BH	The user object password does not match the password of the user defined on the remote server.

E_PATHNAME_SYNTAX	003EH	The syntax of the specified remote file pathname is illegal; it must follow the naming conventions of the server.
E_SUPPORT	0023H	The call attempted to add another access ID to the list of access IDs that already contained the limit of three IDs.
E_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF.

a_close

Closes an open file connection for any type of file.

Syntax, PL/M and C

```
CALL rq$a$close (connection, resp_mbox, except_ptr);
```

```
rq_a_close (connection, resp_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection`	SELECTOR	SELECTOR
`resp_mbox`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection`

A token for the file connection to be closed.

`resp_mbox`

A token for the mailbox that receives an IORS. A null selector means that you do not want to receive an IORS.

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

Use this call when the application needs to change the open or share mode of the connection. The BIOS will not close the connection until all existing I/O requests for the connection have been satisfied. In addition, the BIOS will not send a response to the response mailbox until the file is closed.

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	At least one of these is true: • One or more of the `connection` or `resp_mbox` parameters is not a token for an existing object. • The connection is being deleted. • The connection for a remote driver is no longer active.
E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its object limit. • 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_CONFIGURED	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	Either the connection parameter is not a token for a connection object, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
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

Creates a named directory file and returns a token for the new file connection.

Syntax, PL/M and C

```
CALL rq$a$create$directory (user, prefix, subpath_ptr, access,
      resp_mbox, except_ptr);
```

```
rq_a_create_directory (user, prefix, subpath_ptr, access,
      resp_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
user	SELECTOR	SELECTOR
prefix	SELECTOR	SELECTOR
subpath_ptr	POINTER	STRING far *
access	BYTE	UINT_8
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

user A token for the user object of the new directory's owner. The BIOS makes sure the caller has add-entry access to the parent of the new directory. A null selector specifies the default user object. For DOS files, the BIOS ignores this parameter because the user is always World.

prefix

A token for the connection to be used as the path prefix. A null selector specifies the default prefix.

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 reference an unused location in the directory tree.

access

The owner's initial access rights to the directory. For each bit, a 1 grants access and a 0 denies it.

Bits	**Data File**	**Directory File**
7-4	Reserved (0)	Reserved (0)
3	Update	Change-entry
2	Append	Add-entry
1	Read	List
0	Delete	Delete

The DOS World user always has read (list) access to DOS files and directories; write access is optional.

See also: **a_change_access**, EIOS call **s_change_access**

resp_mbox

A token for the mailbox that receives a directory file connection if the call succeeded, otherwise an IORS. To determine the type of object returned, use the Nucleus system call **get_type**.

except_ptr

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

This call cannot create a connection to an existing directory; use **a_attach_file**.

You cannot create an iRMX-NET remote directory with a virtual root directory as its parent because a virtual root directory has no assigned owner. Otherwise, an E_FACCESS condition code returns.

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK 0000H No exceptional conditions occurred.

E_DEV_OFF_LINE 002EH The prefix parameter references a logical connection to a device. One of these is true of the device:

- It has been physically attached but is now off-line.
- It has been logically attached but never physically attached.

  See also: **attachdevice**, *System Concepts*

E_EXIST 0006H At least one of these is true:

- One or more of the `user`, `prefix`, or `resp_mbox` parameters is not a token for an existing object.
- 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 exceed one or more of these limits:

- The object limit for this job
- 255 outstanding I/O operations for the specified user object
- 255 outstanding I/O operations for the caller's job
- 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 task specified a default prefix using a null selector, but a default prefix cannot be found because of one or more of these reasons: • When this job was created, a 0 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 is not a null selector, the parameter is not a user object. Otherwise, it specifies a default user, but no default user can be found for one of these reasons: • When this job was created, a 0 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 cataloged object *r?iouser* is not a user object. Treat *r?iouser* as a reserved word.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PATHNAME_SYNTAX	003EH	One or more of these is true: • The specified pathname contains invalid characters or has 0 length. • The subpath of the specified remote file exceeds 127 bytes.
E_TYPE	8002H	Either the prefix parameter is not a token for a connection object or a logical device object created by the EIOS, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

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 does not have add-entry access to the parent directory.
E_FEXIST	0020H	A file with the specified pathname 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.
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 cannot be accessed; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*
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. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*
E_IO_MEM	0042H	The memory available to the BIOS job is not sufficient to complete this call.
E_LIMIT	0004H	Processing this call would deplete the remote server's resources.
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 UDF.
E_PASSWORD_MISMATCH	004BH	The user object password does not match the password of the user defined on the remote server.
E_PATHNAME_SYNTAX	003EH	The syntax of the specified remote file pathname is illegal; it must follow the naming conventions of the server.

E_SPACE	0029H	At least one of these is true: • The volume is full. • 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 BIOS is not configured to support space allocation.
E_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF.

a_create_file

Creates a physical, stream, or named data file and returns a token for the new file connection.

Syntax, PL/M and C

```
CALL rq$a$create$file (user, prefix, subpath_ptr, access,
      granularity, size, must_create, resp_mbox, except_ptr);
```

```
rq_a_create_file (user, prefix, subpath_ptr, access,
      granularity, size, must_create, resp_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
user	SELECTOR	SELECTOR
prefix	SELECTOR	SELECTOR
subpath_ptr	POINTER	STRING far *
access	BYTE	UINT_8
granularity	WORD_16	UINT_16
size	WORD_32	UINT_32
must_create	BYTE	UINT_8
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

`user` A token for the user object of the new file's owner, which provides the user ID for access checking. A null selector specifies the default user object. The caller must have add-entry access to the parent of the new directory. The BIOS ignores this parameter for physical, stream, or DOS files because the user is always World.

`prefix`

A token for a device or file connection. A null selector specifies the default prefix. The file created by this call is of the type that is associated with this parameter. 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 and DOS 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 a null pointer, when using a named, DOS, or EDOS file driver, creates an unnamed temporary file. The BIOS automatically deletes this file when the last connection to it is deleted.

`access`

The owner's initial access rights to the new file. This parameter does not apply to physical or stream files. For each bit, a 1 grants access and a 0 denies it.

Bits	**Meaning**
7-4	Reserved, set to 0
3	Update
2	Append
1	Read
0	Delete

The DOS World user always has read (list) access to DOS files and directories; write access is optional.

See also: **a_change_access**

`granularity`

The size of each logical block of space to be allocated to the file. The BIOS ignores this parameter for physical, stream, remote, and DOS files. If necessary, this parameter is rounded up to a multiple of the volume granularity.

Value	**Meaning**
0	Same as volume granularity
1-0FFFEH	Number of bytes per allocation
0FFFFH	The file must be contiguous

When a contiguous file is extended, space is allocated in volume-granularity units. A contiguous file can become noncontiguous when it is extended.

`size`

The number of bytes initially reserved for the file. For stream files and existing remote files, this value must equal 0. If you make this value greater than 0 for stream files, the reserved space may contain unknown data. The BIOS ignores this parameter for physical files and non-existent remote files.

`must_create`

Determines the handling of an existing file. This parameter applies to named files and DOS files. Only the least significant bit is checked.

See also: For ICU-configurable systems, Ability to create existing files, *ICU User's Guide and Quick Reference*

Value	**Meaning**
0	If a data file exists, it will be truncated or expanded based on the size parameter. The file's owner ID and access list are unchanged. If a directory or device file exists, a temporary file is created. The BIOS automatically deletes this file when the last connection to it is deleted. Because this file is created without a path, it can be accessed only through a connection.
1	If a file exists, an E_FEXIST condition code returns.

`resp_mbox`

A token for the mailbox that receives a new file connection if the call succeeds, otherwise an IORS. To determine the type of object returned, use the Nucleus system call **get_type**.

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

Any task can create a temporary file in any directory because temporary files are not listed as ordinary entries in the directory. No add-entry access is required.

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 pathname, as well as through connections to the file.

You cannot create an iRMX-NET remote file with a virtual root directory as its parent because a virtual root directory has no owner and you cannot write to it. An attempt to do so returns an E_FACCESS condition code.

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK 0000H No exceptional conditions occurred.

E_DEV_OFF_LINE 002EH The prefix parameter in this system call references a logical connection to a device. One of these is true of this device:

- It has been physically attached but is now off-line.
- It has been logically attached but never physically attached.

See also: **attachdevice**, *Command Reference*

E_EXIST 0006H At least one of these is true:

- One or more of the `user`, `prefix`, or `resp_mbox` parameters is not a token for an existing object.
- 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 both of these limits to be exceeded:

- The object limit for this job
- 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 using a null selector, but it cannot be found for one of these reasons:

- When the job was created, a 0 was specified for its object directory.
- No default prefix is cataloged in the job's directory.

E_NOUSER 8021H If the user parameter is not a null selector, the parameter is not a token for a user object. Otherwise, it specifies a default user, but no default user can be found for one of these reasons:

- When this job was created, a 0 was specified for its object directory, so the job cannot catalog a default user.
- The job's directory can have entries but a default user is not cataloged there.
- The cataloged object *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 these is true:

- The specified pathname contains invalid characters or has 0 length.
- The subpath of the specified remote file exceeds 127 bytes.

E_TYPE 8002H At least one of these is true:

- The `prefix` parameter is not a token for a connection object or a logical device object created by the EIOS.
- The resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_DEV_DETACHING 0039H The file specified is on a device that the system is detaching.

E_FACCESS 0026H One of these 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 is 1, and the file already exists.

E_FNEXIST	0021H	A file in the specified path does not exist or is marked for deletion.
E_FNODE_LIMIT	003FH	The file cannot be created or extended to this size because it has reached the maximum number of volume blocks. See also: File driver limitations, *System Concepts* manual.
E_FRAGMENTATION	0030H	The disk is too fragmented to extend the file.
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 cannot be accessed; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*
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. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*
E_IO_MEM	0042H	The memory available to the BIOS job is not sufficient to complete this call.
E_LIMIT	0004H	Processing this call would deplete the remote server's resources.
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 UDF.
E_PASSWORD_MISMATCH	004BH	The user object password does not match the password of the user defined on the remote server.

E_PATHNAME_SYNTAX	003EH	The syntax of the specified remote file pathname is illegal; it 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 share-with-readers-only share mode.
E_SPACE	0029H	At least one of these is true: • The volume is full. • 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 these is true: • The BIOS is not configured to allow truncation of files to 0 size. • The BIOS is not configured to allow space allocation on volumes. • The remote file driver does not support creation of a contiguous file. • The remote file driver does not support truncating existing remote files to 0 size.
E_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF.

create_user

Creates a user object, accepts a list of IDs, and returns a token for the new object.

Syntax, PL/M and C

```
user = rq$create$user (IDs_ptr, except_ptr);

user = rq_create_user (IDs_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`user`	SELECTOR	SELECTOR
`IDs_ptr`	POINTER	IDS_STRUCT far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`user` The new user object token.

Parameters

`IDs_ptr`

A pointer to this structure:

```
DECLARE IDs STRUCTURE(
   length                  WORD_16,
   count                   WORD_16,
   IDs(*)                  WORD_16);
```

or

```
typedef struct {
   UINT_16                 length;
   UINT_16                 count;
   UINT_16                 ids[2]; /* adjust to count value */
} IDS_STRUCT;
```

Where:

`length` Number of elements in the ID array.

`count` Number ranging from 1 to `length` of IDs to be included in the user object.

`IDs` Array of IDs, each of which is included in the user object. The first ID is the owner of any file created with reference to this user object.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If the number of ID slots specified by the `length` field is greater than the number of IDs specified by the `count` field, `length` is adjusted to equal `count`.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
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 is either 0 or is greater than the length field.

a_delete_connection

Deletes a file connection created by **a_create_file**, **a_create_directory**, or **a_attach_file**. Use with any type of file..

Syntax, PL/M and C

```
CALL rq$a$delete$connection (connection, resp_mbox,
      except_ptr);
```

```
rq_a_delete_connection (connection, resp_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection`	SELECTOR	SELECTOR
`resp_mbox`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection`

A token for the file connection to be deleted.

`resp_mbox`

A token for the mailbox that receives an IORS. A null selector means that you do not want to receive an IORS.

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

Delete connections when they are no longer needed. This call deletes a connection object and deletes the associated file if both of these are true:

- The file is already marked for deletion by a previous **a_delete_file** call or is a temporary 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.

See also: **a_create_file**, **a_create_directory**, **a_attach_file**

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	At least one of these is true: • One or more of the `connection` or `resp_mbox` parameters is not a token for an existing object. • The connection is being deleted. • The connection for a remote driver is no longer active.
E_LIMIT	0004H	The calling task's job has already reached its object limit, or DOS has run out of file handles.
E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete this call.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NOT_FILE_CONN	0032H	The specified connection is a device connection, not a file connection.
E_SUPPORT	0023H	The specified connection was not created by this job.
E_TYPE	8002H	Either the connection parameter is not a token for a connection object, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
E_IO	002BH	An I/O error occurred, but the connection was still deleted.

a_delete_file

Marks a file for deletion and deletes it. The file type may be stream, named data, named directory, DOS data, or DOS directory.

Syntax, PL/M and C

```
CALL rq$a$delete$file (user, prefix, subpath_ptr, resp_mbox,
      except_ptr);

rq_a_delete_file (user, prefix, subpath_ptr, resp_mbox,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
user	SELECTOR	SELECTOR
prefix	SELECTOR	SELECTOR
subpath_ptr	POINTER	STRING far *
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

user
A token for the user object to be inspected in access checking. A null selector specifies the default user object. This parameter does not apply to stream files. For DOS files, the BIOS ignores this parameter because the user is always World.

prefix
A token for the connection object to be used as the path prefix. A null selector specifies the default prefix.

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.

resp_mbox
A token for a mailbox that receives an IORS when the file is marked for deletion. The file will not actually be deleted until all connections to the file are deleted. A null selector means that you do not want to receive an IORS.

except_ptr
A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

For iRMX files, the caller must have delete access to the file. For DOS files, the caller must have write access to the file.

See also: **a_change_access**, **s_change_access**

Use this call to mark the designated file for deletion and remove 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.

See also: **a_delete_connection**

You cannot delete an iRMX-NET remote file with a virtual root directory as its parent because a virtual root directory has no owner and you cannot write to it. An attempt to do so returns an E_FACCESS condition code.

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_DEV_OFF_LINE	002EH	The prefix parameter references a logical connection to a device. One of these is true of this device: • It has been physically attached but is now off-line. • It has been logically attached but never physically attached. See also: **attachdevice**, *Command Reference*
E_EXIST	0006H	At least one of these is true: • One or more of the `user`, `prefix`, or `resp_mbox` parameters is not a token for an existing object. • 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 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 these limits: • The object limit for this job • 255 outstanding I/O operations for the specified user object • 255 outstanding I/O operations for the caller's job • 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 using a null selector, but a default prefix cannot be found for one of these reasons: • When this job was created, a 0 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 is not a null selector, the parameter is not a token for a user object. Otherwise, it specifies a default user, but no default user can be found for one of these reasons: • When this job was created, a 0 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 cataloged object *r?iouser* is not a user object. Treat *r?iouser* 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 these is true: • The specified pathname contains invalid characters or has 0 length. • The subpath of the specified remote file exceeds 127 bytes.

E_SUPPORT	0023H	The specified connection was not created by this job.
E_TYPE	8002H	At least one of these is true: • The `prefix` parameter is not a token for a connection object or a logical device object created by the EIOS. • The resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
E_DEV_DETACHING	0039H	The specified file 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 these is true: • The user object does not have delete access to the file. • 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. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*
E_IO_MEM	0042H	The memory available to the BIOS is not sufficient to complete the call.

E_LIMIT	0004H	Processing this call would deplete the remote server's resources.
E_NAME_NEXIST	0049H	The user object does not represent a valid user, or the user object is not properly defined in the remote server's 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 user object password does not match the password of the user defined on the remote server.
E_PATHNAME_SYNTAX	003EH	The syntax of the specified remote file pathname is illegal; it must follow the naming conventions of the server.
E_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF.

delete_user

Deletes a user object.

Syntax, PL/M and C

```
CALL rq$delete$user (user, except_ptr);
```

```
rq_delete_user (user, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`user`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`user` A token for the user object to be deleted.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Deleting a user object has no effect on connections created with the user object.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The user parameter is not a token for an existing object.
E_LIMIT	0004H	Processing the call would exceed the limit of 255 outstanding I/O operations.
E_NOT_CONFIGURED	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.

encrypt

Encodes a STRING pointed to by the password_ptr parameter. There is no way to decrypt the encrypted STRING with any iRMX system call.

Syntax, PL/M and C

```
CALL rq$encrypt (password_ptr, key_ptr, encryption_ptr,
      except_ptr);
```

```
rq_encrypt (password_ptr, key_ptr, encryption_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
password_ptr	POINTER	STRING far *
key_ptr	POINTER	UINT_8 far *
encryption_ptr	POINTER	STRING far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

password_ptr

A pointer to an 8-character STRING containing the data to be encrypted.

key_ptr

A pointer to two ASCII characters that serve as an encryption key. These two characters become the second and third characters 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.

encryption_ptr

A pointer to a 15-character 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 character.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This system call is typically used to encrypt a password supplied by a user during logon or other system access verification. The `key_ptr` parameter enables 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.

See also: Data Encryption Standard (DES) algorithm, Federal Information Processing Standard Publication #46, January 15, 1977

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
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_CONFIGURED	0008H	This call is not part of the present configuration.

a_get_connection_status

Returns information about the connection status of any type of file.

Syntax, PL/M and C

```
CALL rq$a$get$connection$status (connection, resp_mbox,
      except_ptr);
```

```
rq_a_get_connection_status (connection, resp_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection

A token for the file connection whose status is to be returned.

resp_mbox

A token for the mailbox that is to receive a token for this segment. The calling task is responsible for deleting the segment after examining it.

```
DECLARE conn_status STRUCTURE(
   status                 WORD_16,
   file_driver            BYTE,
   flags                  BYTE,
   open_mode              BYTE,
   share_mode             BYTE,
   file_ptr               WORD_32,
   access                 BYTE);
```

or

```
typedef struct {
   UINT_16                status;
   UINT_8                 file_driver;
   UINT_8                 flags;
   UINT_8                 open_mode;
   UINT_8                 share_mode;
   UINT_32                file_ptr;
   UINT_8                 access;
} CONN_STATUS_STRUCT
```

Where:

`status` A condition code giving the outcome of the operation. If this code is not E_OK, consider the remaining fields invalid.

`file_driver`
Specifies the type of file driver to which this connection is attached.

Value	Type
1	Physical
2	Stream
3	DOS
4	Named
5	Remote File Driver (iRMX-NET)
6	EDOS
7-max	Loadable file drivers, including NFS. The ID for these drivers can vary; it is assigned in the order the driver is loaded.

`flags` Contains two flag bits that when set, indicate:

Bits	Meaning
7-3	Reserved, set to 0.
2	This is a device connection.
1	The connection is active and can be opened.
0	Reserved, set to 0.

`open_mode` The mode established when this connection was opened:

Value	Meaning
0	Connection is closed
1	Open for reading
2	Open for writing
3	Open for reading and writing

`share_mode`
The share mode established when this connection was opened:

Value	Meaning
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, 1 grants access and 0 denies it.

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

For remote iRMX-NET files, the access bits are interpreted as:

Bits	Data File	Directory File
7-4	Reserved	Reserved
3	Write	Ignored; set the same as bit 2
2	Write	Write; set the same as bit 3
1	Read	List
0	Ignored	Ignored

For NFS files, access bits can be mapped differently for different OSs.

See also: Accessing NFS files, Chapter 17, *System Concepts*

The DOS World user always has read (list) access to DOS files and directories; write access is optional.

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

When the status of a file connection to a virtual root directory is requested, list 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.

The BIOS does not check the access rights of an iRMX-NET remote file when you create a connection to the file, but checks during operations on the connection. This won't affect your programs if you follow these guidelines:

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

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	At least one of these is true: • One or more of the `connection` or `resp_mbox` parameters is not a token for an existing object. • The connection is being deleted. • The connection for a remote driver is no longer active.
E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its object limit. • 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_CONFIGURED	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	Either the connection parameter is not a token for a connection object, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
E_IO	002BH	An I/O error occurred, which might have prevented the operation from being completed. Examine the unit_status field of the IORS for more information. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*
E_NOT_FILE_CONN	0032H	For remote files, the connection parameter must be a file connection, not a device connection.

get_default_prefix

Returns the default prefix of a specified job.

Syntax, PL/M and C

```
connection = rq$get$default$prefix (job, except_ptr);

connection = rq_get_default_prefix (job, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection`	SELECTOR	SELECTOR
`job`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`connection`

A token for the connection object that is the default prefix for the designated job.

Parameters

`job` A token for the job whose default prefix is sought. A null selector specifies the calling task's job.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOPREFIX	8022H	A default prefix was requested, but cannot be found for one of these reasons: • When this job was created, a 0 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. • The `job` parameter is not a token for an existing object. • The prefix that is cataloged is of the wrong type. The default prefix must be a connection object or logical device object created by the EIOS. • The job parameter is not a job token.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

get_default_user

Returns the default user object of a specified job.

Syntax, PL/M and C

```
user_ID = rq$get$default$user (job, except_ptr);

user_ID = rq_get_default_user (job, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
user_ID	SELECTOR	SELECTOR
job	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

user_ID

A token for the user object that is the default user for the designated job.

Parameters

job — A token for the job whose default user object is sought. A null selector specifies the calling task's job.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOUSER	8021H	A default user cannot be found for one of these reasons: • When this job was created, a 0 was specified for its object directory, so the job cannot catalog a default user. • The job's directory can have entries but a default user is not cataloged there. • The object which is cataloged with the name *r?iouser* is not a user object. Treat *r?iouser* as a reserved word. • The `job` parameter is not a job token. • 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.

a_get_directory_entry

Returns the filename associated with an entry number in a named or DOS directory.

Syntax, PL/M and C

```
CALL rq$a$get$directory$entry (connection, entry_num,
      resp_mbox, except_ptr);
```

```
rq_a_get_directory_entry (connection, entry_num, resp_mbox,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
entry_num	WORD_16	UINT_16
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection

A token for a named or DOS directory only.

entry_num

The entry number of the desired filename. Entries within a directory are numbered sequentially starting from 0. The E_EMPTY_ENTRY condition code returns if there is no entry associated with this number.

resp_mbox

The mailbox that receives a token for this segment. The calling task is responsible for deleting this segment after examining it.

```
DECLARE dir_entry_info STRUCTURE(
   status                 WORD_16,
   name (14)              BYTE);
```

or

```
typedef struct {
   UINT_16                status;
   UINT_8                 name[14]
 } DIR_ENTRY_INFO_STRUCT;
```

Where:

`status` Indicates how the operation was completed. E_OK, E_EMPTY_ENTRY, and E_DIR_END condition codes all indicate successful completion.

`name` The filename contained in the specified entry. The filename is left-justified and padded with blanks to the right. This field is valid only if `status` is E_OK.

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

The caller must have list access to the designated directory. DOS World users always have read (list) access.

See also: **a_change_access**, **s_change_access**

As an alternative to using this system call, an application task can open and read a directory file.

The **a_get_directory_entry** system call is not supported for iRMX-NET remote directories. Use **a_open** and **a_read**, or **s_open** and **s_read_move**, to read remote directories.

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	At least one of these is true: • One or more of the `connection` or `resp_mbox` parameters is not a token for an existing object. • The connection is being deleted.
E_IFDR	002FH	At least one of these is true: • This system call applies only to named directories, but the `connection` parameter specifies another type of file. • 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_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	Either the connection parameter is not a token for a connection object, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
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 does not have list 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.

a_get_extension_data

Returns extension data stored with a BIOS named data or directory file. This call is not valid for DOS files or for files accessed through NFS.

Syntax, PL/M and C

```
CALL rq$a$get$extension$data (connection, resp_mbox,
      except_ptr);
```

```
rq_a_get_extension_data (connection, resp_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection

A token for a file connection whose extension data is to returned.

resp_mbox

The mailbox that receives a token for this segment. The calling task is responsible for deleting this segment after examining it.

```
DECLARE file_status STRUCTURE (
   status                 WORD_16,
   count                  BYTE,
   info(*)                BYTE);
```

or

```
typedef struct {
   UINT_16                status;
   UINT_8                 count;
   UINT_8                 info[_NUM_FILE_INFO];
                                    /* adjust to fit count */
 } FILE_STATUS_STRUCT;
```

Where:

`status`	A condition code indicating the outcome of the operation. If this code is not E_OK, consider the remaining fields invalid.
`count`	The number from 0 to 255 of bytes returned; set to 0 for remote files.
`info`	The extension data.

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

A_get_extension_data can only be applied to connections created using the named file driver.

A file descriptor is associated with each file created through the BIOS. Some of the information in the descriptor is used by the BIOS and can be accessed through **a_get_file_status**. Up to 255 additional bytes of the file descriptor, known as extension data, are available for use by OS extensions.

The first three bytes of extension data are reserved for use by the BIOS. OS extensions can write extension data by using **a_set_extension_data** and they can read extension data by using **a_get_extension_data**.

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	At least one of these is true: • One or more of the `connection` or `resp_mbox` parameters is not a token for an existing object. • The connection is being deleted. • 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 another type of file.

E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its object limit. • 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_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	Either the connection parameter is not a token for a connection object, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
E_IO	002BH	An I/O error occurred which might have prevented the operation from completing.

get_file_driver_status

Returns information on a specified file driver. Use this call to build a table of all available file drivers (resident and loadable) currently available in the system.

Syntax, PL/M and C

```
CALL rq$get$file$driver$status (file_driver, ret_data_ptr,
      except_ptr);
```

```
rq_get_file_driver_status (file_driver, ret_data_ptr,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
file_driver	BYTE	UINT_8
ret_data_ptr	POINTER	FD_STATUS_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

file_driver

Specifies the file driver ID. These are some of the typical file driver IDs:

ID	Description
0	Reserved; not a valid file driver ID
1	Physical file driver, always present
2	Stream file driver, always present
3	Native DOS file driver, ICU-configurable or loadable
4	Named file driver, ICU-configurable or loadable
5	Remote file driver, ICU-configurable or loadable
6	EDOS file driver, ICU-configurable or loadable
7-max	Loadable file drivers, including NFS. The ID for these drivers can vary; it is assigned in the order the driver is loaded. The maximum number is ICU-configurable.

`ret_data_ptr`

A pointer to this structure:

```
DECLARE fd_status_struct STRUCTURE(
   max_file_drivers        BYTE,
   num_file_drivers        BYTE
   flags                   BYTE,
   buffer_size             WORD_16,
   filesystem              BYTE,
   io_task_priority        BYTE,
   name_length             BYTE,
   name(14)                BYTE,
   reserved(19)            BYTE);
```

or

```
typedef struct {
   UINT_8                  max_file_drivers;
   UINT_8                  num_file_drivers;
   UINT_16                 flags;
   UINT_16                 buffer_size;
   UINT_8                  file_system;
   UINT_8                  io_task_priority;
   UINT_8                  name_length;
   UINT_8                  name[14];
   UINT_8                  reserved[19];
} FD_STATUS_STRUCT;
```

Where:

`max_file_drivers`

Largest possible file driver ID value. The number of loadable file drivers is ICU-configurable (default = 16).

`num_file_drivers`

Number of file drivers currently available in the system, including loadable and resident drivers.

flags Encoded as:

Bit(s)	Meaning
15	File driver present at this ID; all other fields in the structure are invalid unless this bit is set.
14	1 = loaded file driver 0 = resident file driver
3-13	Reserved, set to 0
2	Convert filenames to lower case
1	DUIBs required
0	User object required

buffer_size

Default size for EIOS buffers.

file_system

File system supported by this file driver (only meaningful if bit 1 of the flags field is set). Indicates:

Bit(s)	File System Type
6-7	Reserved, set to 0
5	EDOS
4	Remote (including Remote File Driver and NFS)
3	iRMX Named (or other hierarchical)
2	DOS
1	Stream
0	Physical

io_task_priority

Default priority for I/O tasks associated with this file driver. Should normally be 0 (uses task priority field in the DUIB as default).

name_length

Actual length of the name field (excluding blanks).

name Unique file driver identifier of up to 14 bytes (padded with blanks).

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

To build a table of all available file drivers, first make this call with a file_driver number of 1 to obtain the value of max_file_drivers from FD_STATUS_STRUCT. Then, loop max_file_drivers times to obtain information on each file driver. A file driver is present at a given file driver ID if bit 15 of the flags field is set.

Condition Codes

E_PARAM	8004H	One of these is true: • The file driver ID is 0 or larger than the maximum allowable value. • The structure referenced by ret_data_ptr is not writable or large enough to hold the return data.

a_get_file_status

Returns device-dependent status and attribute information about a specified file of any type. Additional information returns for named files.

Syntax, PL/M and C

```
CALL rq$a$get$file$status (connection, resp_mbox, except_ptr);
```

```
rq_a_get_file_status (connection, resp_mbox, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`connection`	SELECTOR	SELECTOR
`resp_mbox`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection`

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

`resp_mbox`

The mailbox that receives a token for this segment. The information returned depends on the file type specified. For all types of files, the first part of this structure through the `dev_conn` field returns. If the contents of the `named_file` field indicate a named, remote, or DOS file, the second part (from `file_ID` on) returns also.

```
DECLARE file_info STRUCTURE(
   status                 WORD_16,
   num_conn               WORD_16,
   num_reader             WORD_16,
   num_writer             WORD_16,
   share                  BYTE,
   named_file             BYTE,
   dev_name(14)           BYTE,
   file_drivers           WORD_16,
   functs                 BYTE,
   flags                  BYTE,
   dev_gran               WORD_16,
   dev_size               WORD_32,
   dev_conn               WORD_16,
   file_ID                WORD_16,
   file_type              BYTE,
   file_gran              BYTE,
   owner_ID               WORD_16,
   create_time            WORD_32,
   access_time            WORD_32,
   modify_time            WORD_32,
   file_size              WORD_32,
   file_blocks            WORD_32,
   vol_name(6)            BYTE,
   vol_gran               WORD_16,
   vol_size               WORD_32,
   accessor_count         WORD_16,
   first_access           BYTE,
   first_ID               WORD_16,
   second_access          BYTE,
   second_ID              WORD_16,
   third_access           BYTE,
   third_ID               WORD_16,
   vol_flags              BYTE);
```

or

```
typedef struct {
   UINT_16                 status;
   UINT_16                 num_conn;
   UINT_16                 num_reader;
   UINT_16                 num_writer;
   UINT_8                  share;
   UINT_8                  named_file;
   UINT_8                  dev_name[14];
   UINT_16                 file_drivers;
   UINT_8                  functs;
   UINT_8                  flags;
   UINT_16                 dev_gran;
   UINT_32                 dev_size;
   UINT_16                 dev_conn;
   UINT_16                 file_ID;
   UINT_8                  file_type;
   UINT_8                  file_gran;
   UINT_16                 owner_ID;
   UINT_32                 create_time;
   UINT_32                 access_time;
   UINT_32                 modify_time;
   UINT_32                 file_size;
   UINT_32                 file_blocks;
   UINT_8                  vol_name[6];
   UINT_16                 vol_gran;
   UINT_32                 vol_size;
   UINT_16                 accessor_count;
   UINT_8                  first_access;
   UINT_16                 first_ID;
   UINT_8                  second_access;
   UINT_16                 second_ID;
   UINT_8                  third_access;
   UINT_16                 third_ID;
   UINT_8                  vol_flags;
} FILE_INFO_STRUCT;
```

Where:

`status` Indicates how the **get_file_status** operation was completed. If this condition code is not E_OK, consider the remaining fields invalid.

`num_conn` The number of connections to the file. For remote and NFS files, this field indicates the number of connections between the calling job and the file.

`num_reader`
The number of connections currently open for reading. For remote and NFS files a 0 indicates either no connection or a connection open for writing only, and a 1 indicates an open readable or read/writable connection.

`num_writer`
The number of connections currently open for writing. For remote and NFS files a 0 indicates either no connection or a connection open for reading only, and a 1 indicates an open writable or read/writable connection.

`share` Indicates the current share mode of the file.

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

If a remote or NFS file is open, the share mode used to open the connection is returned, but if the file connection is not open, share mode 3 is indicated.

`named_file`
Tells whether this structure contains any information beyond the `dev_conn` field.

Value	**Meaning**
0	No
0FFH	Yes, fields beyond `dev_conn` are valid

`dev_name` The name of the physical device where this file resides. This name is left-justified and padded with blanks to the right.

For remote files, this is the name of the remote server on which the file resides. For NFS files, this is the host name and path used when the device was attached.

`file_drivers`

Indicates what kinds of files can reside on this device.

File Type Bit	File Type
7-6	Reserved
5	EDOS file
4	Remote (iRMX-NET) or NFS file
3	Named file
2	DOS
1	Stream file
0	Physical file

`functs` Describes the functions supported by the device where this file resides. A bit set to 1 indicates the corresponding function is supported.

Bit	Function
7	FCLOSE
6	FOPEN
5	FDETACHDEV
4	FATTACHDEV
3	FSPECIAL
2	FSEEK
1	FWRITE
0	FREAD

This field is not supported by the iRMX-NET remote file driver; 0 returns for remote files.

`flags` Meaningful only for diskette drives. This field is not supported by iRMX-NET or the NFS file driver; 0 returns for such remote files.

Bits	Value	Function
7-5		Reserved; set to 0
4	0	Standard diskette, for MBI only; track 0 is single-density, 128-byte sectors
	1	Uniform diskette or not a diskette
3	0	High (quad) density
	1	Low (double) density For 8" diskettes, set to 0
2	0	Single sided
	1	Double sided

Bits	**Value**	**Function**		
1	0	Single density		
	1	Not single density		
		Disk Size	**Bit 1**	**Bit 3**
		3.5D	1	1
		3.5Q	1	0
		5.25D	1	1
		5.25Q	1	0
		8S	0	0
		8D	1	0
0	0	This field is undefined		
	1	Bits 7-1 are valid		

See also: Supporting the standard diskette format, *Driver Programming Concepts*

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

`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.

See also: **diskverify**, *Command Reference*

`file_type` Indicates the type of the file:

Value	**Meaning**
6	Directory file
8	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, the file's granularity is 512. For remote, NFS, and DOS files, 1 is returned.

`owner_ID` The first ID in the user object that was specified when the file was created.

`create_time, access_time, modify_time`

The date and time when the file was created, last accessed, or last modified. The date/time value is the number of seconds since midnight, January 1, 1978. For ICU-configurable systems, an ICU option determines whether the OS maintains these fields.

See also: Timing facilities required, *ICU User's Guide and Quick Reference*

`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 is always one for DOS files. For iRMX files, this list may have been added to after the file was created; you get the current value.

`first_access, second_access, third_access`

Access masks for as many IDs as are indicated by `accessor_count`. The only DOS accessor is World. DOS access options are limited to either read-only or full access. The bits of the access masks are defined in this table. For each bit, 1 grants access. Access rights for NFS files may be mapped differently for different OSs.

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

`first_ID, second_ID, third_ID`

ID values for the accessors. User IDs for NFS files may be mapped differently for different OSs.

See also: Accessing NFS files, Chapter 17, *System Concepts*

vol_flags The vf_integrity flag:

Bits	Value	Meaning
7-1		Reserved.
0	0	Volume properly shut down.
	1	Possible disk corruption (volume was attached but was not subsequently shut down).

except_ptr

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	At least one of these is true: • The resp_mbox parameter is not a mailbox token. • The connection is being deleted. • The connection for a remote driver is no longer active.
E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its object limit. • 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_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	One or more of the connection or resp_mbox parameters is a token for an object of the wrong type.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
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 UDF.

get_global_time

Reads the time of day from the battery-backed-up hardware clock.

Syntax, PL/M and C

```
CALL rq$get$global$time (date_time_ptr, except_ptr);
```

```
rq_get_global_time (date_time_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
date_time_ptr	POINTER	SET_TIME_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

date_time_ptr

A pointer to this structure:

```
DECLARE set_time STRUCTURE (
   seconds               BYTE,
   minutes               BYTE,
   hours                 BYTE,
   days                  BYTE,
   months                BYTE,
   years                 WORD_16);
```

or

```
typedef struct {
   UINT_8                seconds;
   UINT_8                minutes;
   UINT_8                hours;
   UINT_8                days;
   UINT_8                months;
   UINT_16               years;
} SET_TIME_STRUCT;
```

Where:

`seconds`	The current value of the seconds count.
`minutes`	The current value of the minutes count.
`hours`	The current value of the hours count.
`days`	The current value of the days count.
`months`	The current value of the month count.
`years`	The current value of the year count.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The BIOS accesses the appropriate registers on the hardware clock to read the global date and time values.

This system call supports the Time-of-Day Clock on the Multibus I SBC 546 and 549 Terminal Communications Controller boards, the Multibus II CSM, the Multibus I SBC 86C38 board, and PC Bus Systems.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	This call was made from an environment that did not contain a hardware clock.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SHARE	0028H	The hardware clock was busy because another task was accessing it.
E_SUPPORT	0023H	The clock type is not supported.

a_get_path_component

Returns the name of a data or directory file, as cataloged in its parent directory.

Syntax, PL/M and C

```
CALL rq$a$get$path$component (connection, resp_mbox,
      except_ptr);
```

```
rq_a_get_path_component (connection, resp_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection

A token for the file connection whose name is sought.

resp_mbox

The mailbox that receives a token for this segment. The calling task is responsible for deleting this segment after examining it.

```
DECLARE filename STRUCTURE(
   status                 WORD_16,
   name                   STRING);
```

or

```
typedef struct {
   UINT_16                status;
   STRING                 name[14];
} FILENAME_STRUCT;
```

Where:

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

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

except_ptr

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

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. A null string returns if:

- The connection is to the root directory of a volume
- The file is marked for deletion or is a temporary file
- A connection to a physical or stream file is specified

A_get_path_component can be used in combination with **a_attach_file** to derive all of the components of a pathname. Suppose, for example, that a file has the path name *A/B/C*, and that your task has only a token for the file. This 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.

See also: **a_attach_file**

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	At least one of these is true: • One or more of the `connection` or `resp_mbox` parameters is not a token for an existing object. • 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_CONFIGURED	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	Either the connection parameter is not a token for a connection object, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
E_FNEXIST	0021H	The file is marked for deletion, so the name string is undefined.
E_INVALID_FNODE	003DH	The fnode for the specified file is invalid. The file cannot be accessed; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*
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 BIOS job is not sufficient to complete the call.

inspect_user

Accepts a token for a user object and returns a list of the IDs contained in the user object.

Syntax, PL/M and C

```
CALL rq$inspect$user (user, IDs_ptr, except_ptr);
```

```
rq_inspect_user (user, IDs_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
user	SELECTOR	SELECTOR
IDs_ptr	POINTER	IDS_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

user A token for the user object being inspected.

IDs_ptr

A pointer to this structure:

```
DECLARE IDs STRUCTURE(
   length                 WORD_16,
   count                  WORD_16,
   IDs(*)                 WORD_16);
```

or

```
typedef struct {
   UINT_16                length;
   UINT_16                count;
   UINT_16                ids[2]; /* adjust to fit count */
 } IDS_STRUCT;
```

Where:

length The upper limit on the number of IDs to return. The calling task must supply this value; 0 values are not permitted.

count Actual number of IDs that the BIOS returns.

IDs The IDs the BIOS returns.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If the `length` value is smaller than the actual number of IDs in the user object, only the specified number of IDs returns.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The user parameter is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The length field contains a 0 value.
E_TYPE	8002H	The user parameter is a token for an object of the wrong type.

rq_install_duibs

Installs a cluster of Device Unit Information Blocks (DUIBs) for loadable device drivers into the BIOS. These DUIBs, and the physical devices they represent, can then be attached with the **a_physical_attach_device** system call. Use this system call for device drivers you write.

Syntax, PL/M and C

```
CALL rq$install$duibs (num_duibs, duibs_ptr, aux_ptr,
      except_ptr);
```

```
rq_install_duibs (num_duibs, duibs_ptr, aux_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
num_duibs	WORD_16	UINT_16
duibs_ptr	POINTER	DUIB_TABLE_STRUCT far *
aux_ptr	POINTER	void far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

num_duibs

The number of DUIBs pointed to by duibs_ptr.

duibs_ptr

A pointer to a cluster of DUIBs to be installed into the BIOS.

```
DECLARE DUIB_TABLE_STRUCT STRUCTURE(
   duibs (_NUM_DUIBS)     DUIB_STRUCT);
```

or

```
typedef struct {
   DUIB_STRUCT            duibs [_NUM_DUIBS];
} DUIB_TABLE_STRUCT
```

```
DECLARE DUIB_STRUCT STRUCTURE(
   name (14)              BYTE,
   file_drivers           WORD_16,
   functs                 BYTE,
   flags                  BYTE,
   dev_gran               WORD_16,
   dev_size               WORD_32,
   device                 BYTE,
   unit                   BYTE,
   dev_unit               WORD_16,
   init_io                WORD_32,
   finish_io              WORD_32,
   queue_io               WORD_32,
   cancel_io              WORD_32,
   device_info_ptr        POINTER,
   unit_info_ptr          POINTER,
   update_timeout         WORD_16,
   num_buffers            WORD_16,
   priority               BYTE,
   fixed_update           BYTE,
   max_buffers            BYTE,
   reserved               BYTE);
```

or

```
typedef struct {
   UINT_8                 name [14];
   UINT_16                file_drivers;
   UINT_8                 functs;
   UINT_8                 flags;
   UINT_16                dev_gran;
   UINT_32                dev_size;
   UINT_8                 device;
   UINT_8                 unit;
   UINT_16                dev_unit;
   UINT_32                init_io;
   UINT_32                finish_io;
   UINT_32                queue_io;
   UINT_32                cancel_io;
   void far *             device_info_p;
   void far *             unit_info_p;
   UINT_16                update_timeout;
   UINT_16                num_buffers;
   UINT_8                 priority;
   UINT_8                 fixed_update;
   UINT_8                 max_buffers;
   UINT_8                 reserved;
} DUIB_STRUCT;
```

Where:

`name` The DUIB name. This name uniquely identifies the device-unit to the I/O System. Use only the first 13 bytes. The fourteenth is used by the I/O System. Names with less than 14 characters are extended with spaces.

The name is assigned as part of the driver configuration process. You specify the DUIB name when attaching a unit using the **a_physical_attach_device** system call. Device drivers do not read or write this field.

`file_drivers`

Specifies which file driver(s) can attach this device-unit:

Bit	**Driver No.**	**Driver**
5	6	EDOS
4	5	Remote
3	4	Named
2	3	DOS
1	2	Stream
0	1	Physical

`functs` Specifies the valid I/O function(s) for this device-unit:

Bit	Function
7	close
6	open
5	detach device (always set)
4	attach device (always set)
3	special
2	seek
1	write
0	read

To provide accurate status information, this field should indicate the device's ability to perform the I/O functions. Each device driver must be able to either perform the function or return a condition code indicating the inability to perform that function. Device drivers do not read or write this field.

`flags` This field does not apply to PC-AT ROM BIOS-based diskette driver. Specifies characteristics of diskette devices:

Bits	Value	Meaning
7-5	0	Reserved; set to 0.
4	0	Standard diskette, for MB I only
	1	Uniform diskette or not a diskette
3	0	Quad density
	1	Double density For 8 inch diskettes, set to 0
2	0	Single-sided
	1	Double-sided
1	0	Single density
	1	Not single density

Disk Size	Bit 1	Bit 3
3.5D	1	1t
3.5Q	1	0
5.25D	1	1
5.25Q	1	0
8S	0	0
8D	1	0

Bits	Value	Meaning
0	0	This field is undefined
	1	Bits 7-1 are valid

`dev_gran` Specifies the device granularity in bytes. This field applies to random access devices, and to some common devices such as tape drives. It specifies the minimum number of bytes of information the device reads or writes in one operation. If the device is a disk or tape drive, set to the sector size for the device. Otherwise, set to `0`.

`dev_size` Specifies the number of bytes of information the device-unit can store.

`device` Specifies the device number of the device with which this device-unit is associated. Device drivers do not access this field.

`unit` The unit number of this device-unit. This distinguishes the unit from the other units of the device.

`dev_unit` The device-unit number. This number distinguishes the device-unit from the other units in the entire hardware system. Device drivers can ignore this field.

`init_io` Specifies the offset address of the init_io procedure associated with this unit (the base portion is the driver code segment). Custom device drivers must supply this procedure and the finish_io, queue_io, and cancel_io procedures. For common, random access, and terminal drivers, the procedures are supplied with the I/O System. For loadable device drivers, this field specifies the driver type. Device drivers do not access this field.

`finish_io` Specifies the offset address of the finish_io procedure associated with this unit (the base portion is the driver code segment). Device drivers do not access this field. For loadable drivers, this field specifies the driver type.

`queue_io` Specifies the offset address of the queue_io procedure associated with this unit (the base portion is the driver code segment). Device drivers do not access this field. For loadable drivers, this field specifies the driver type.

`cancel_io` Specifies the offset address of the cancel_io procedure associated with this unit (the base portion is the driver code segment). Device drivers do not access this field. For loadable drivers, this field specifies the driver type.

`device_info_ptr`

Pointer to a structure containing additional information about the device: the DINFO table. Each common, random access, and terminal device driver requires a DINFO table in a particular format.

When writing a custom driver, you can place information in the DINFO table according to the needs of the driver. Specify a `0` for this parameter if the associated device driver does not use this field.

For flat model applications only, treat this parameter as two separate fields in the structure. The first field has the name listed above and is a near pointer. The second field has the same name with `_seg` appended at the end. It is a segment selector for the pointer.

`unit_info_ptr`

Pointer to a structure containing more information about the unit: the UINFO table. Random access and terminal device drivers require a UINFO table in a particular format.

When writing a custom device driver, place information in this structure according to the needs of the driver. Specify a `0` if the associated device driver does not use this field.

For flat model applications only, treat this parameter as two separate fields in the structure. The first field has the name listed above and is a near pointer. The second field has the same name with `_seg` appended at the end. It is a segment selector for the pointer.

`update_timeout`

Specifies the number of system clock ticks the I/O System must wait before writing a partial sector after processing a write request for a disk device. Except for disk device drivers, set to 0FFFFH. This field applies only to the device-unit specified by this DUIB; the field is independent of updating done either because of the value in the `fixed_update` field of the DUIB or the **a_update** system call. Device drivers do not access this field.

`num_buffers`

A `0` indicates the device is not a random access device. Otherwise, the number of buffers of `dev_gran` size that the I/O System allocates. The I/O System uses the buffers for data blocking and deblocking, so that data is read or written beginning on sector boundaries. The random access high-level device driver procedures guarantee that no data is written or read across track boundaries in a single request. Device drivers do not access this field.

`priority` Specifies the priority of the I/O System service task for the device. Device drivers do not access this field.

`fixed_update`

TRUE indicates that the fixed update option was selected for the device-unit when the driver was configured, FALSE indicates otherwise. This option causes the I/O System to finish any write requests that had not been finished earlier because less than a full sector remained to be written. Fixed updates are performed throughout the entire system whenever a time interval (specified during configuration) elapses. This is independent of the updating indicated for a particular device by the `update_timeout` field of the DUIB or the updating of a particular device indicated by the **a_update** system call of the I/O System. Device drivers do not access this field.

`max_buffers`

Specifies the maximum number of buffers the EIOS can allocate for a connection to this device-unit when the connection is opened by a call to **s_open**. The value in this field is specified during driver configuration. Device drivers do not access this field.

See also: DUIBs, *Driver Programming Concepts*

`aux_ptr`

Reserved. Set to null.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The maximum number of clusters that can exist in the system is a configuration option.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_LIMIT	0004H	The maximum number of clusters in the system has been reached.

install_file_driver

Installs a loadable file driver into the BIOS.

Syntax, PL/M and C

```
file_driver = rq$install$file$driver (data_ptr, config_ptr,
      ret_info_ptr, except_ptr);
```

```
file_driver = rq_install_file_driver (data_ptr, config_ptr,
      ret_info_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`file_driver`	BYTE	UINT_8
`data_ptr`	POINTER	LOADABLE_FD_DATA_STRUCT far *
`config_ptr`	POINTER	LOADABLE_FD_CONFIG_STRUCT far *
`ret_info_ptr`	POINTER	LOADABLE_FD_INFO_STRUCT far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`file_driver`

The file driver ID for the loaded driver. These are the possible values:

ID	Description
0	Reserved; not a valid file driver ID
1	Physical file driver (non-loadable)
2	Stream file driver (non-loadable)
3	Native DOS file driver
4	Named file driver
5	Remote file driver
6	EDOS file driver
7-max	Available for loadable file drivers; the maximum value is ICU-configurable

Parameters

`data_ptr`

If not null, a pointer to this structure. A null pointer uninstalls the file driver (see Additional Information).

```
DECLARE loadable_fd_data_struct STRUCTURE(
   conn_entries           WORD_16,
   att_dev_stack_size     WORD_16,
   dev_desc_size          WORD_16,
   xface_mbox             SELECTOR,
   flags                  WORD_16,
   buffer_size            WORD_16,
   filesystem             BYTE,
   io_task_prio           BYTE,
   name_length            BYTE,
   name(14)               BYTE,
   reserved(19)           BYTE);
```

or

```
typedef struct {
   UINT_16                conn_entries;
   UINT_16                att_dev_stack_size;
   UINT_16                dev_desc_size;
   SELECTOR               xface_mbox;
   UINT_16                flags;
   UINT_16                buffer_size;
   UINT_8                 file_system;
   UINT_8                 io_task_prio;
   UINT_8                 name_length;
   UINT_8                 name[14];
   UINT_8                 reserved[19];
} LOADABLE_FD_DATA_STRUCT
```

Where:

`conn_entries`

Specifies the size of the connection object for this file driver.

`att_dev_stack_size`

Specifies the size of the **attach_device** task's stack.

`dev_desc_size`

Specifies the size of the device descriptor for devices attached to this file driver.

`xface_mbox`

A token for a mailbox to be used if the default device attach task is not used for this file driver. If 0, the standard **attach_device** task and its mailbox are used.

`flags` Control bits defined as:

Bit(s)	Meaning
0	User object required
1	DUIBs required
2	Convert filenames to lower case
3-5	Reserved, set to 0

`buffer_size`

Default buffer size for EIOS buffers.

`file_system`

Defines the type of file system supported by this file driver, specifying the DUIBs that can be used with this file driver (only meaningful if bit 1 is set in the flags field). Encoded as:

Bit(s)	File System Type
0	Physical
1	Stream
2	DOS
3	iRMX Named (or other hierarchical)
4	Remote
5	EDOS
6-7	Reserved, set to 0

`io_task_priority`

Default priority for I/O tasks associated with this file driver. If not 0, this field overrides the task priority field in the DUIBs. Should normally be 0.

`name_length`

Actual length of the name field (excluding blanks).

`name` Unique file driver identifier of up to 4 bytes (padded with blanks).

`config_ptr`

A pointer to this structure:

```
DECLARE loadable_fd_config_struct STRUCTURE(
   initialize             POINTER,
   io_task                POINTER,
   update                 POINTER,
   attach_funct(4)        POINTER,
   io_funct(21)           POINTER,
   valid_request(21)      BYTE);
```

or

```
typedef struct {
   void far *            initialize;
   void far *            iotask;
   void far *            update;
   void far *            attach_funct[4];
   void far *            io_funct[21];
   UINT_8                valid_request[21];
} LOADABLE_FD_CONFIG_STRUCT;
```

Where:

`initialize`
A pointer to the file driver initialization procedure. A null pointer indicates that no initialization is required.

`io_task` A pointer to the I/O task used with the file driver. A null pointer specifies the BIOS common I/O task.

`update` A pointer to the file driver update procedure.

`attach_funct`
An array of pointers to the 4 file driver attach functions.

`io_funct` An array of pointers to the 21 file driver I/O interfaces.

`valid_request`
Each byte specifies whether the corresponding driver I/O interface is valid for this file driver.

See also: *Driver Programming Concepts* for more information on these elements

Note

For flat model applications only, treat the `initialize`, `io_task`, and `update` parameters as two separate fields each in the structure. The first field has the name listed above and is a near pointer. The second field has the same name with `_seg` appended at the end. It is a segment selector for the pointer.

`ret_info_ptr`
A pointer to this structure. It provides access to several BIOS objects and procedures which may be required for correct file driver operation. To use the objects within this structure, copy them all into global variables of the same name.

```
DECLARE loadable_fd_info_struct STRUCTURE(
   conn_region           SELECTOR,
   conn_ext              SELECTOR,
   detach_device         POINTER,
   cancel_dev_io         POINTER,
   device_io             POINTER);
```

or

```
typedef struct {
   SELECTOR              conn_region;
   SELECTOR              conn_ext;
   void far *            detach_device;
   void far *            cancel_dev_io;
   void far *            device_io;
} LOADABLE_FD_INFO_STRUCT
```

Where:

`conn_region`
A token for the global BIOS connection region. This region is used for mutual exclusion around all connection management operations.

`conn_ext` A token for the global BIOS connection extension object.

`detach_device`
A pointer to the common BIOS detach_device procedure. This procedure must be called from the file driver's detach_device procedure when a device is physically detached.

`cancel_dev_io`
A pointer to the common BIOS cancel_io procedure. This procedure calls the device driver's cancel_io procedure.

`device_io`
A pointer to the common BIOS device_io procedure. This procedure calls the device driver queue_io procedure. It should be called to perform all I/O from the file driver.

Note

For flat model applications only, treat the `detach_device`, `cancel_dev_io`, and `device_io` parameters as two separate fields each in the structure. The first field has the name listed above and is a near pointer. The second field has the same name with `_seg` appended at the end. It is a segment selector for the pointer.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If the `ret_info_ptr` is a null pointer, the file driver is installed, but no file driver information is returned.

A file driver can only be installed once, even with a different file driver ID. This is enforced by comparing the file driver name with all other file driver names in the system.

See also: *Driver Programming Concepts* for more information on file drivers

Condition Codes

E_FEXIST	0006H	A file driver with the same ASCII name has already been installed in the system.
E_PARAM	8004H	One of these conditions is true: • The file driver ID is 0 or larger than the maximum allowable value. • The structure referenced by `config_ptr` is not readable. • The structure referenced by `config_ptr` is not large enough. • The structure referenced by `data_ptr` is not readable (if not a null pointer). • The structure referenced by `data_ptr` is not large enough. • The structure referenced by `ret_info_ptr` is not writable (if not a null pointer). • The structure referenced by ret_info_ptr is not large enough.

a_open

Opens an asynchronous file connection for I/O operations, for any type of file.

Syntax, PL/M and C

```
CALL rq$a$open (connection, mode, share, resp_mbox,
      except_ptr);
```

```
rq_a_open (connection, mode, share, resp_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection`	SELECTOR	SELECTOR
`mode`	BYTE	UINT_8
`share`	BYTE	UINT_8
`resp_mbox`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection`

A token for the connection to be opened.

`mode` The mode desired for the open connection; set to 1 to open directories.

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

`share` Specifies the share mode for the file to which you are opening a connection:

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

`resp_mbox`

The mailbox that receives a token for an IORS. A null selector means that you do not want to receive an IORS.

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

The connection must be opened before reading, writing, and seeking can be performed on the associated file.

Directory files can be opened and read, but only by specifying a 1 (read) for the mode parameter and a 3 (share all) for the share parameter. Any other combination will return an error.

A_open also initializes the file pointer to byte-position 0. Subsequent BIOS calls, such as **a_seek**, **a_read**, and **a_write**, will move this pointer.

The `mode` and `share` parameters are compared to the current share mode of the file, which may have been set by a previous **a_open** system call. If they are not compatible, an E_SHARE condition code returns. No deadlock occurs, however, because open calls are not queued. The system does not automatically notify callers when the share mode of the file changes.

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.

See also: **a_seek**, **a_read**, **a_write**

The BIOS does not check the access rights of an iRMX-NET remote file when you create a connection to the file, but does check during operations on the connection. This won't affect your programs if you:

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

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	At least one of these is true: • One or more of the `connection` or `resp_mbox` parameters is not a token for an existing object. • The connection is being deleted. • The connection for a remote driver is no longer active.

E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its object limit. • 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_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The mode or share parameter is outside the range 1-3, or 0-3 respectively.
E_SUPPORT	0023H	The specified connection was not created by this job.
E_TYPE	8002H	Either the connection parameter is not a token for a connection object, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
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 these is true: • The file's current share mode is not compatible with the mode or the share parameter. • This call is attempting to open a directory for some operation other than read or share with all users.
E_FACCESS	0026H	The connection does not have access compatible with the mode specified.

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. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*
E_LIMIT	0004H	Processing this call would deplete the remote server's resources.
E_FTYPE	0027H	The requested operation is not valid for this file type.

a_physical_attach_device

Attaches the specified device to the BIOS.

⚠ **CAUTION**

Any task that uses this call loses its device independence. When the containing job is deleted, any attached devices are automatically detached, and connections to files on the device are automatically deleted. To prevent this, use the EIOS call **logical_attach_device**.

Syntax, PL/M and C

```
CALL rq$a$physical$attach$device (dev_name_ptr, file_driver,
      resp_mbox, except_ptr);
```

```
rq_a_physical_attach_device (dev_name_ptr, file_driver,
      resp_mbox, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`dev_name_ptr`	POINTER	STRING far *
`file_driver`	BYTE	UINT_8
`resp_mbox`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`dev_name_ptr`

A pointer to a STRING containing the logical name of the remote disk, the physical device name. The maximum length is 14 characters. For all file types except NFS (NFS supports extended device names of up to 256 characters), the BIOS truncates the name to 14 characters if it is longer . To prevent possible duplication of names, do not use device names longer than 14 characters. For devices accessed through the Remote File Driver, specify the name of the server to be attached. For NFS devices, specify the name as *host*:/*shared_directory*.

See also: For ICU-configurable systems, DEV parameter, *ICU User's Guide and Quick Reference* `file_driver`

Specifies the kind of files that the device will create when the returned device connection is used in subsequent calls to **a_create_file**.

Value	File Driver
1	Physical
2	Stream
3	DOS
4	Named
5	Remote
6	EDOS
7-max	Loadable file drivers, including NFS. The IDs can vary, depending on which driver is loaded first. To find what ID is currently assigned to a specific loadable driver, first call **rq_get_file_driver_status**.

`resp_mbox`

The mailbox that receives a token for a new connection if the call succeed, otherwise an IORS. 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**. To determine the type of object returned, use the Nucleus system call **get_type**.

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

Only a few selected tasks should perform all device attaching and detaching, passing tokens for the devices to other tasks as necessary.

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. Otherwise, an E_ILLVOL condition code returns.

See also: **a_create_file**,
EIOS call **logical_attach_device**,
Nucleus call **get_type**,
Formatting disks, *Command Reference*

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The resp_mbox parameter is not a token for an existing object.
E_LIMIT	0004H	Processing this call would exceed one or more of these limits: • The object limit for this job • 255 outstanding I/O operations 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_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The number representing the file driver is not valid or a null selector was specified for the response mailbox.
E_TYPE	8002H	The resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_ALREADY_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 specified device does not exist.
E_ILLVOL	002DH	At least one of these is true: • The specified disk volume is not properly formatted for use with the named file driver. • 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. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*
E_IO_MEM	0042H	The memory available to the BIOS job is not sufficient to complete the call.
E_LIMIT	0004H	Processing this call would deplete the remote server's resources.
E_PROTOCOL	02E9H	The iNA 960 version on the local system does not have the iNA R.0 to R3.0 compatibility code and the server to be attached has iNA R.0 loaded.

a_physical_detach_device

Detaches a device that was attached using **a_physical_attach_device**.

Syntax, PL/M and C

```
CALL rq$a$physical$detach$device (connection, hard, resp_mbox,
      except_ptr);
```

```
rq_a_physical_detach_device (connection, hard, resp_mbox,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection`	SELECTOR	SELECTOR
`hard`	BYTE	UINT_8
`resp_mbox`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection`
A token for the connection object for the device that is to be detached.

`hard` Specifies whether or not you want a hard detach of the device.

Value	Meaning
0	No
0FFH	Yes

`resp_mbox`
A token for the mailbox that receives an IORS. A null selector means that you do not want to receive an IORS.

`except_ptr`
A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

This call deletes the file connection objects associated with the device connections. A device that is detached with this call must be reattached before any files can be attached or reattached to the device.

A hard detach automatically deletes the connection objects for all files attached to the device. If you do not specify a hard detach, first detach all files from the device using **a_delete_connection**; otherwise, the condition code E_OUTSTANDING_CONNS returns.

Whether you specify a hard detach or not, there will be no attached files on the device after using **a_physical_detach_device**.

See also: **a_physical_attach_device**, **a_delete_connection**

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	One or more of the connection or resp_mbox parameters is not a token for an existing object.
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_NOT_DEVICE_CONN	0033H	The specified connection is not a device connection.
E_SUPPORT	0023H	The specified connection was not created by this job.
E_TYPE	8002H	Either the connection parameter is not a token for a connection object, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
E_FNEXIST	0021H	The specified device 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 on an open connection; use with any type of file.

Syntax, PL/M and C

```
CALL rq$a$read (connection, buff_ptr, count, resp_mbox,
      except_ptr);
```

```
rq_a_read (connection, buff_ptr, count, resp_mbox, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
connection	SELECTOR	SELECTOR
buff_ptr	POINTER	UINT_8 far *
count	WORD_32	NATIVE_WORD
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection

A token for the open file connection to be read.

buff_ptr

A pointer to the buffer that receives the data. The specified buffer can be in a segment allocated by the Nucleus, but this is not a requirement.

count The number of bytes to be read.

resp_mbox

A token for the mailbox that receives the IORS indicating the status of the read operation. A null selector means that you do not want to receive the IORS.

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 EOF, 0 returns.

See also: IORS, Chapter 1,
Accessing the IORS, *Programming Techniques*

except_ptr

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

A call to **a_read** will not be successful unless the mode of the open connection permits reading.

See also: **a_open**, **a_change_access**, **s_change_access**

The data is read as a string of bytes, starting at the current position of the connection's file pointer. Any number of bytes can be requested. It is more efficient to start reads on device block boundaries. After the read operation is finished, the file pointer points just past the last byte read.

DOS directory files can only be read a multiple of 6 bytes at a time, on 6-byte boundaries. This corresponds directly to the Named File Driver structure. Otherwise, E_SUPPORT returns.

Because segments have a maximum length of 4 Gbytes, data transfers of this size can be requested.

If all the connections to a stream file are requesting read operations, 0 returns along with an E_FLUSHING condition code.

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_BAD_BUFF	8023H	At least one of these is true: • The target memory buffer is not a writable segment. • The target memory buffer crosses a segment boundary.
E_BUFFERED_CONN	0036H	The specified connection was opened with an EIOS call. Use the EIOS **s_read_move** rather than the BIOS **a_read**.
E_EXIST	0006H	At least one of these is true: • One or more of the `connection` or `resp_mbox` parameters is not a token for an existing object. • 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	At least one of these is true: • The specified connection was not created by this job. • The request involved a DOS directory but did not follow the 6-byte boundary, multiple of 6-byte restriction.
E_TYPE	8002H	Either the connection parameter is not a token for a connection object, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
E_CONN_NOT_OPEN	0034H	This connection is not open for reading or updating.
E_FLUSHING	002CH	At least one of these is true: • The specified connection was closed before the read operation was completed. • 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. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*

a_rename_file

Changes the pathname of a named (including DOS and remote) data or directory file.

Syntax, PL/M and C

```
CALL rq$a$rename$file (connection, user, prefix, subpath_ptr,
      resp_mbox, except_ptr);

rq_a_rename_file (connection, user, prefix, subpath_ptr,
      resp_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection`	SELECTOR	SELECTOR
`user`	SELECTOR	SELECTOR
`prefix`	SELECTOR	SELECTOR
`subpath_ptr`	POINTER	STRING far *
`resp_mbox`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

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 null selector specifies the default user object.

For DOS files, the BIOS ignores this parameter because the user is always World.

`prefix`

A token for the connection to be used as the starting point in a path scan. A null selector specifies the default prefix.

`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_ptr` parameter cannot be a null STRING.

`resp_mbox`

The mailbox that receives a token for an IORS. A null selector means that you do not want to receive an IORS.

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

Renaming a directory changes the paths of any files contained in the directory.

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. All DOS users may rename files as long as the World user has write access to the file.

See also: **a_change_access**, **s_change_access**

For named data or directory files, this call 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.

Restrictions are:

- DOS users cannot rename a file or a directory to a different subdirectory.
- Any attempt to rename a directory as its own parent causes the BIOS to return an exception code.
- You cannot simultaneously rename a file and move it to another device.

The **a_rename_file** system call cannot rename an iRMX-NET virtual root directory, a file in a virtual root directory, or a public directory on a remote server. Otherwise, an E_FACCESS condition code returns.

The BIOS does not check the access rights of an iRMX-NET remote named file when you create a connection to the file, but checks during operations on the connection. This won't affect your programs if you:

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

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_DEV_OFF_LINE	002EH	The prefix parameter in this system call refers to a logical connection to a device. One of these is true of the device: • It has been physically attached but is now off-line. • It has been logically attached but never physically attached. See also: **attachdevice**, *Command Reference*
E_EXIST	0006H	At least one of these is true: • One or more of the `connection`, `user`, `prefix`, or `resp_mbox` parameters is not a token for an existing object. • The connection specified by the `prefix` and/or `connection` parameters is being deleted. • The connection for a remote driver is no longer active.
E_IFDR	002FH	This system call applies only to named or DOS files, but the connection parameter specifies 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 • 255 outstanding I/O operations for the specified user object • 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 the call.

E_NOPREFIX	8022H	The call specified a default prefix using a null selector, but a default prefix cannot be found for one of these reasons: • When this job was created, a 0 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 is not a null selector, it is not a user object. Otherwise, it specifies a default user object, but no default user object can be found for one of these reasons: • When this job was created, a 0 was specified for its object directory, so the job cannot catalog a default user object. • The job's directory can have entries but a default user object is not cataloged there. • The cataloged object *r?iouser* is not a user object. Treat *r?iouser* as a reserved word.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NOT_SAME_DEVICE	003AH	One or more of these is true: • The `connection` and the `prefix` parameters refer to different devices. • An attempt was made to rename a file across volumes.
E_PATHNAME_SYNTAX	003EH	One or more of these is true: • The specified pathname contains invalid characters or has 0 length. • The subpath of the specified remote file exceeds 27 bytes.
E_SUPPORT	0023H	The specified connection was not created by this job.

E_TYPE	8002H	At least one of these is true: • The `connection` parameter is not a token for a connection object. • The `prefix` parameter is a token for an object of the wrong type. It must be either a connection object or a logical device object created by the EIOS. • The resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
E_DEV_DETACHING	0039H	The file specified is on a device that the system is detaching.
E_FACCESS	0026H	At least one of these is true: • The specified file does not have add-entry access to the parent directory. • The specified connection does not have delete access to the file. • The call is attempting to rename the root directory or a bit-map file.
E_FEXIST	0020H	A file with the specified pathname 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_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 cannot be accessed; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*

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. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*
E_IO_MEM	0042H	The memory available to the BIOS job is not sufficient to complete the call.
E_LIMIT	0004H	Processing this call would deplete the remote server's resources.
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 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 user object password does not match the password of the user defined on the remote server.
E_PATHNAME_SYNTAX	003EH	The syntax of the specified remote file pathname is illegal; it must follow the naming conventions of the server.
E_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF.
E_SPACE	0029H	At least one of these is true: • The volume is full. • 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	A DOS user attempted to rename a directory as a subdirectory.

a_seek

Moves the file pointer of an open connection, for physical and named (including DOS and remote) data or directory files.

Syntax, PL/M and C

```
CALL rq$a$seek (connection, mode, move_size, resp_mbox,
      except_ptr);
```

```
rq_a_seek (connection, mode, move_size, resp_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
mode	BYTE	UINT_8
move_size	WORD_32	UINT_32
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection

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

mode Describes the movement of the file pointer:

Value	File Pointer Movement
1	Back by move_size bytes; if the pointer moves past the beginning of the file, it is set to 0 (first byte).
2	Set to the location specified by move_size.
3	Forward by move_size bytes.
4	Move to the EOF, then back by move_size bytes; if the pointer moves past the beginning of the file, it is set to 0 (first byte). This option is not supported for DOS directories; E_SUPPORT returns.

move_size

The number of bytes involved in the seek. The interpretation of `move_size` depends on the `mode` setting.

resp_mbox

The mailbox that receives a token for an IORS. A null selector means that you do not want to receive an IORS.

except_ptr

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

Use this call for random access to file contents. The file pointer can be moved to any byte position in the file; the first byte is byte 0.

For named files, you can use **a_seek** to position the file pointer beyond the EOF. If you then invoke **a_write**, the BIOS extends the file to accommodate the writing operation. The file will contain random data between the old EOF and the pointer, where the write begins.

You can also invoke **a_read** with the file pointer beyond the EOF, but the BIOS returns 0 in the `actual` field of the IORS, signifying the EOF.

See also: **a_write**, **a_read** in this chapter,
IORS, Chapter 1,
Accessing the IORS, *Programming Techniques*

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_BUFFERED_CONN	0036H	The connection parameter was produced by the EIOS.
E_EXIST	0006H	At least one of these is true: • One or more of the `connection` or `resp_mbox` parameters is not a token for an existing object. • The connection is being deleted. • 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 these is true: • The calling task's job has already reached its object limit. • 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_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The mode parameter value is outside the range 1-4.
E_SUPPORT	0023H	Either the specified connection was not created by this job or the file is a directory file.
E_TYPE	8002H	Either the connection parameter is not a token for a connection object, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
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. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*
E_PARAM	8004H	This call attempted to seek beyond the end of the physical device. This applies only to physical files.

set_default_prefix

Sets the default prefix for an existing job.

Syntax, PL/M and C

```
CALL rq$set$default$prefix (job, prefix, except_ptr);

rq_set_default_prefix (job, prefix, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`job`	SELECTOR	SELECTOR
`prefix`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`job` A token for the job whose default prefix is to be set. A null selector specifies the current job.

`prefix`

A token for the connection that is to become the default prefix.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This system call catalogs the connection supplied as the `prefix` parameter in the object directory of the job supplied as the `job` parameter. The BIOS catalogs the prefix under the name *$*. If an object is already cataloged under the name *$*, the BIOS uncatalogs that object before cataloging the new prefix.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	When this job was created, a 0 was specified for the object directory, so a default prefix cannot be cataloged.
E_EXIST	0006H	One or more of the job or prefix parameters is not a token for an existing object.
E_LIMIT	0004H	The prefix parameter 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	At least one of these is true: • The `job` parameter is not a job token. • The prefix parameter is not a token for a connection object or a logical device object created by the EIOS.

set_default_user

Sets the default user object for an existing job.

See also: Default user object, *System Concepts*

Syntax, PL/M and C

```
CALL rq$set$default$user (job, user, except_ptr);

rq_set_default_user (job, user, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`job`	SELECTOR	SELECTOR
`user`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`job` A token for the job whose default user object is to be set. A null selector designates the calling task's job.

`user`

A token for the user object that is to become the default user.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	When this job was created, a 0 was specified for the object directory, so a default prefix cannot be cataloged.
E_EXIST	0006H	One or more of the job or user parameters is not a token for an existing object.
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 parameter is a token for an object of the wrong type.

a_set_extension_data

Writes the extension data for a BIOS named data or directory file. This call is not valid for DOS files or for files accessed through NFS. For DOS files the call is ignored.

Syntax, PL/M and C

```
CALL rq$a$set$extension$data (connection, data_ptr, resp_mbox,
      except_ptr);
```

```
rq_a_set_extension_data (connection, data_ptr, resp_mbox,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
data_ptr	POINTER	EXT_DATA_STRUCT far *
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection

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

data_ptr

A pointer to this structure:

```
DECLARE ext_data STRUCTURE(
   count                 BYTE,
   info(*)               BYTE);
```

or

```
typedef struct {
   UINT_8                count;
   UINT_8                info[_NUM_EXT_INFO];
                                   /* adjust to fit count */
} EXT_DATA_STRUCT;
```

Where:

count — Number of bytes up to 255 of extension data being written. For remote files, set to 0.

info — The extension data.

`resp_mbox`

The mailbox that receives a token for an IORS. A null selector means that you do not want to receive an IORS.

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

Each file created through the BIOS has an associated file descriptor containing information about the file. Some of that information is used by the BIOS and can be accessed by tasks through **a_get_file_status**. Up to 255 additional bytes of the file descriptor, known as extension data, are available for use by OS extensions, depending upon how the volumes were formatted. For named volumes, the first three bytes of this extension data are reserved for use by the BIOS.

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 using the named file driver.

See also: **a_get_extension_data**, **a_get_file_status**

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	At least one of these is true: • One or more of the `connection` or `resp_mbox` parameters is not a token for an existing object. • 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_CONFIGURED	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	Either the connection parameter is not a token for a connection object, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
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. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*
E_PARAM	8004H	At least one of these is true: • The `count` field in the `ext_data` structure contains a value greater than the value specified when the disk was formatted. • The connection parameter references a remote file and the count field does not contain a 0.

a_set_file_status

Changes the owner and/or time stamps of a file.

Syntax, PL/M and C

```
CALL rq$a$set$file$status (connection, set_info_ptr, resp_mbox,
      except_ptr);
```

```
rq_a_set_file_status (connection, resp_mbox, set_info_ptr,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
set_info_ptr	POINTER	SET_FILE_STATUS_STRUCT far *
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection

A token for a connection to the file.

set_info_ptr

A pointer to this structure:

```
DECLARE set_file_status_struct STRUCTURE(
   select                 WORD_16,
   owner                  WORD_16,
   create_time            WORD_32,
   modify_time            WORD_32,
   access_time            WORD_32);
```

or

```
typedef struct set_file_status_struct {
   UINT_16                select;
   UINT_16                owner;
   UINT_32                create_time;
   UINT_32                modify_time;
   UINT_32                access_time;
} SET_FILE_STATUS_STRUCT
```

Where:

`select` Specifies the file attributes to set; encoded as:

Bit	Meaning
0	Change owner
1	Set creation time
2	Set last modified time
3	Set last access time
4-5:	Reserved, must be 0

`owner` File owner ID

`create_time`
The date and time the file was created.

`modify_time`
The date and time the file was last modified.

`access_time`
The date and time the file was last accessed.

`resp_mbox`
A token for a mailbox that receives the IORS. A null selector indicates no IORS desired.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

When setting a file's time stamps, use care if you're using other I/O operations on the open connection. Write connections (such as **rq_a_write**, **rq_a_truncate**, etc.) will cause the last modified and last access time stamps to be set to the current time. Read operations (such as **rq_a_read**, **rq_a_get_file_status**, etc.) will cause the last access time stamp to be set to the current time. In addition, write operations may cause a buffer flush when the connection is closed, overriding this system call and updating the time stamps to the current time.

See also: **rq_a_write**, **rq_a_truncate**, **rq_a_read**, **rq_a_get_file_status**

Not all file drivers support this system call due to file system limitations. This is the level of support provided by each standard file driver:

File Driver	**Support**
Physical	Not supported
Stream	Not supported
DOS	Only last modified time
Named	Full support
Remote	Local full support, remote support is system-dependent
EDOS	Only last modified time
NFS	Fully supported except you cannot change the owner

On file drivers that support the setting the time stamp(s) but not changing the file owner (for example, DOS and EDOS) E_SUPPORT is always returned if the `change_owner` bit is set in the `select` word, and no other action is performed. In general, make the application file-driver independent, and make separate calls to change the file owner and the file time stamps.

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	At least one of these is true: • The `resp_mbox` parameter is not a mailbox token. • The connection is being deleted. • The connection for a remote driver is no longer active.
E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its object limit. • 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_CONFIGURED	0008H	This system call is not part of the present configuration.

E_SUPPORT	0023H	The file driver associated with the specified connection does not support this system call.
E_TYPE	8002H	One or more of the connection or resp_mbox parameters is a token for an object of the wrong type.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
E_CONN_NOT_OPEN	0034H	The connection is either not open, or is not open with write access.
E_FACCESS	0026H	The specified connection does not have update or append access to the file.
E_IO	002BH	An I/O error occurred which might have prevented the operation from completing.
E_NOT_FILE_CONN	0032H	For remote and NFS 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 UDF.

set_global_time

Sets the battery-backed-up hardware clock to a specified time.

Syntax, PL/M and C

```
CALL rq$set$global$time (date_time_ptr, except_ptr);
```

```
rq_set_global_time (date_time_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
date_time_ptr	POINTER	SET_TIME_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

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 this form:

```
DECLARE set_time STRUCTURE (
   seconds              BYTE,
   minutes              BYTE,
   hours                BYTE,
   days                 BYTE,
   months               BYTE,
   years                WORD_16);
```

or

```
typedef struct {
   UINT_8               seconds;
   UINT_8               minutes;
   UINT_8               hours;
   UINT_8               days;
   UINT_8               months;
   UINT_16              years;
} SET_TIME_STRUCT;
```

Where:

`seconds`	The value to which the seconds counter is set. Do not exceed 59.
`minutes`	The value to which the minutes counter is set. Do not exceed 59.
`hours`	The value to which the hours counter is set. Do not exceed 23.
`days`	The value to which the days counter is set. Do not exceed 3.
`months`	The value to which the months counter is set. Do not exceed 2.
`years`	The value to which the years counter is set.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The BIOS writes the new values into the appropriate registers on the clock hardware.

This system call supports the Time-of-Day clock on the Multibus I SBC 546 Terminal Communications Controller board, the Multibus II CSM, the Multibus I SBC 86C38 board, and PC systems.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	This call was made from an environment that did not contain a hardware clock.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	One or more of the values specified in the set_time structure is illegal.
E_SHARE	0028H	The global time-of-day clock was busy because another entity was accessing it.
E_SUPPORT	0023H	The configured clock type is not a supported type.

a_special

Enables tasks to perform a variety of device-level functions. This call is not valid for DOS files or for devices accessed through NFS. For DOS files, the call returns an E_IFDR exception.

Syntax, PL/M and C

```
CALL rq$a$special (connection, spec_func, ioparm_ptr,
      resp_mbox, except_ptr);
```

```
rq_a_special (connection, spec_func, ioparm_ptr, resp_mbox,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection`	SELECTOR	SELECTOR
`spec_func`	WORD_16	UINT_16
`ioparm_ptr`	POINTER	void far *
`resp_mbox`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

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`

Specifies the function being requested when combined with the file driver associated with the `connection` parameter. Each function is described in detail after the Additional Information heading.

Note

Bits 8 and 12 of the `spec_func` field are reserved; do not use values that manipulate these bits in your applications or device drivers. Mask bits 8 and 12 when your device driver receives a function code from the I/O system.

This table summarizes the values you can assign to the `spec_func` parameter:

Function Code	File Driver	Description
0	Physical	Format track
0	Stream	Query
1	Stream	Satisfy
2	Physical/Named	Notify (The only function supported for remote servers.)
3	Physical	Get disk data
3	Physical	Get tape data
4	Physical	Get terminal data
5	Physical	Set terminal data
6	Physical	Set signal
7	Physical	Rewind tape
8	Physical	Read tape file mark
9	Physical	Write tape file mark
10	Physical	Retension tape
11	Physical	Reserved for Intel
12	Physical	Set bad track/sector information
13	Physical	Get bad track/sector information
14, 15		Reserved
16	Physical	Get terminal status
17	Physical	Cancel terminal I/O
18	Physical	Resume terminal I/O
19	Physical/Named	Perform Disk Mirroring
20	Named/DOS/EDOS	Get device free space data
21-32767		Reserved
32768-65535		Available for user devices, except for values that use bits 8 or 12.

`ioparm_ptr`

A pointer to a parameter block whose contents depends upon the special function being requested. Enter a null value if the special function you request does not require a parameter block.

`resp_mbox`

The mailbox that receives a token for an IORS. A null selector means that you do not want to receive an IORS.

`except_ptr`

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

The special functions (specified with the `spec_func` parameter) are described below, in numerical order.

Format a Track (Function Code 0)

Call **a_special** with an open file connection, `spec_func` equal to 0, and `ioparm_ptr` pointing to this structure:

```
DECLARE format_track STRUCTURE(
   track_number          WORD_16,
   interleave            WORD_16,
   track_offset          WORD_16,
   fill_char             WORD_16);
```

or

```
typedef struct {
   UINT_16               track_number;
   UINT_16               interleave;
   UINT_16               track_offset;
   UINT_8                fill_char;
} FORMAT_TRACK_STRUCT;
```

Where:

`track_number`

The number of the track to be formatted: from 0 to 1 less than the number of tracks on the volume. Other values cause an E_SPACE condition code. When formatting a RAM-disk or a tape, use 0.

`interleave`

The interleave factor for the track: the number of physical sectors to advance when locating the next logical sector. 0 or 1 skips no physical sectors between logical sectors. If the specified interleave factor is greater than the number of physical sectors on a track, the OS divides the specified value by the number of physical sectors and uses the remainder as `interleave`. This field does not apply to tapes.

`track_offset`

The number of physical sectors to advance when locating the first logical sector (index mark). This field does not apply to tapes.

`fill_char` A character with which each sector is written. Some drivers ignore this value and fill the sector with a character they establish.

Query Stream File Operations (Function Code 0)

Call **a_special**, using the connection for a stream file, with `spec_func` set to 0. The `ioparm_ptr` parameter is ignored. Use this function to find out what is being requested by another 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.

If a read or write request is queued at the file, the information requested returns 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.

See also: IORS, Chapter 1,
Accessing the IORS, *Programming Techniques*,
IORS fields, *Driver Programming Concepts*

If no read or write request is occurring on 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 is satisfied, the IORSs for both requests return with E_STREAM_SPECIAL in the `status` field.

Satisfy Stream File Transactions (Function Code 1)

Call **a_special**, with a stream file connection and `spec_func` set to `1; the ioparm_ptr` is ignored. Use this function to 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 determine the number of bytes sent by checking the `actual` field in their respective IORSs. An E_STREAM_SPECIAL condition code returns if no request is queued at the stream file or if a request for information is queued.

See also: IORS, Chapter 1,
Accessing the IORS, *Programming Techniques*

Usually, when 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 requests to read 52 bytes, but Task B only writes 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.

Request Notification that a Volume is Unavailable (Function Code 2)

Use this function to be notified when a volume becomes unavailable because a person has opened a door to a diskette drive or pressed the online/offline button on other mass storage drives. Call **a_special** with a token for a device connection, with `spec_func` set to 2, and with `ioparm_ptr` pointing to this structure:

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

or

```
typedef struct notify_struct {
   SELECTOR              mailbox;
   SELECTOR              object;
} NOTIFY_STRUCT;
```

Where:

`mailbox` A token for a mailbox. Some task should be dedicated to waiting at the mailbox.

`object` A token for an object. When the BIOS detects that the volume is unavailable or is detached by **a_physical_detach_device**, this object is sent to the mailbox.

For most drives, notification occurs immediately. For some 5.25" diskette drives, notification occurs when the BIOS first tries to perform an operation on the unavailable volume. On those drives, use this sequence of events when changing volumes:

⚠ **CAUTION**

Whenever you change a volume without first detaching the device and then reattaching it, the BIOS 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.

1. Detach the unit, using **a_physical_detach_device**.
2. Remove the old volume.
3. Install the new volume.
4. Reattach the unit, using **a_physical_attach_device**.

If the volume is unavailable, the BIOS will not execute I/O requests to the device on which the volume was mounted. Such requests return with the `status` field of the IORS set to E_IO and the `unit_status` field set to IO_OPRINT, meaning that operator intervention is required.

See also: IORS, Chapter 1,
Accessing the IORS, *Programming Techniques*

If any task issues a subsequent notification request for the same device connection, the BIOS replaces the old `mailbox` and `object` values with the new s specified. It does not return an exception code.

To restore the availability of a volume, perform these 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 null selector value in the `mailbox` parameter.

For iRMX-NET 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 client or server, or line transmission errors. The remote file driver returns E_IO to the `status` field and IO_OPRINT to the `unit_status` field of the IORS.

See also: IORS, Chapter 1,
Accessing the IORS, *Programming Techniques*

To restore the availability of a remote server, perform these 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.

Get Disk Data (Function Code 3)

Use this function to obtain specification information about a Winchester drive with an SBC 214/215G/221(S) disk controller or a drive with an SBC 220 SMD controller.

Call **a_special** with a token for a device connection, `spec_func` set to 3, and `ioparm_ptr` pointing to this structure:

```
DECLARE disk_drive_data STRUCTURE(
   cylinders             WORD_16,
   fixed                 BYTE,
   removable             BYTE,
   sectors               BYTE,
   sector_size           WORD_16,
   alternates            BYTE);
```

or

```
typedef struct {
   UINT_16               cylinders;
   UINT_8                fixed;
   UINT_8                removable;
   UINT_8                sectors;
   UINT_16               sector_size;
   UINT_8                alternates;
} DISK_DRIVE_DATA_STRUCT;
```

Where:

`cylinders` The total number of cylinders on the drive.

`fixed` The number of heads on the fixed disk drive.

`removable` The number of heads on the floppy disk drive.

`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.

Get Tape Data (Function Code 3)

Use this function to obtain specification information about a tape drive connected to an SBC 214 controller, an SBC 212 controller, or an SBX 217C board mounted on an SBC 215G controller.

Call **a_special** with a token for the device connection, with `spec_func` set to 3, and with `ioparm_ptr` pointing to this structure:

```
DECLARE tape_drive_data STRUCTURE(
   tape                  BYTE,
   reserved(7)           BYTE);
```

or

```
typedef struct {
   UINT_8                tape;
   UINT_8                reserved[7];
} TAPE_DRIVE_DATA_STRUCT;
```

Where:

`tape` Receives information encoded as:

Bits	**Meaning**
7-4	Number of tracks on the tape
3-1	Reserved
0	Indicates whether the unit is present 0 = Unit not present 1 = Unit present

Get Terminal Data (Function Code 4)
Set Terminal Data (Function Code 5)

Terms unique to terminal devices, such as line editing, translation, OS Command (OSC) sequences, and the Terminal Support Code (TSC), appear in this description. Terminal attributes relate with OSC characters and sequences. Where this applies, the label OSC x:y appears in parentheses, where x and y are upper-case characters. You can use the OSC Query sequence when debugging, to ensure that your tasks invoked **a_special** correctly.

See also: OSC sequences, translation, line editing, raw input and type-ahead buffers, *Driver Programming Concepts*

Call **a_special** with a token for a connection to a terminal device driver; get or set the terminal attributes with spec_func equal to 4 or 5. Ioparm_ptr points to a structure of this form. If any of the first five parameters (connection_flags through scroll_lines) is 0, the BIOS leaves the parameter at its previous setting. In this way, you can set some parameters without affecting others.

```
DECLARE term_attrib STRUCTURE(
   num_words              WORD_16,
   num_used               WORD_16,
   connection_flags       WORD_16,
   terminal_flags         WORD_16,
   in_baud_rate           WORD_32,
   out_baud_rate          WORD_32,
   scroll_lines           WORD_16,
   page_width             BYTE,
   page_length            BYTE,
   cursor_offset          BYTE,
   overflow_offset        BYTE,
   special_modes          WORD_16,
   high_water_mark        WORD_16,
   low_water_mark         WORD_16,
   fc_on_char             WORD_16,
   fc_off_char            WORD_16,
   link_parameter         WORD_16,
   spc_hi_water_mark      WORD_16,
   special_char(4)        BYTE);
```

or

```
typedef struct term_attrib_struct {
   UINT_16                 num_words;
   UINT_16                 num_used;
   UINT_16                 connection_flags;
   UINT_16                 terminal_flags;
   NATIVE_WORD             in_baud_rate;
   NATIVE_WORD             out_baud_rate;
   UINT_16                 scroll_lines;
   UINT_8                  page_width;
   UINT_8                  page_length;
   UINT_8                  cursor_offset;
   UINT_8                  overflow_offset;
   UINT_16                 special_modes;
   UINT_16                 high_water_mark;
   UINT_16                 low_water_mark;
   UINT_16                 fc_on_char;
   UINT_16                 fc_off_char;
   UINT_16                 link_parameter;
   UINT_16                 spc_hi_water_mark;
   UINT_8                  special_char[4];
} TERM_ATTRIB_STRUCT;
```

Where:

`num_words` The number of 16-bit words, beyond the `num_words` and `num_used` fields, containing the terminal data. To access all of the information, set this field to at least 18. This field does not refer to the number of parameters, since the NATIVE_WORD parameters can be 32 bits, and other parameters are only one byte long.

`num_used` The number of 16-bit words of valid parameter data. For Get Data function, **a_special** fills in the structure with up to `num_words` of the current values and sets `num_used` to the number of 16-bit words actually returned.

`connection_flags`
Attributes that apply only to this terminal connection. Changes made with `connection_flags` take effect after a read operation. If 0, all bits are ignored. After changing the connection attributes, immediately read the connection to ensure that the changes are in effect. If not in flush mode, set the connection to flush mode, then read 255 characters from the connection. The read returns immediately with the available characters.

Bits	Value	Meaning
15-10	0	Reserved, set to 0.
9	0	Characters move from raw-input buffer to type-ahead buffer.
	1	Bypass the type-ahead and line-edit buffers. This disables all TSC features.
8	0	The interrupt task moves characters from the raw input buffer to the type-ahead buffer.
	1	The service task does it.
7-6	0	Act upon OSC sequences in the input or output stream (OSC C:C).
	1	Input stream only.
	2	Output stream only.
	3	Do not act upon any OSC sequences.
5	0	Accept output control characters (OSC C:O).
	1	Ignore output control characters.
4	0	Set character parity bit to 0 (OSC C:W).
	1	Do not alter parity bit.
3	0	Set parity bit to 0 (OSC C:R).
	1	Do not alter parity bit.
2	0	Echo characters to the screen (OSC C:E).
	1	Do not echo.
1-0	0	Invalid Entry, E$PARAM returned.
	1	Transparent mode, no line editing. Input is transmitted to a requesting task exactly as entered at the terminal except for control characters. Data is buffered until the requested number of characters has been entered.
	2	Normal mode, line editing. Edited data accumulates in a buffer until a <CR> is entered (OSC C:T) except for control characters.
	3	Flush mode, no line editing. Input is transmitted to the requesting task exactly as in transparent mode. Data is buffered until an input request is received. Then the contents of the buffer (or the number of characters requested, if the buffer contains more than that number) are transmitted to the requesting task. Any characters remaining in the buffer are saved for the next input request except for control characters.

terminal_flags

Attributes that apply to the terminal and therefore to all connections to the terminal.

Bits	Value	Meaning
15-13		Reserved, set to 0
12	0	The vertical axis coordinates increase from top to bottom on the screen (OSC T:F)
	1	They decrease
11	0	The horizontal axis coordinates increase from left to right across the screen (OSC T:F)
	1	They decrease
10	0	Horizontal coordinate listed or entered first (OSC T:F)
	1	Vertical coordinate first
9	0	Disable control character translation
	1	Enable translation (OSC T:T)
8-6	0	Output parity bit always 0 (OSC T:W)
	1	Output parity bit always
	2	Even parity
	3	Odd parity
	4	No output parity
	5-7	Invalid values
5-4	0	Input parity bit always 0 (OSC T:R)
	1	Input parity bit unchanged
	2	Even parity, parity bit indicates the presence () or absence (0) of an error on input
	3	Odd parity, parity bit indicates the presence () or absence (0) of an error on input
3	0	No modem
	1	Used with a modem (OSC T:M)
2	0	VDT output medium (OSC T:H)
	1	Printed hard copy
1	0	Full duplex line protocol (OSC T:L)
	1	Half duplex
0		Reserved, set to 1

`in_baud_rate`

The input baud rate indicator (OSC T:I).

Value	**Meaning**
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 (OSC T:O).

Value	**Meaning**
0 - 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; use `in_baud_rate` to set the baud rate and specify a 1 for `out_baud_rate`.

`scroll_lines`

The maximum number of lines sent each time the operator enters the control character when ready for terminal display (OSC T:S); the default is <Ctrl-W>.

`page_width`

The number of character positions on each line of the terminal's screen (OSC T:X).

`page_length` The number of lines on the terminal's screen (OSC T:Y).

`cursor_offset`

The value that starts the numbering sequence of both the X and Y axes (OSC T:U).

`overflow_offset`

The value to which the numbering of the axes must fall back after reaching 27 (OSC T:V).

`special_modes`

The remainder of the terminal attributes apply only to buffered devices, such as the SBC 548 and the SBC 88/56 boards. These devices maintain their own input and output buffers separately from those managed by the BIOS's TSC. If you aren't sure whether you can set these fields, check bit 5 of this parameter; if set, your board is a buffered device. The Hostess 550 does not support these fields.

Bits	**Value**	**Meaning**
15	0	Not a buffered device.
	1	Buffered device.
14-2		Reserved, set to 0.
1	0	Disable Special Character Mode. Send special characters through the normal input stream.
	1	Enable Special Character Mode (OSC T:D). Send an interrupt whenever a special character defined in the special_char array is typed. This feature is used in conjunction with the spc_hi_water_mark field to indicate the number of characters to buffer before the interrupt is sent. If the characters are signal characters, the TSC sends units to the appropriate semaphores when the characters reach the line-edit buffer.
0	0	Disable flow control.
	1	Enable flow control (OSC T:G).

`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 (OSC T:J).

`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 (OSC T:K.).

`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 `low_water_mark`. Normally this character tells the connecting device to resume sending data (OSC T:P).

`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 (OSC T:Q).

`link_parameter`

Specifies the characteristics of the physical link between the terminal and a device (OSC T:N). Not all device drivers support `link_parameter`. This field is supported by those boards using the TCC driver and ATCS driver. You cannot change `link_parameter` values for a COM port on a PC.

See also: Supplied device drivers, *Command Reference*

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

Bits	**Value**	**Meaning**
15	0	Link_parameter field is not used; the terminal_flags field is used instead.
	1	Link_parameter is used. The driver passes the low-order byte to the controller, which sets the parity, character length, and stop bits.
14-6		Reserved
5-4	0	1 stop bit
	1	1 1/2 stop bits
	2	2 stop bits
	3	Invalid value
3-2	0	6 bits/character. Unused bit positions are ignored in transmit data, set to 1 in receive data.
	1	7 bits/character. Unused bit position is ignored in transmit data, set to 1 in receive data.
	2	8 bits/character.
	3	Invalid value.
1-0	0-1	Invalid value.
	2	Even parity.
	3	Odd parity.

`spc_hi_water_mark`

This field is used in conjunction with the Special Character Mode field. If Special Character Mode is enabled in the `special_modes` field, the device's input buffer fills to contain this number of special characters before an interrupt is sent (OSC T:A).

`special_char(4)`

Holds special characters (OSC T:Z). If you define less than 4 special characters, fill the remaining slots in the array with duplicates of the last one.

Set Signal Characters (Function Code 6)

This function associates a keyboard character with a semaphore, so that whenever the character is entered into the terminal, the BIOS automatically sends a unit to the semaphore. Character-semaphore pairs are called signals. Up to 2 signal characters, each character being associated with a different semaphore, are allowed. Call **a_special** with a device connection, `spec_func` equal to 6, and `ioparm_ptr` pointing to this structure:

```
DECLARE signal_pair STRUCTURE(
   semaphore              TOKEN,
   character              BYTE);
```

or

```
typedef struct {
   SELECTOR               semaphore;
   UINT_8                 character;
} SIGNAL_PAIR_STRUCT;
```

Where:

`semaphore` A token for the semaphore to be associated with the character. To delete a signal character, use a null selector.

`character` The signal character.

Value	Meaning
20H-40H	Type-ahead buffer (and input buffer if a buffered device) is cleared and a unit is sent to the associated semaphore when it receives a character in the 0 to FH range (add 20H to desired control character).
0-1FH, 7FH	TSC sends a unit to the associated semaphore when it receives this ASCII value

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

Use these functions to perform 4 different operations on tape drives only:

Code	Meaning
7	The tape drive rewinds a tape to its load point. This function also terminates tape read and write operations. If a write operation, the tape drive writes a file mark before rewinding the tape.
8	The tape drive moves the tape to the next file mark. This function also terminates tape read operations. The value of the search field in the read_file_mark structure (see below) determines the direction of the search.
9	The tape drive writes a file mark at the current position. This function also terminates tape write operations.
10	The tape drive fast-forwards the tape to the end and then rewinds it to the load point.

If using Function Code 8, `ioparm_ptr` points to this structure:

```
DECLARE read_file_mark STRUCTURE (search BYTE);
```

or

```
typedef struct {
   UINT_8                 search;
} READ_FILE_MARK_STRUCT;
```

Where:

`search` A value indicating the direction of the search:

Value	Meaning
00	search forward
0FFH	search backward (for start/stop drives only)

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

Use these functions to set (write) or get (read) the bad track information of a volume. When writing, bad track information already on the volume will be overwritten. If you wish to change existing information, get, modify, then set it. The `ioparm_ptr` parameter must point to this structure:

```
DECLARE bad_track_info STRUCTURE(
   reserved                WORD_16,
   count                   WORD_16,
   bad_tracks(1024)        WORD_32),
   badtracks(1024)         STRUCTURE(
   cylinder                WORD_16,
   head                    BYTE,
   sector                  BYTE)
   AT (@bad_track_info.bad_tracks);
```

or

```
typedef struct {
   UINT_16                 cylinder;
   UINT_8                  head;
   UINT_8                  sector;
} BAD_TRACK_STRUCT;

typedef struct {
   UINT_16                 reserved;
   UINT_16                 count;
   BAD_TRACK_STRUCT        bad_tracks[1024];
} BAD_TRACK_INFO_STRUCT;
```

Where:

`reserved` Reserved for use by the device driver.

`count` The number of bad tracks/sectors listed in the `bad_tracks` structure, up to the maximum of 1024. A 0 in the `count` field indicates that no valid information is available (get) or that there are no bad tracks (set).

`bad_tracks`
A structure used to store the bad track/sector list. For each entry, a sub-structure defines the cylinder, head, and sector for each bad track. List bad tracks in ascending order.

Get Terminal Status (Function Code 16)

See also: Function Code 4, **a_special**,
Line editing, OSC sequences, translation, *Driver Programming Concepts*

Call **a_special** with a connection for the terminal, `spec_func` equal to 16, and `ioparm_ptr` pointing to this structure:

```
DECLARE term_status STRUCTURE(
   terminal_flags         WORD_16,
   input_conn_flags       WORD_16,
   input_state            WORD_16,
   input_conn             TOKEN,
   input_count            WORD_32,
   input_actual           WORD_32,
   raw_buf_count          WORD_16,
   type_ahead_count       BYTE,
   num_input_requests     BYTE,
   output_conn_flags      WORD_16,
   output_state           WORD_16,
   output_conn            TOKEN,
   output_count           WORD_32,
   output_actual          WORD_32,
   out_buf_count          WORD_16,
   num_output_requests    BYTE);
```

or

```
typedef struct {
   UINT_16                terminal_flags;
   UINT_16                input_conn_flags;
   UINT_16                input_state;
   SELECTOR               input_conn;
   NATIVE_WORD            input_count;
   NATIVE_WORD            input_actual;
   UINT_16                raw_buf_count;
   UINT_8                 type_ahead_count;
   UINT_8                 num_input_requests;
   UINT_16                output_conn_flags;
   UINT_16                output_state;
   SELECTOR               output_conn;
   NATIVE_WORD            output_count;
   NATIVE_WORD            output_actual;
   UINT_16                out_buf_count;
   UINT_8                 num_output_requests;
} TERM_STATUS_STRUCT;
```

Where:

`terminal_flags`

The current attributes associated with the terminal. For bit encoding information, see the `terminal_flags` parameter in the description of function codes 4 and 5.

`input_conn_flags`

The current attributes associated with the terminal's active input connection. For bit encoding information, see the `connection_flags` parameter in the description of function codes 4 and 5.

`input_state`

The internal state of this terminal's input connection. Encoded as:

Bits	**Value**	**Meaning**
15	0	Type-ahead buffer not full
	1	Full
14	0	In line-edit mode, current line not canceled
	1	Canceled
13, 12		Reserved
11	0	No modem query pending
	1	Modem query pending

Bits	Value	Meaning
10	0	Terminal not waiting for a carrier
	1	Waiting; must be configured for a modem
9	0	Terminal not waiting for a ring interrupt
	1	Waiting; must be configured for a modem
8	0	Terminal configured for a modem not available
	1	Available
7	0	In line-edit mode, last line not recalled
	1	Recalled with <Ctrl-R>
6	0	Escape sequence is not being processed
	1	Escape sequence is being processed
5	0	Current character not preceded by a <Ctrl-P>
	1	Preceded by a <Ctrl-P>; interpreted as data, not as a line editing character
4	0	Complete line not processed
	1	Processed and ready for transfer from the line-edit buffer to the application task's buffer
3	0	OSC sequence is not being processed
	1	OSC sequence is being processed
2		Reserved
1	0	Current input request not completed
	1	Completed
0	0	Input request has not been set up
	1	Set up

`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 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.

`type_ahead_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 bit encoding information, see the `connection_flags` parameter in the description of function codes 4 and 5.

`output_state`

The internal state of this terminal's output connection. Use this value to determine if a terminal's output is hindered in some way (for example, because an XOFF was received). If the logical-and of `output_state` and E0H is not 0, output is hindered. Resume terminal output by invoking **a_special** with function code 18. The bit encoding is:

Bits	**Value**	**Meaning**
15-10		Reserved.
9	0	Terminal's current output request not canceled.
	1	Canceled and is being flushed.
8	0	Output not blocked by XOFF.
	1	Output blocked.
7	0	Not in scroll mode.
	1	In scroll mode.
6	0	Output not blocked.
	1	Blocked.
5	0	Not discarding terminal output.
	1	In discarding mode.
4		Reserved
3	0	Transmitting characters on an interrupt-driven basis.
	1	Ready to transmit a character once the next output request arrives.
2	0	Output request has not been set up.
	1	Set up.

Bits	Value	Meaning
1-0	0	Output character processing is occurring normally.
	1	An ESC character has been encountered in the output stream requiring special handling; it may be part of an escape or OSC sequence or require translation.
	2	The previously encountered escape character is part of an OSC sequence that is being processed.
	3	The previously encountered escape character is part of an escape sequence that is being translated.

`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 TSC or the buffered device.

`num_output_requests`

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

Cancel Terminal I/O (Function Code 17)

This function cancels all requests associated with a specified connection to a terminal. It does not flush the outstanding input from the terminal.

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 canceled without affecting other input requests on the same terminal device.

Call **a_special** with a connection for the terminal, with `spec_func` equal to 17, and with `ioparm_ptr` pointing to this structure:

```
DECLARE cancel_io STRUCTURE (cancel_conn_t   TOKEN);
```

or

```
typedef struct {
   SELECTOR              cancel_conn_t;
} CANCEL_IO_STRUCT;
```

Where:

`cancel_conn_t`
A token for the connection whose requests are to be canceled. Setting `cancel_conn_t` to a null selector cancels all input requests associated with the specified connection. To determine which connection is active and can be canceled, invoke **a_special** with `spec_func` equal to 16 and check the token returned in the `input_conn` parameter.

Resume Terminal I/O (Function Code 18)

This function enables a program to resume an output request that is blocked because an output control character was entered at the terminal. Call **a_special** with any connection for the blocked terminal and with `spec_func` equal to 18. The `ioparm_ptr` parameter is ignored.

Perform Disk Mirroring (Function Code 19)

➡ **Note**

The two hard disks must have the same formatted capacity, device granularity and should be the same model number to ensure the same formatted capacity.

This function performs disk mirroring operations on the primary hard disk of the mirror set. The iRMX PCI device driver implements the actual mirroring, error detection and rollover, and on-line resynchronization.

See also: The *mirror.lit* and *mirror.h* files for the literal definitions for this subfunction

Each mirrored disk contains a structure located in the Volume Label at a byte offset of 896. When the first attach is performed on a hard disk, the device driver uses this structure to detect whether this hard disk was part of a mirror set and, if it was, to identify the name of the mirror. The format of this structure is:

```
DECLARE mirr_state_struct STRUCTURE(
   other_name(4)          BYTE,
   valid_flg              WORD_32,
   incarnation            WORD_32,
   prim_flg               BYTE,
   good_flg               BYTE);
```

or

```
typedef struct {
   UINT_8                 other_name[4];
   UINT_32                valid_flg;
   UINT_32                incarnation;
   UINT_8                 prim_flg;
   UINT_8                 good_flg;
} MIRR_STATE_STRUCT;
```

Where:

`other_name`

Specifies the null-terminated DUIB name of the other hard disk of the mirror set. The DUIB name must be in capital letters, be null terminated, and be a maximum of 14 characters, not including the null.

`valid_flg` Specifies if the mirror set is valid. A valid set has the value `600DD5CH` (looks like `gooddisc`) on both disks; an invalid set has the value `deadbeef`. If the mirror set is valid, the device driver automatically re-enables mirroring. The valid flag is set at the end of a normal detach, if no I/O errors have occurred. The device driver clears the flag on each disk when it reads the disk so that mirroring would not be automatically enabled if the system crashes.

`incarnation`

A pattern that is written on the disks to uniquely identify the correct instance of a mirror set.

`prim_flg` Specifies if this hard disk is the primary unit of a mirror set.

Value	**Meaning**
1	Primary
2	Secondary

good_flg Indicates whether this disk was good when it was detached.

Value	Meaning
0AAH	Disk was good
055H	Not good

To perform disk mirroring operations, call **a_special** with a token for a connection and spec_func set to 19. Ioparm_ptr must point to a data structure which contains a command byte followed by other fields that are dependent on the subfunction being performed. These are the subfunctions:

Value	Meaning
1	Create mirror set
2	Enable mirroring with resync
3	Disable mirroring
4	Request mirror event notification
5	Get mirror status
6	Get mirror attach status
7	Set mirror options
8-0FFH	Reserved

Subfunction 1 creates the mirror set with the specified secondary hard disk. The primary and secondary hard disk must have the same capacity and device granularity.

```
DECLARE mirr_create_struct STRUCTURE(
   cmd                        BYTE,
   sec_name(6)                BYTE);
```

or

```
typedef struct {
   UINT_8                  cmd;
   UINT_8                  sec_name[6];
} MIRR_CREATE_STRUCT;
```

Where:

cmd Has a value of 1.

sec_name The DUIB name of the secondary hard disk.

Subfunction 2 enables mirroring with resynchronization. Use this function only after a mirror set has been created or if the mirror set has rolled over. Specify the direction of the resynchronization in the structure. The device driver ensures that the destination hard disk of the resynchronization operation is not the good hard disk. The subfunction returns immediately; resynchronization is performed in the background, one track at a time. I/O System read and write operations are allowed on the mirror set while the resynchronization is in progress. If a write is directed at the disk being resynchronized, the device driver delays the write operation until the resynchronization is complete. The device driver signals resynchronization completion or abort using the Request Mirror Event Notification subfunction.

```
DECLARE mirr_resync_struct STRUCTURE (
   cmd                    BYTE,
   resync_dir             BYTE);
```

or

```
typedef struct {
   UINT_8              cmd;
   UINT_8              resync_dir;
} MIRR_RESYNC_STRUCT;
```

Where:

`cmd` Has a value of 2.

`resync_dir`

Has one of these valid values:

Value	**Meaning**
1	Data is copied from the primary to the secondary.
2	Data is copied from the secondary to the primary.

Subfunction 3 disables the mirroring operation and is valid only after the mirror set has been created. If a resynchronization is in progress, the resynchronization is aborted. All pending I/O operations on the mirror set are completed before mirroring is disabled. The call returns an error if the mirror set does not exist.

```
DECLARE mirr_disable_struct STRUCTURE (
   cmd                       BYTE);
```

or

```
typedef struct {
   UINT_8              cmd;
} MIRR_DISABLE_STRUCT;
```

Where:

`cmd` Has a value of 3.

Subfunction 4 requests the device driver to notify the task of a mirror event and provides a data mailbox to the device driver for reporting the event. Once a message has been sent to the mailbox, the application must issue a new request for mirror event notification. The device driver saves one event per mirror set if a request for event notification for the mirror set has not been issued.

```
DECLARE mirr_notify_struct STRUCTURE (
   cmd                         BYTE,
   reserved                    BYTE,
   mailbox       TOKEN);
```

or

```
typedef struct {
   UINT_8                 cmd;
   UINT_8                 reserved;
   SELECTOR               mailbox;
} MIRR_NOTIFY_STRUCT;
```

Where:

`cmd` Has a value of 4.

`mailbox` The token for a data mailbox, not a message mailbox. The device driver sends a 1 byte message to the mailbox after a mirror event has occurred. These are valid event codes:

Value	**Meaning**
1	Resync complete
2	Resync aborted
3	Rollover

Subfunction 5 gets the status of the mirror set and returns it in this structure:

```
DECLARE mirr_stat_struct STRUCTURE (
   cmd                     BYTE,
   mirr_set_state          BYTE,
   err_flg                 BYTE,
   last_scsi_err(3)        BYTE,
   last_pci_error          BYTE,
   read_policy             BYTE,
   primary_unit(16)        BYTE,
   sec_unit(16)            BYTE,
   src_good_unit(16)       BYTE,
   last_err_unit(16)       BYTE,
   last_rmx_err            WORD_16,
   last_err_addr           WORD_32,
   resync_percent          BYTE);
```

or

```
typedef struct {
   UINT_8                  cmd;
   UINT_8                  mirr_set_state;
   UINT_8                  err_flg;
   UINT_8                  last_scsi_err[3];
   UINT_8                  last_pci_error;
   UINT_8                  read_policy;
   UINT_8                  primary_unit[16];
   UINT_8                  sec_unit[16];
   UINT_8                  src_good_unit[16];
   UINT_8                  last_err_unit[16];
   UINT_16                 last_rmx_err;
   UINT_32                 last_err_addr;
   UINT_8                  resync_percent;
} MIRR_STAT_STRUCT;
```

Where:

`cmd` Has a value of 5.

`mirr_set_state`

The state of the mirror set. These values are possible:

Value	**Meaning**
0	Not part of a mirror set
1	Mirror set created
2	Mirroring enabled
3	Resync in progress
4	Rollover

`err_flg` Indicates whether the error status returned is valid.

Value	**Meaning**
0FFH	Valid
0	Invalid

`last_scsi_err`

Contains 3 bytes of SCSI error status of the last error that occurred on the mirror set.

See also: Errors, in your SCSI documentation

`last_pci_err`

The PCI error status of the last error that occurred on the mirror set.

See also: Error messages, *How to Use the Peripheral Controller Interface Server*

`read_policy`

Indicates:

Value	**Meaning**
1	Reads are performed from the primary.
2	Reads are performed from the secondary.
3	Reads are performed alternately.

`primary_unit`

The DUIB name of the primary unit.

`sec_unit` The DUIB name of the secondary unit.

`src_good_unit`

The DUIB name of the source unit if a resync is in progress, or of the good unit if the mirror set has rolled over.

`last_err_unit`

The DUIB name of the unit on which an error occurred.

`last_rmx_err`

The iRMX condition code of the last error that occurred on the mirror set.

`last_err_addr`
The block address of the last error that occurred on the mirror set.

`resync_percent`
The amount of resynchronization that is complete, shown in a percentage value. For example, 25% complete means that there is 75% more to go before the resynchronization is complete.

Subfunction 6 returns the status of a hard disk that is controlled by a device driver. This call may be directed at any attached disk hard disk controlled by the device driver. The hard disk need not be part of any mirror set.

```
DECLARE mirr_attach_struct STRUCTURE (
   cmd                          BYTE,
   attach_status                BYTE,
   other_name(16)               BYTE,
   incarnation                  WORD_32,
   good_flg                     BYTE);
```

or

```
typedef struct {
   UINT_8                  cmd;
   UINT_8                  attach_status;
   UINT_8                  other_name[16];
   UINT_32                 incarnation;
   UINT_8                  good_flg;
} MIRR_ATTACH_STRUCT;
```

Where:

`cmd` Has a value of 6.

`attach_status`
The mirroring status when the device is attached.

Value	**Meaning**
0	No mirroring information is available.
1	Mirror set is valid.
2	Mirror set is not valid.
3	This device is not the primary unit.
4	An error occurred on the secondary during attach.
5	The secondary is inconsistent.

`other_name`
The DUIB name of the other unit of the mirror set.

`incarnation`
The pattern that is written on the disks when the mirror set was detached, to uniquely identify the correct instance of a mirror set.

good_flg Specifies whether this disk was marked good when it was detached:

0AAH	Disk was good
055H	Disk was not good

Subfunction 7 dynamically changes some parameters associated with a mirror set.

```
DECLARE mirr_opt_struct STRUCTURE(
   cmd                     BYTE,
   read_policy             BYTE);
```

or

```
typedef struct {
   UINT_8                  cmd;
   UINT_8                  read_policy;
} MIRR_OPT_STRUCT;
```

Where:

cmd Has a value of 7.

read_policy
Indicates:

Value	**Meaning**
1	Reads are performed from the primary.
2	Reads are performed from the secondary.
3	Reads are performed alternately.

The read policies are in effect only when mirroring is enabled. At other states, the reads are performed from one hard disk. During resynchronization, that hard disk is the source hard disk. During rollover, that hard disk is the surviving hard disk.

Get Device Free Space Data (Function Code 20)

This function returns information about the free space available on the specified device.

Call **a_special** with a device or file connection, function set to 20, and ioparm_ptr pointing to a structure of this form. Set resp_mbox to null.

```
DECLARE device_free_struct STRUCTURE(
   sector_size                WORD_16
   device_size                WORD_32
   bytes_free                 WORD_32
   files_free                 WORD_32
   reserved(2)                WORD_32);
```

or

```
typedef struct {
   UINT_16                sector_size;
   UINT_32                device_size;
   UINT_32                bytes_free;
   UINT_32                files_free
   UINT_32                reserved[2];
}DEVICE_FREE_STRUCT;
```

Where:

`sector_size`
: The minimum I/O transfer size for the device.

`device_size`
: The total number of bytes available on the device (when empty).

`bytes_free`
: The number of bytes available in the device file system.

`files_free`
: The number of files available in the device filesystem. A returned value of `0FFFFFFFFH` indicates that this field does not apply; the number of files in the file system is limited only by the space on the device (DOS and EDOS file drivers).

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_BUFFERED_CONN	0036H	The connection parameter was opened with an EIOS call.
E_EXIST	0006H	At least one of these is true: • One or more of the `connection`, `resp_mbox`, `mailbox`, `object`, or `semaphore` parameters or fields is not a token for an existing object. • The connection is being deleted. • The connection for a remote driver is no longer active. • The mirror set secondary hard disk's DUIB is not configured.
E_IO	002BH	An error occurred while initializing the mirror set's secondary hard disk.
E_IFDR	002FH	The spec_func requested is not valid for the file type 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.

<table>
<tr><td>E_PARAM</td><td>8004H</td><td>At least one of these is true:
• The spec_func parameter was greater than 20 but less than 32K.
• The entire user-provided structure does not have the correct read/write accesses as described below:
<table>
<tr><th>Not Readable</th><th>Not Writable</th></tr>
<tr><td>format track</td><td>get disk/tape date</td></tr>
<tr><td>notify</td><td>get terminal data</td></tr>
<tr><td>set terminal data</td><td>get bad track info</td></tr>
<tr><td>set signal</td><td></td></tr>
<tr><td>set bad track info</td><td></td></tr>
</table>
• The ioparm_ptr pointer is invalid.
• The auxiliary pointer is invalid.
• The mirror set's secondary hard disk does not have the same device capacity or device granularity as the primary hard disk.
• The mirror disk resynchronization direction value is out of range, or the resync destination unit is the same as the good unit.
• The mirror disk read policy value is out of range.</td></tr>
<tr><td>E_STATE</td><td>0007H</td><td>One of these is true:
• The mirror set has not been created.
• Resynchronization is already in progress.
• Mirroring is already enabled.
• The mirror set already exists.</td></tr>
<tr><td>E_SUPPORT</td><td>0023H</td><td>The specified connection was not created by this job.</td></tr>
<tr><td>E_TYPE</td><td>8002H</td><td>One or more of the connection, resp_mbox, mailbox, or semaphore parameters or fields is a token for an existing object of the wrong type.</td></tr>
</table>

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
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. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*
E_IO_ALT_ASSIGNED	0058H	An alternate has already been assigned for a bad track.
E_IO_MEM	0042H	The memory pool of the BIOS on the server does not have enough memory for the system call to finish.
E_IO_NO_SPARES	0057H	No more alternate tracks are available.
E_NOT_DEVICE_CONN	0033H	The function code is 2 (notify), but the specified connection is not a device connection. This applies only to named and physical files.
E_SPACE	0029H	This call attempted to format a track of a physical file beyond the end of the volume, or of a RAM disk other than track 0.
E_STREAM_SPECIAL	003CH	This applies only to stream files. One of these is true: • This is a query request, but another query is already queued. • This is a satisfy request, but either a query request is queued, or no requests are queued.

a_truncate

Truncates a named (including DOS and remote) data file at the current setting of the file pointer, freeing all allocated space beyond the pointer. Directory files cannot be truncated; an attempt returns E_SUPPORT.

Syntax, PL/M and C

```
CALL rq$a$truncate (connection, resp_mbox, except_ptr);
```

```
rq_a_truncate (connection, resp_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection

A token for an open connection to the file being truncated.

resp_mbox

The mailbox that receives a token for an IORS. A null selector means that you do not want to receive an IORS.

except_ptr

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

Use **a_seek** to position the pointer before calling **a_truncate**. If the file pointer is at or beyond the EOF, no operation is performed.

For iRMX files, the designated file connection must be open for writing and the user must have update access to the file. For DOS files, the World user must have write access to the file.

See also: **a_change_access**, EIOS call **s_change_access**

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

File pointers for connections to the file are not adjusted by the truncation operation, and may be invalid or beyond the new EOF. If you then invoke **a_write**, the BIOS extends the file to accommodate the writing operation. The file will contain random data between the old EOF and the pointer to where the write begins.

You can also invoke **a_read** with the file pointer beyond the EOF, but the BIOS will return the `actual` field of the IORS as 0, signifying the EOF.

See also: **a_seek**, **a_write**, **a_read**

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_BUFFERED_CONN	0036H	The connection was produced by the EIOS. You cannot use it with BIOS calls.
E_EXIST	0006H	At least one of these is true: • One or more of the `connection` or `resp_mbox` parameters is not a token for an existing object. • The connection is being deleted. • The connection for a remote driver is no longer active.
E_IFDR	002FH	This system call applies only to named files (including DOS and remote), but the connection parameter specified some other type of file.
E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its object limit. • 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_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SUPPORT	0023H	At least one of these is true: • The specified connection was not created by this job. • The file is a directory file.

E_TYPE	8002H	Either the connection parameter is not a token for a connection object, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
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. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*

a_update

Updates all physical, named, remote, and DOS data or directory files on a device by writing all partial sectors that remain buffered in the BIOS.

Syntax, PL/M and C

```
CALL rq$a$update (connection, resp_mbox, except_ptr);

rq_a_update (connection, resp_mbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection

A token for a file or device connection.

resp_mbox

The mailbox that receives a token for an IORS. A null selector means that you do not want to receive an IORS.

except_ptr

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

When the BIOS performs an **a_write** operation, it writes only entire sectors. If a partial sector remains to be written, the BIOS usually leaves the data in an output buffer. The next time **a_write** is called, the BIOS combines the leftover data in the buffer with the data in the new request and again begins writing entire sectors. **A_update** forces the BIOS to finish the writing operation for a device by writing all buffers pertaining to files on a particular device. This ensures that files on removable volumes such as diskettes are updated before removal.

A_update has no effect on buffers that the EIOS manages.

See also: **a_write**

Three different events can cause the BIOS to update a device:

- Calling **a_update**
- Fixed updating
- Timeout updating

Fixed updating and timeout updating are triggered by the passing of possibly different amounts of time.

See also: Fixed updating and timeout updating, *Introducing the iRMX Operating Systems*

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	At least one of these is true: • One or more of the connection or resp_mbox parameters is not a token for an existing object. • The connection is being deleted. • 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 these is true: • The calling task's job has already reached its object limit. • 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_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	Either the connection parameter is not a token for a connection object, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
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. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*
E_NOT_FILE_CONN	0032H	The connection parameter is a device connection, not a file connection.

wait_io

Returns the concurrent condition code for the prior call to the calling task. Use with any type of file.

Syntax, PL/M and C

```
actual = rq$wait$io (connection, resp_mbox, time_limit,
      except_ptr);
```

```
actual = rq_wait_io (connection, resp_mbox, time_limit,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
actual	WORD_32	NATIVE_WORD
connection	SELECTOR	SELECTOR
resp_mbox	SELECTOR	SELECTOR
time_limit	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

actual

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**, or if the exception value is other than E_OK.

Parameters

connection

A token for the connection specified in the prior asynchronous system call.

resp_mbox

A token for the response mailbox specified in the prior asynchronous system call.

time_limit

Specifies how long the task calling **wait_io** is willing to wait for the IORS to arrive at the response mailbox.

Value	Meaning
0	Do not wait.
1-65534	Wait this number of Nucleus clock intervals.
65535	Wait forever.

`except_ptr`

A pointer to a variable declared by the application where either the concurrent condition code for the prior asynchronous system call or the sequential condition code for **wait_io** returns.

Additional Information

Use **wait_io** following a call to **a_read**, **a_write**, or **a_seek**. If applicable, **wait_io** also returns the number of bytes read or written.

There are two ways in which a task calling **a_read**, **a_write**, or **a_seek** can receive the information in the IORS. One way is for the task to wait at the response mailbox, receive the IORS there, extract the information, and delete the segment.

See also: IORS, Chapter 1,
Accessing the IORS, *Programming Techniques*

The other way is to call **wait_io**. After the asynchronous portion of the previous I/O call has been completed, **wait_io** returns the result of that call as follows:

- To `actual`, the number of bytes read for **a_read** or written for **a_write**. If the previous call was to **a_seek**, the value in `actual` is undefined.
- To the location pointed to by the `except_ptr` parameter, the concurrent condition code from the previous I/O call or the sequential condition code from the **wait_io** call. If either of these is not E_OK, the previous call's concurrent code returns; if both of the condition codes are not E_OK, the **wait_io** sequential code returns.

Wait_io does not return E_LIMIT, E_MEM, and E_SUPPORT, so if one of these returns, it came from the previous I/O call. If the previous I/O call caused an E_IO condition code, **wait_io** does not return this code. In this case only, **wait_io** returns these condition codes for that call (see descriptions under Condition Codes):

Value	**Mnemonic**
50H	E_IO_UNCLASS
51H	E_IO_SOFT
52H	E_IO_HARD
53H	E_IO_OPRINT
54H	E_IO_WRPROT
55H	E_IO_NO_DATA
56H	E_IO_MODE
57H	E_IO_N_OSPARES
58H	E_IO_ALT_ASSIGNED

When **wait_io** is used with EIOS calls, and an exception code is returned, the actual field in the IORS is invalid.

For applications using **wait_io**, tasks do not have to deal with and delete the IORS. The BIOS maintains its own supply of IORSs that can be used repeatedly. This enhances performance because the BIOS does not have to create a segment every time an IORS is needed. This provides a significant advantage with the frequently-used calls **a_read**, **a_write**, and **a_seek**.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	At least one of these is true: • One or more of the `connection` or `resp_mbox` parameters is not a token for an existing object. • The specified connection and/or response mailbox 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 these 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 BIOS tried to perform the operation the configured number of times. All attempts failed. Another retry probably won't be successful.
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 these is true: • The calling task was not willing to wait, and there was no IORS at the specified mailbox. • The specified waiting period elapsed before the response mailbox received an IORS.
E_TYPE	8002H	At least one of these is true: • The `connection` parameter is not a token for a connection object. • The `resp_mbox` parameter is not a mailbox token. • The object received at the response mailbox is not a segment or is a segment that is not an IORS.

wait_iors

Waits for an IORS and copies it to a user-provided buffer.

Syntax, PL/M and C

```
CALL rq$wait$iors (conn, mbox, time, iors_ptr, except_ptr);
```

```
rq_wait_iors (conn, mbox, time, iors_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`conn`	SELECTOR	SELECTOR
`mbox`	SELECTOR	SELECTOR
`time`	WORD_16	UINT_16
`iors_ptr`	POINTER	void *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`conn` A token for the connection specified in the prior asynchronous system call.

`mbox` A token for the response mailbox specified in the prior asynchronous system call.

`time` Specifies how long the calling task is willing to wait.

Value	Meaning
0	Do not wait.
1-65534	Wait this number of Nucleus clock intervals.
65535	Wait forever.

`iors_ptr`
A pointer to a buffer declared by the application where the IORS will be placed.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Wait_iors can be called after any BIOS asynchronous system call. It returns an IORS in the preallocated buffer provided in the call. Internally, the IORS segment is copied to the buffer and then either recycled (for read/write/seek) or deleted. This call simplifies I/O for a flat model application since the application cannot access an IORS segment directly without a far pointer. **Wait_iors** can be used by both segmented and flat model applications.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	The `conn` parameter is not a connection token or the `mbox` parameter is not a mailbox token.
E_PARAM	8004H	The `iors_ptr` parameter is not writable.
E_TIME	0001H	No IORS was received in the time specified or the caller was not willing to wait and there was no IORS at the mailbox.

a_write

Writes data from the calling task's buffer to a connected physical, stream, named, remote, or DOS file. You cannot write to directory files; an attempt returns E_SUPPORT.

⚠ **CAUTION**

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

Syntax, PL/M and C

```
CALL rq$a$write (connection, buff_ptr, count, resp_mbox,
      except_ptr);
```

```
rq_a_write (connection, buff_ptr, count, resp_mbox,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
buff_ptr	POINTER	UINT_8 far *
count	WORD_32	NATIVE_WORD
resp_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

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

The number of bytes to be written.

resp_mbox

The mailbox that receives a token for an IORS. A null selector means that you do not want to receive an IORS.

except_ptr

A pointer to a variable declared by the application where the sequential part of the call returns a condition code.

Additional Information

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

See also: **a_change_access**, **s_change_access**

A_write starts writing 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. It may be more efficient to start writes on device block boundaries and write an integral number of device blocks.

Segments have a maximum length of 4 Gbytes, and data transfers of this size can be requested.

For named files, use **a_seek** to position the file pointer beyond the EOF. If you then invoke **a_write**, the BIOS extends the file to accommodate the writing operation. The file will contain random data between the old EOF and the pointer to where the write begins.

Condition Codes

Sequential Condition Codes: returned immediately to except_ptr

E_OK	0000H	No exceptional conditions occurred.
E_BAD_BUFF	8023H	At least one of these 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 was opened with an EIOS call. You cannot use it with **a_read**.
E_EXIST	0006H	At least one of these is true: • One or more of the `connection` or `resp_mbox` parameters is not a token for an existing object. • 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	At least one of these is true: • The specified connection was not created by this job. • The file is a directory file.
E_TYPE	8002H	Either the connection parameter is not a token for a connection object, or the resp_mbox parameter is not a mailbox token.

Concurrent Condition Codes: returned asynchronously to resp_mbox

E_OK	0000H	No exceptional conditions occurred.
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 these is true: • The specified connection was closed before the write operation could be performed. • The specified file is a stream file, and all other connections are also requesting to write to the file.
E_FNODE_LIMIT	003FH	The file cannot be created or extended to this size because it has reached the maximum number of volume blocks. See also: File driver limitations, *System Concepts* manual
E_FRAGMENTATION	0030H	The disk is too fragmented to extend the file. 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. See also: IORS, Chapter 1, Accessing the IORS, *Programming Techniques*

E_SPACE	0029H	At least one of these is true: • The volume is full. • The operation attempted to write beyond the end of the device. This applies only to physical files.
E_SUPPORT	0023H	If carried out, the write operation would extend the file, but the BIOS is not configured to allow file extension.

❑ ❑ ❑

Extended I/O System Calls 4

s_attach_file

Creates a connection to an existing name DOS, remote, physical, or stream file.

Syntax, PL/M and C

```
connection = rq$s$attach$file (path_ptr, except_ptr);

connection = rq_s_attach_file (path_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
path_ptr	POINTER	STRING far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

connection

The token for the new connection to the file.

Parameters

path_ptr

A pointer to a STRING containing the pathname of the file to be attached.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

For a named file (including DOS), the EIOS computes access rights for the connection, which are based on the file's access list and the user IDs in the default user object of the calling task's job. If the file's access list enables no access to the users listed in the default user object, the call creates the connection, but enables no access.

See also: Access rights, *System Concepts*

The iRMX-NET remote file's access rights are checked during operations on the connection. This won't affect your programs if you do this:

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

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_ALREADY_ATTACHED	0038H	The EIOS cannot attach the device containing the file because the BIOS has done so.
E_CONTEXT	0005H	The calling task's job is not an I/O job.
E_DEV_DETACHING	0039H	The device containing the specified file is being detached.
E_DEVFD	0022H	The EIOS attempted to physically attach a device that had formerly been only logically attached. It found that the device and the device driver specified in the logical attachment were incompatible.
E_EXIST	0006H	The connection parameter references a file on an invalid device. The BIOS generates this code.
E_FACCESS	0026H	The default user object is not allowed access to the 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 path_ptr parameter specifies a data file as a directory.
E_ILLVOL	002DH	The EIOS attempted to physically attach a device that had formerly been only logically attached. It found that the volume does not contain named files. The named file driver was requested during logical attachment.
E_INVALID_FNODE	003DH	The fnode for the specified file is invalid. The file cannot be accessed; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*

E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_MEM	0042H	The BIOS job did not have enough memory to perform the requested function.
E_IO_OPRINT	0053H	The device was off-line. Operator intervention is required.
E_IO_SOFT	0051H	A soft I/O error occurred. The BIOS tried to perform the operation a number of times and failed. The number of retries is a configuration parameter. Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_LIMIT	0004H	At least one of these is true: • The calling task reached the object limit. • DOS has run out of file handles. • The user object or the calling task's job is involved in 255 I/O operations. • The calling task's job is not an I/O job. • Processing this call would deplete the remote server's resources.
E_LOG_NAME_NEXIST	0045H	The specified path contains a logical name, but the call was unable to find this name in the object directories of the calling task's local job, the global job, or the root job.
E_LOG_NAME_SYNTAX	0040H	The specified logical name contains at least one of these syntax errors: • DOS has run out of file handles. • The logical name is missing matching colons. • The specified path contains a logical name that exceeds 12 characters, has no characters, or contains invalid characters.
E_MEDIA	0044H	The device containing the specified file is off-line.

E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E_NAME_NEXIST	0049H	The user object does not represent a verified user or is not properly defined in the remote server's UDF. Only dynamic logon creates verified users.
E_NOPREFIX	8022H	The default prefix for the calling task's job is undefined, or is not a valid device or file connection.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NOT_LOG_NAME	8040H	The specified path contains a logical name that represents an object that is not a device connection or a file connection.
E_NOUSER	8021H	The calling task's job does not have a default user, or is not a user object.
E_PARAM	8004H	The EIOS attempted to physically attach a device that had formerly been only logically attached. The logical attachment referred to a file driver (named, physical, or stream) that is not configured into your system. See also: For ICU-configurable systems, DFD parameter, *ICU User's Guide and Quick Reference*
E_PASSWORD_MISMATCH	004BH	The password of the default user object does not match the password of the corresponding user defined on the remote server.
E_PATHNAME_SYNTAX	003EH	The specified pathname contains invalid characters.
E_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF. The server's UDF must have World read permission.

s_catalog_connection

Creates a logical name for a connection by cataloging the connection in the object directory of a specific job.

Syntax, PL/M and C

```
CALL rq$s$catalog$connection (job, connection, log_name_ptr,
      except_ptr);
```

```
rq_s_catalog_connection (job, connection, log_name_ptr,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
job	SELECTOR	SELECTOR
connection	SELECTOR	SELECTOR
log_name_ptr	POINTER	STRING far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

`job` A token for the job in whose object directory the logical name is cataloged. If a null selector, the EIOS catalogs the connection in the object directory of the calling task's job.

`connection`
A token for the connection to be assigned the logical name. If a null selector, the EIOS looks up the name in the object directory of the calling task's job.

`log_name_ptr`
A pointer to a STRING of 12 or fewer characters, possibly delimited with colons, containing the logical name. The OS removes the colons so that a logical name with colons is the same as one without; *:F0:* is the same as *F0*. Colons do not count in the length of the name. To use this logical name in other EIOS calls, use colons.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The EIOS converts the characters in the `log_name_ptr` STRING to uppercase and catalogs the connection in the object directory of the specified job. Two situations affect the outcome of this system call:

- If the job's object directory contains the logical name, the new connection replaces the existing object in the directory.
- If the `connection` parameter is a null selector, the system copies the logical name and its definition from the calling task's job into the object directory of the specified job.

Do not delete a task while it is using this system call.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The job in which the task is attempting to catalog the connection has an object directory that is 0 bytes long.
E_EXIST	0006H	The job or connection parameter is not a token for an existing object.
E_LIMIT	0004H	At least one of these is true: • The object directory for the specified job is full. • The calling task's job is not an I/O job.
E_LOG_NAME_NEXIST	0045H	The EIOS was unable to find the specified logical name in the object directory of the calling task's job.
E_LOG_NAME_SYNTAX	0040H	The specified logical name contains at least one of these syntax errors: • The logical name was missing matching colons. • The specified path contains a logical name that exceeds 12 characters, has no characters, or contains invalid characters.
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_NOT_CONNECTION	8042H	The connection parameter is not a connection object token.
E_TYPE	8002H	The job parameter is a not a job token.

s_change_access

Changes the access list for a named file (including remote and DOS). This system call can be used for data or directory files, including those created by the BIOS.

Syntax, PL/M and C

```
CALL rq$s$change$access (path_ptr, id, access, except_ptr);

rq_s_change_access (path_ptr, id, access, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
path_ptr	POINTER	STRING far *
id	WORD_16	UINT_16
access	BYTE	UINT_8
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

path_ptr

A pointer to a STRING containing a path to the file whose access is changed.

id

The ID of the user whose access to the file is changed, not necessarily the owner's ID. If the file's access list contains the ID, the EIOS changes the ID's current access. If not, the EIOS adds the ID to the file's access list, unless the list is full (contains three entries). For DOS files, no IDs can be added or deleted since the user is World. For NFS files, user IDs may be mapped differently between different OSs.

access

Defines the new access rights to be assigned to the specified user. If 0, the EIOS removes the specified ID (for iRMX files only) from the access list. If not 0, the meaning of the various bit settings vary if the file is a data file or a directory file. The following tables show the access rights for data and directory files. Setting a bit to 1 enables access, 0 denies access. For NFS files, access rights may be mapped differently between different OSs.

See also: Accessing NFS Files, Chapter 17, *System Concepts*

Bits	**Data File Access Rights**
7-4	Reserved. Set to 0.
3	Update: Permission to write over any information in the file using **s_write_move** or **a_write**, and permission to truncate the file using **s_truncate_file** or **a_truncate**. Does not include permission to add information to the end of the file. Set this bit to the same value as bit 2 (Append) for remote files.
2	Append: Permission to write information at the end of the file using **s_write_move** or **a_write**. Does not include permission to write over information in the file or permission to truncate the file. Set this bit to the same value as bit 3 (Update) for remote files.
1	Read: Permission to read data from the file using **s_read_move** or **a_read**.
0	Delete: Permission to delete the entire file using **s_delete_file** or **a_delete_file**. Enable changing the filename using **s_rename_file** or **a_rename_file**. This bit is ignored for remote files.

Bits	**Directory File Access Rights**
7-4	Reserved. Set to 0.
3	Change entry: Permission to change the access list associated with a file contained in the directory using **a_change_access** or **s_change_access**. This does not include permission to add new entries or change the access list of the directory where the file is cataloged. This bit is ignored for remote directories.
2	Add entry: Permission to add files to the directory using **a_create_file**, **a_create_directory**, **a_rename_file**, **s_create_file**, **s_create_directory**, or **s_rename_file**. This does not include permission to change existing entries.
1	List: Permission to read information from the directory using **a_read**, **a_get_directory_entry**, or **s_read_move**.
0	Delete: Permission to delete the directory using **a_delete_file** or **s_delete_file**. Enable changing the directory name by using **a_rename_file** or **s_rename_file**. This bit is ignored for remote directories.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

To change the access rights associated with a file, one of the IDs in the job's default user object must be the owner of the file, have change-entry access to the parent directory of the file, or be the system manager.

See also: Owners, access rights, default user objects, *System Concepts*

The DOS World user always has read (list) access to DOS files and directories; write (delete, append, update, add-entry and change-entry) access is optional.

You cannot change the access rights of an iRMX-NET virtual root directory, because a virtual root directory has no assigned owner; an E_FACCESS condition code returns.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_ALREADY_ATTACHED	0038H	The EIOS cannot attach the device containing the file because the BIOS has done so.
E_CONTEXT	0005H	The calling task's job is not an I/O job.
E_DEV_DETACHING	0039H	The device containing the specified file is being detached.
E_DEVFD	0022H	The EIOS attempted to physically attach a device that had formerly been only logically attached. It found that the device and the device driver specified in the logical attachment were incompatible.
E_FACCESS	0026H	The job containing the calling task meets none of the prerequisites for using this system call. None of the IDs in the job's default user object is the owner of the file, nor does any have change-entry access to the file's parent directory.

E_FNEXIST	0021H	One of these is true: • A file in the specified path, or the target file itself, does not exist or is marked for deletion. • The physical device was not found. The device was specified by the original call to **a_physical_attach_device** and is indicated in this call by the `path_ptr` parameter. See also: BIOS call **a_physical_attach_device**
E_FTYPE	0027H	The path_ptr parameter specifies a data file as a directory.
E_IFDR	002FH	The file driver associated with this connection is the physical or stream file driver.
E_ILLVOL	002DH	The EIOS attempted to physically attach a device that had formerly been only logically attached. It found that the volume does not contain named files. The named file driver was requested during logical attachment.
E_INVALID_FNODE	003DH	The fnode for the specified file is invalid. The file cannot be accessed; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_OPRINT	0053H	The device was off-line. Operator intervention is required.
E_IO_SOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed. The number of retries is a configuration parameter. Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_IO_WRPROT	0054H	The volume is write-protected.

E_IO_MEM	0042H	The BIOS job does not currently have a block of memory large enough to allow this system call to complete.
E_LIMIT	0004H	At least one of these is true: • The user object or the calling task's job is involved in 255 I/O operations. • The calling task's job is not an I/O job. • Processing this call would deplete the remote server's resources.
E_LOG_NAME_NEXIST	0045H	The specified path contains a logical name, but the call was unable to find this name in the object directories of the calling task's local job, the global job, or the root job.
E_LOG_NAME_SYNTAX	0040H	The specified logical name contains at least one of these syntax errors: • The logical name was missing matching colons. • The specified path contains a logical name that exceeds 12 characters, has no characters, or contains invalid characters.
E_MEDIA	0044H	The device containing the specified file is off-line.
E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E_NAME_NEXIST	0049H	The user object does not represent a verified user or is not properly defined in the remote server's UDF. Only dynamic logon creates verified users.
E_NOPREFIX	8022H	The default prefix for the calling task's job is undefined, or is not a valid device or file connection.
E_NOT_FILE_CONN	0032H	The path_ptr parameter specifies a path in which the prefix portion is not a file connection.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

E_NOT_LOG_NAME	8040H	The specified path contains a logical name that refers to an object that is not a device connection or a file connection.
E_NOUSER	8021H	The calling task's job does not have a default user, or is not a user object.
E_PARAM	8004H	The EIOS attempted to physically attach a device that had formerly been only logically attached. The logical attachment referred to a file driver (named, physical, or stream) that is not configured into the system. See also: For ICU-configurable systems, DFD parameter, *ICU User's Guide and Quick Reference*
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 specified pathname contains invalid characters.
E_SUPPORT	0023H	At least one of these is true: • The calling task attempted to change access for a file other than a named file. • The calling task attempted to add another user ID to the file's access list, but the list contains three entries. Delete an entry before adding another. • The connection specified in the call is not contained in the job making the call. • For NFS files, the group ID could not be changed. This occurs if the iRMX ID is not World or does not map to the user ID or group ID on the remote system.
E_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF. The server's UDF must have World read permission.

s_close

Closes an open file connection for any type of file. This system call cannot be used to close connections that were opened by the BIOS.

Syntax, PL/M and C

```
CALL rq$s$close (connection, except_ptr);
```

```
rq_s_close (connection, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection`

A token for an open file connection that was opened by **s_open**.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

S_close closes a connection using this protocol:

1. Wait until all currently running I/O operations for the file are completed.
2. Ensure that any information in a partially filled output buffer is written to the file.
3. Release any buffers associated with the file.
4. Close the connection to the file, deleting neither the file nor the connection.

The EIOS performs no access checking before closing the connection.

Do not delete a task while it is using this system call.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CANNOT_CLOSE	0041H	An error occurred while flushing data from EIOS buffers to an output device.
E_CONN_NOT_OPEN	0034H	One of these is true: • The connection is not open. • The connection was opened by **a_open** rather than **s_open**.
E_EXIST	0006H	The connection parameter is not a token for an existing object.
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_MODE	0056H	A tape drive attempted a read (write) before the previous write (read) completed.
E_IO_NO_DATA	0055H	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 I/O System tried to perform the operation a number of times and failed. The number of retries is a configuration parameter. Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_IO_WRPROT	0054H	The volume is write-protected.
E_LIMIT	0004H	At least one of these is true: • The calling task's job is not an I/O job. • The calling task's job is involved in 255 I/O operations.
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_NOT_CONNECTION	8042H	The connection parameter is not a connection object token.
E_SUPPORT	0023H	The specified connection was not created by a task in the calling task's job.

s_create_directory

Creates a new directory file and automatically adds a new entry to the parent directory. The new directory is compatible with those created by the BIOS. This system call cannot be used to obtain connections to existing directories.

Syntax, PL/M and C

```
connection = rq$s$create$directory (path_ptr, except_ptr);

connection = rq_s_create_directory (path_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection`	SELECTOR	SELECTOR
`path_ptr`	POINTER	STRING far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`connection`

A token that represents a connection to the new directory. Use this token as a parameter in other system calls that access the directory.

Parameters

`path_ptr`

A pointer to a STRING containing the pathname of the new directory.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

After creation, the new directory contains no entries. The first ID in the job's default user object becomes the owner of the directory. The default user object for the calling task's job must have add-entry access to the parent of the new directory.

See also: **s_change_access**, BIOS call **a_change_access**

The DOS World user always has read (list) access to DOS files and directories; write (delete, append, update, add-entry and change-entry) access is optional.

The calling task must use the `path_ptr` parameter to specify the location of the new directory within the named file structure (including remote and DOS files). The entry in the parent directory provides the owner of the new directory with full access to the new directory.

You cannot create a remote directory with an iRMX-NET virtual root directory as the parent, because a virtual root directory has no assigned owner and cannot be written to; an E_FACCESS condition code returns.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_ALREADY_ATTACHED	0038H	The EIOS cannot attach the device containing the file because the BIOS has done so.
E_CONTEXT	0005H	The calling task's job is not an I/O job.
E_DEV_DETACHING	0039H	The device containing the specified file is being detached.
E_DEVFD	0022H	The EIOS attempted to physically attach a device that had formerly been only logically attached. It found that the device and the device driver specified in the logical attachment were incompatible.
E_FACCESS	0026H	The user object associated with the calling task's job does not have add-entry access to the parent directory.
E_FEXIST	0020H	The file already exists.
E_FNEXIST	0021H	At least one of these is true: • A file in the specified path does not exist or is marked for deletion. • The specified device is not part of the current configuration.
E_FNODE_LIMIT	003FH	The volume contains the maximum number of files. No more fnodes are available for new files.
E_FTYPE	0027H	The path_ptr parameter specifies a data file as a directory.
E_ILLVOL	002DH	The EIOS attempted to physically attach a device that had formerly been only logically attached, and found that the volume does not contain named files. The named file driver was requested during logical attachment.

E_INVALID_FNODE	003DH	The fnode for a directory in the specified pathname is invalid. The file cannot be accessed; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_OPRINT	0053H	The device was off-line. Operator intervention is required.
E_IO_SOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed; The number of retries is a configuration parameter. Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_IO_WRPROT	0054H	The volume is write-protected.
E_IO_MEM	0042H	The BIOS job does not currently have a block of memory large enough to allow this system call to complete.
E_LIMIT	0004H	At least one of these is true: • The user object or the calling task's job is involved in 255 I/O operations. • The calling task's job is not an I/O job. • Processing this call would deplete the remote server's resources.
E_LOG_NAME_NEXIST	0045H	The specified path contains a logical name, but the call was unable to find this name in the object directories of the calling task's local job, the global job, or the root job.
E_LOG_NAME_SYNTAX	0040H	The specified logical name contains at least one of these syntax errors: • The logical name was missing matching colons. • The specified path contains a logical name that exceeds 12 characters, has a length of 0 characters, or contains invalid characters.

E_MEDIA	0044H	The device containing the specified file is off-line.
E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E_NAME_NEXIST	0049H	The user object does not represent a verified user or is not properly defined in the remote server's UDF. Only dynamic logon creates verified users.
E_NOPREFIX	8022H	The default prefix for the calling task's job is undefined, or is not a valid device or file connection.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NOT_LOG_NAME	8040H	The specified path contains a logical name that refers to an object that is not a device connection or a file connection.
E_NOUSER	8021H	The calling task's job does not have a default user, or is not a user object.
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_PARAM	8004H	The EIOS attempted to physically attach a device that had formerly been only logically attached. The logical attachment referred to a file driver that is not configured into your system. See also: For ICU-configurable systems, DFD parameter, *ICU User's Guide and Quick Reference*
E_PATHNAME_SYNTAX	003EH	The specified pathname contains invalid characters.
E_SUPPORT	0023H	You cannot create any directories on this volume.

E_SPACE	0029H	At least one of these is true: • The volume is full. • 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_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF. The server's UDF must have World read permission.

s_create_file

Creates a new physical, stream, or named data file (including DOS and remote), not a named directory file. The created file is compatible with files created by the BIOS.

Syntax, PL/M and C

```
connection = rq$s$create$file (path_ptr, except_ptr);

connection = rq_s_create_file (path_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection`	SELECTOR	SELECTOR
`path_ptr`	POINTER	STRING far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`connection`

The token that represents the connection to the new file.

Parameters

`path_ptr`

A pointer to a STRING that contains the pathname of the file to be created. This parameter also indicates what kind of file (stream, physical, or named data) to create.

See also: Named, remote, physical, and stream file paths, *System Concepts*

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If the file specified by the `path_ptr` parameter exists, the EIOS attempts to truncate the file to 0 length and return a connection to the empty file. The owner and the accessor list for the file remain unchanged. The call succeeds only if both of these are true:

- The file exists and all open connections to the file allow sharing with writers.
- For named files, an ID in the default user object of the calling task's job has update access to the existing file.

The DOS World user always has read (list) access to DOS files and directories; write (delete, append, update, add-entry and change-entry) access is optional.

To prevent the file from being truncated accidentally, use **s_attach_file**; if the call to **s_attach_file** returns a condition code indicating the file does not exist, use **s_create_file**.

See also: **s_attach_file**

You cannot create an iRMX-NET remote file with a virtual root directory as its parent because a virtual root directory has no owner and no write access; an E_FACCESS condition code returns.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_ALREADY_ATTACHED	0038H	The EIOS cannot attach the device containing the file because the BIOS has done so.
E_CONTEXT	0005H	The calling task's job is not an I/O job.
E_DEV_DETACHING	0039H	The device containing the specified file is being detached.
E_DEVFD	0022H	The EIOS attempted to physically attach a device that had formerly been only logically attached, and found that the device and the device driver specified in the logical attachment were incompatible.
E_FACCESS	0026H	At least one of these is true: • The default user object associated with the calling task's job does not have add-entry access to the parent directory. • The default user object associated with the calling task's job does not have update access to the existing file with the specified pathname.
E_FNEXIST	0021H	At least one of these is true: • A file in the specified path does not exist or is marked for deletion. • The specified physical device was not found.

E_FNODE_LIMIT	003FH	The file cannot be created or extended to this size because it has reached the maximum number of volume blocks. See also: File driver limitations, *System Concepts* manual
E_FTYPE	0027H	The path_ptr parameter specifies a data file as a directory.
E_ILLVOL	002DH	The EIOS attempted to physically attach a device that had formerly been only logically attached, and found that the volume does not contain named files. This prevented the call from completing physical attachment.
E_INVALID_FNODE	003DH	The fnode for a directory in the specified pathname is invalid. The file cannot be accessed; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_OPRINT	0053H	The device was off-line. Operator intervention is required.
E_IO_SOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed. The number of retries is a configuration parameter. Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_IO_WRPROT	0054H	The volume is write-protected.
E_IO_MEM	0042H	The BIOS job does not currently have a block of memory large enough to allow this system call to complete.

E_LIMIT	0004H	At least one of these is true: • The calling task has reached the object's limit. • The user object or the calling task's job is involved in 255 I/O operations. • The calling task's job is not an I/O job. • Processing this call would deplete the remote server's resources.
E_LOG_NAME_NEXIST	0045H	The specified path contains a logical name, but the call was unable to find this name in the object directories of the calling task's local job, the global job, or the root job.
E_LOG_NAME_SYNTAX	0040H	The specified logical name contains at least one of these syntax errors: • The logical name was missing matching colons. • The specified path contains a logical name that exceeds 12 characters, does not contain at least one character, or contains invalid characters.
E_MEDIA	0044H	The device containing the specified file is off-line. The media may be inserted incorrectly.
E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E_NAME_NEXIST	0049H	The user object does not represent a verified user or is not properly defined in the remote server's UDF. Only dynamic logon creates verified users.
E_NOPREFIX	8022H	The default prefix for the calling task's job is undefined, or is not a valid device or file connection.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NOT_LOG_NAME	8040H	The specified path contains a logical name that refers to an object that is not a device connection or a file connection.

E_NOUSER	8021H	The calling task's job does not have a default user object, or the object cataloged in *r?iouser* is not a user object.
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_PARAM	8004H	The EIOS attempted to physically attach a device that had formerly been only logically attached. The logical attachment referred to a file driver that is not configured into your system, so the physical attachment is not possible. See also: For ICU-configurable systems, DFD parameter, *ICU User's Guide and Quick Reference*
E_PATHNAME_SYNTAX	003EH	The specified pathname contains invalid characters.
E_SHARE	0028H	You are trying to create a file that exists. The EIOS must truncate the existing file to 0 length to do the create. Truncation failed for one or more of these reasons: • Another open connection does not allow sharing with writers. • The default user for the calling task's job does not have update access to the file.
E_SPACE	0029H	At least one of these is true: • The volume is full. • 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 BIOS configuration does not allow the truncation of an existing file to 0 length. See also: For ICU-configurable systems, ACE parameter, *ICU User's Guide and Quick Reference*

E_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF. The server's UDF must have World read permission.

create_io_job

Obsolete, but provided for compatibility. Creates an I/O job containing one task of up to 1 Mbyte. You can call **rq_create_io_job** only from another I/O job. This call is not supported for flat model applications.

See also: **rqe_create_io_job**

Syntax, PL/M and C

```
io_job = rq$create$io$job (pool_min, pool_max, except_handler,
      job_flags, task_priority, start_address, data_seg,
      stack_ptr, stack_size, task_flags, msg_mbox, except_ptr);
```

```
io_job = rq_create_io_job (pool_min, pool_max, except_handler,
      job_flags, task_priority, start_address, data_seg,
      stack_ptr, stack_size, task_flags, msg_mbox, except_ptr);
```

See also: **rqe_create_io_job**

rqe_create_io_job

Creates an I/O job containing one task. **Rqe_create_io_job** can be called only from another I/O job. This system call is not supported for flat model applications.

See also: Application Loader calls **rqe_a_load_io_job** and **rqe_s_load_io_job** for flat model applications

Syntax, PL/M and C

```
io_job = rqe$create$io$job (pool_min, pool_max, except_handler,
      job_flags, task_priority, start_address, data_seg,
      stack_ptr, stack_size, task_flags, msg_mbox, except_ptr);
```

```
io_job = rqe_create_io_job (pool_min, pool_max, except_handler,
      job_flags, task_priority, start_address, data_seg,
      stack_ptr, stack_size, task_flags, msg_mbox, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
io_job	SELECTOR	SELECTOR
pool_min	WORD_32	UINT_32
pool_max	WORD_32	UINT_32
except_handler	POINTER	EXCEPTION_STRUCT far *
job_flags	WORD_16	UINT_16
task_priority	BYTE	UINT_8
start_address	POINTER	void (far *) (void)
data_seg	SELECTOR	SELECTOR
stack_ptr	POINTER	UINT_16 far *
stack_size	WORD_32	NATIVE_WORD
task_flags	WORD_16	UINT_16
msg_mbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

io_job

The token that represents the newly created job; valid if an E_OK condition code returns.

Parameters

`pool_min`

Specifies the initial and minimum allowable size of the new job's memory pool in 16-byte paragraphs. The memory initially allocated is always contiguous. Additional memory is not necessarily contiguous. If the base of the `stack_ptr` parameter is 0, ensure that `pool_min` is no less than 32 plus the number of 16-byte paragraphs required to contain the stack. Otherwise, the E_PARAM condition code returns.

`pool_max`

Specifies the maximum allowable size of the new job's memory pool in 16-byte paragraphs, up to 4 Gbytes. If `pool_max` is less than `pool_min`, the E_PARAM condition code returns.

See also: Memory pools, *System Concepts*

`except_handler`

A pointer to a structure of this form:

```
DECLARE except_handler STRUCTURE(
   exception_handler_ptr  POINTER,
   exception_mode         BYTE);
```

or

```
typedef struct {
   void far *             exception_handler_ptr;
   UINT_8                 exception_mode;
} EXCEPTION_STRUCT;
```

Where:

`exception_handler_ptr`

Designates the new job's default exception handler by pointing to the first instruction of your exception handler. To designate the system default exception handler, use a null pointer.

`exception_mode`

Indicates when to pass control to the new task's exception handler as follows:

Value	Pass Control To Exception Handler
0	Never
1	On programmer errors only
2	On environmental conditions only
3	On all exceptional conditions

See also: Exception handlers, exception modes, *System Concepts* `job_flags`

Indicates whether to check the validity of objects used as parameters in system calls. If bit 1 is 0, the Nucleus will validate objects. All other bits must be set to 0.

See also: Nucleus call **rqe_create_job**

`task_priority`

Establishes the priority of the new job's initial task.

Value	Meaning
0	Priority equals the maximum priority of the EIOS initial job.
not 0	The specified priority value.

See also: For ICU-configurable systems, TP parameter, *ICU User's Guide and Quick Reference*

`start_address`

A pointer to the first instruction of the new job's initial task.

`data_seg`

Specifies:

Value	Meaning
Null selector	The new job's initial task uses no data segment, or it creates one for itself.
Valid selector	The base address of the data segment of the new job's initial task.

See also: Nucleus call **rqe_create_job**

`stack_ptr`

Specifies:

Value	Meaning
Null pointer	The Nucleus allocates a stack for the new job's initial task, of length specified by the stack_size parameter.
Valid pointer	References the base of the stack for the new job's initial task. Your program must allocate the stack during run-time unless it was allocated during ICU system configuration. The base of the stack must be an iRMX segment object.

See also: For ICU-configurable systems, SSA parameter, *ICU User's Guide and Quick Reference*

`stack_size`

Specifies the size in bytes of the stack for the new job's initial task. The minimum size is 400h. The Nucleus allocates enough additional bytes to make the stack occupy whole 16-byte paragraphs. Otherwise the stack is the size specified here.

See also: Stack, *Programming Techniques* `task_flags`

Indicates whether the new job's initial task uses floating-point instructions, and whether the initial task in the job should run immediately or wait until **start_io_job** is issued.

Bits	Value	Meaning
15-2	0	Reserved, set to 0
1	0	Task runs immediately
	1	Task waits
0	0	No floating-point instructions
	1	Floating-point instructions

`msg_mbox`

A token for a mailbox. When a task exits by invoking **exit_io_job**, the EIOS sends a message to this mailbox. This message can be received by the task's job. To send no message, assign a null selector to `msg_mbox`. The format of the message is as follows.

```
DECLARE message STRUCTURE(
   termination_code       WORD_16,
   user_fault_code        WORD_16,
   job_token              TOKEN,
   return_data_len        BYTE,
   return_data(*)         BYTE);
```

or

```
typedef struct {
   UINT_16                termination_code;
   UINT_16                user_fault_code;
   SELECTOR               job_token;
   UINT_8                 return_data_len;
   UINT_8                 return_data[_NUM_RETURN_DATA];
/* Adjust to fit return_data_len */
} MESSAGE_STRUCT;
```

Where:

`termination_code`

Indicates why an I/O job terminated as follows.

Value	Meaning
0	A task invoked **exit_io_job**, and no problem occurred. The job has not yet been deleted, and some of its tasks might still be ready.
1	The job was deleted because some task invoked **delete_job**.
Other	A task invoked **exit_io_job** because some problem occurred. The job has not yet been deleted and some of its tasks might still be ready.

`user_fault_code`

If `termination_code` is not 0 or 1, this field contains a user-encoded reason for task termination. The meaning of this field is provided by the terminating task, not by the OS.

`job_token` A token for the terminated job.

`return_data_len`

Specifies the length in bytes of the `return_data` parameter. The maximum length is 89 bytes.

`return_data`

An array that contains data specified by the terminating task when it invoked **exit_io_job**.

See also: **start_io_job**, **exit_io_job**,
Nucleus call delete_job

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

I/O jobs differ from other jobs in these ways:

- Some default job parameters are specified at system configuration time.
- **Create_io_job** provides default values for the `global_job`, `default_user`, and `default_prefix` attributes. These values are set during system configuration and are passed from parent job to child job.
- The EIOS can send a termination message to a mailbox whenever a task in the I/O job calls **exit_io_job**. Specify the mailbox by using the `msg_mbox` parameter.

Do not delete a task in an I/O job if the connection has not been deleted. If you do so, the connection will not be available to any other task.

For ICU-configurable systems, initial I/O jobs are set up at system configuration time.

See also: Parent job and child job, *System Concepts*,
I/O Jobs screen, *ICU User's Guide and Quick Reference*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The calling task's job is not an I/O job.
E_EXIST	0006H	At least one of these is true: • The token cataloged under the name RQGLOBAL (the global job) is not a token for an existing object. See also: Global object directory, *System Concepts* • The msg_mbox parameter is not a token for an existing mailbox.
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_NOUSER	8021H	The calling task's job does not have a default user, or the object cataloged under the logical name *r?iouser* is not a user object. See also: *r?iouser*, *System Concepts*

E_PARAM	8004H	At least one of these is true: • The `pool_min` parameter is less than 32, or greater than `pool_max`. • `Task_priority` is not 0 and is greater than (numerically less than) the maximum priority of the calling I/O job. • The exception_mode parameter is outside the range 0-3.
E_IO_JOB	0047H	The EIOS could not create an I/O job because the default directory size (DDS) configuration parameter is too small.

s_delete_connection

Deletes a file connection, but not a device connection. You must meet special requirements to use this system call with connections created by the BIOS.

See also: Connections, *System Concepts*

Syntax, PL/M and C

```
CALL rq$s$delete$connection (connection, except_ptr);
```

```
rq_s_delete_connection (connection, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`connection`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection`

A token for the file connection to be deleted.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If the connection is open, **s_delete_connection** automatically closes it before deleting it. The EIOS does not check access before deleting a connection.

If the file has been marked for deletion by a previous system call and there are no more connections to the file, **s_delete_connection** deletes the file.

Do not delete a task while it is using this system call.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The connection parameter is not a token for an existing object.
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_MODE	0056H	A tape drive attempted a read (write) before the previous write (read) completed.

E_IO_NO_DATA	0055H	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 I/O System tried to perform the operation a number of times and failed. The number of retries is a configuration parameter. Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_IO_WRPROT	0054H	The volume is write-protected.
E_LIMIT	0004H	At least one of these is true: • The associated job or the job's default user object is involved in 255 I/O operations. • The calling task's job is not an I/O job.
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_NOT_CONNECTION	8042H	The connection parameter is not a connection object token.
E_SUPPORT	0023H	The specified connection was not created by a task in this job.

s_delete_file

Marks and deletes a stream, named data (including DOS and remote), or named directory file, but not a physical file. This system call can also delete files created by the BIOS.

Syntax, PL/M and C

```
CALL rq$s$delete$file (path_ptr, except_ptr);
```

```
rq_s_delete_file (path_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`path_ptr`	POINTER	STRING far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`path_ptr`

A pointer to a STRING that specifies the path for the file to be deleted. The form of the path depends on the kind of file.

See also: Path syntax, *System Concepts*

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This system call marks the specified file for deletion, but the EIOS postpones deletion until these criteria are met:

- For stream and named data files, the deletion occurs as soon as no connections to the file remain. Use **s_delete_connection** to delete connections.
- For named directories, the directory must be empty, and no connections to the directory can remain. Otherwise, an E_DIR_NOT_EMPTY condition code returns.

For iRMX files, the caller must have delete access to the file; for DOS files, the caller must have write access to the file. The DOS World user always has read (list) access to DOS files and directories; write (delete, append, update, add-entry and change-entry) access is optional.

See also: **s_change_access**, BIOS call **a_change_access**

You cannot delete an iRMX-NET remote file that has a virtual root directory as its parent, because a virtual root directory has no assigned owner and no write access; an E_FACCESS condition code returns.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_ALREADY_ATTACHED	0038H	The specified device is attached.
E_CONTEXT	0005H	The calling task's job is not an I/O job.
E_DEV_DETACHING	0039H	The device containing the specified file is being detached.
E_DEVFD	0022H	The EIOS attempted to physically attach a device that had formerly been only logically attached, and found that the device and the device driver specified in the logical attachment were incompatible.
E_DIR_NOT_EMPTY	0031H	The calling task is attempting to delete a directory that is not empty.
E_FACCESS	0026H	At least one of these is true: • The default user object associated with the calling task's job does not have delete access to the specified file. • The call is attempting to delete a bit map file or the root directory.
E_FNEXIST	0021H	At least one of these is true: • A file in the specified path, or the target file itself, does not exist or is marked for deletion. • The physical device was not found. The device was specified by the original call to **a_physical_attach_device** and is indicated in this call by the path_ptr parameter.
E_FTYPE	0027H	A path component is not a directory file.

E_ILLVOL	002DH	The EIOS attempted to physically attach a device that had formerly been only logically attached, and found that the volume does not contain named files. The named file driver was requested during logical attachment.
E_IFDR	002FH	The specified file is a physical file.
E_INVALID_FNODE	003DH	The fnode associated with a file is marked not allocated, or the fnode number is out of range.
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_OPRINT	0053H	The device was off-line. Operator intervention is required.
E_IO_SOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed. The number of retries is a configuration parameter. Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_IO_WRPROT	0054H	The volume is write-protected.
E_IO_MEM	0042H	The BIOS job does not currently have a block of memory large enough to allow this system call to complete.
E_LIMIT	0004H	At least one of these is true: • Either the user object or the calling task's job is involved in 255 I/O operations. • The calling task's job is not an I/O job. • Processing this call would deplete the remote server's resources.
E_LOG_NAME_NEXIST	0045H	The specified path contains a logical name, but the call was unable to find this name in the object directories of the calling task's local job, global job, or the root job.

E_LOG_NAME_SYNTAX	0040H	The specified logical name contains at least one of these syntax errors: • The logical name was missing matching colons. • The specified path contains a logical name that exceeds 12 characters, contains no characters, or contains invalid characters.
E_MEDIA	0044H	The device containing the specified file is off-line.
E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E_NAME_NEXIST	0049H	The user object does not represent a verified user or is not properly defined in the remote server's UDF. Only dynamic logon creates verified users.
E_NOPREFIX	8022H	The default prefix for the calling task's job is undefined, or is not a valid device or file connection.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NOT_FILE_CONN	0032H	In the specified path, the subpath portion is null and the prefix portion is not a file connection.
E_NOT_LOG_NAME	8040H	The specified path contains a logical name that refers to an object that is not a device connection or a file connection.
E_NOUSER	8021H	The calling task's job does not have a default user object, or the object cataloged in *r?iouser* is not a user object.
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_PARAM	8004H	The EIOS attempted to physically attach a device that is logically attached. That logical attachment refers to a file driver that is not configured into your system. See also: For ICU-configurable systems, DFD parameter, *ICU User's Guide and Quick Reference*
E_PATHNAME_SYNTAX	003EH	The specified pathname contains invalid characters.
E_SUPPORT	0023H	The task is attempting to delete a physical file.
E_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF. The server's UDF must have World read permission.

exit_io_job

Sends a message to a previously designated mailbox and deletes the calling task.

Syntax, PL/M and C

```
CALL rq$exit$io$job (user_fault_code, return_data_ptr,
      except_ptr);
```

```
rq_exit_io_job (user_fault_code, return_data_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
user_fault_code	WORD_16	UINT_16
return_data_ptr	POINTER	UINT_8 far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

user_fault_code

The encoded reason for terminating the job. To terminate the job under normal circumstances, use 0. To terminate the job because of a problem, use a condition code that identifies the problem. The EIOS sends a structure containing this value to the mailbox specified in **create_io_job**.

Note

If you set this parameter to return any status code other than E_OK, Soft-Scope will report an error condition.

See also: **create_io_job**, UDI call **dq_exit**

return_data_ptr

A pointer to a buffer for return data provided by the calling task. This data returns to the message mailbox specified in **create_io_job**. A null pointer indicates no data returns. If the data is longer than 89 bytes, only the first 89 bytes are returned.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Use this system call to bring about an orderly deletion of an I/O job. It enables a task to delete itself and have the EIOS notify the parent job of the deletion.

➠ **Note**

Before an I/O job exits, it must uncatalog any objects it cataloged in other directories (global or root). Otherwise, the objects remain even though the connection is deleted. From then on, an error occurs if you use the connection or refer to the logical name.

When a task in an I/O job created by **create_io_job** invokes **exit_io_job**, this occurs:

1. The EIOS deletes the task (not the job) that invoked **exit_io_job**.
2. The EIOS sends a termination message to the mailbox specified in **create_io_job**.

Under certain circumstances, this system call does not delete the calling task; the EIOS returns control to the calling task and issues a condition code to indicate the nature of the problem:

- If **delete_task**, which the EIOS calls, returns an exceptional condition code to the EIOS
- If the calling task is an interrupt task

See also: **delete_task**, Nucleus call

The termination message is not sent in these circumstances:

- If the `msg_mbox` parameter of the **create_io_job** was set to a null selector
- If the mailbox specified in the `msg_mbox` parameter of **create_io_job** no longer exists

If the `return_data_ptr` is not a valid pointer or is not readable, no exception is returned to the task that calls **exit_io_job**. Instead, `return_data_ptr` is treated as a null pointer, and termination of the job continues. In this case, a termination message is still sent to the message mailbox, but the return data string is of zero length.

To detect this condition, your application should check for a zero-length termination message received at the mailbox specified in **create_io_job**. Such a message means one of two things:

- The exiting job sent a NULL pointer for `return_data_ptr`.
- The `return_data_ptr` was invalid or unreadable.

Condition Codes

E_CONTEXT	0005H	The task invoking **exit_io_job** is an interrupt task and cannot be deleted.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

s_get_connection_status

Provides status information about file and device connections created by the BIOS or the EIOS. You must meet special requirements to use this system call with connections created by the BIOS.

See also: Connections, *System Concepts*

Syntax, PL/M and C

```
CALL rq$s$get$connection$status (connection, info_ptr,
      except_ptr);
```

```
rq_s_get_connection_status (connection, info_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection`	SELECTOR	SELECTOR
`info_ptr`	POINTER	CONNECTION_INFO_STRUCT far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection`

A token for the connection whose status is sought.

`info_ptr`

A pointer to this structure:

```
DECLARE connection_info STRUCTURE(
   file_drivers          BYTE,
   flags                 BYTE,
   open_mode             BYTE,
   share_mode            BYTE,
   file_ptr              WORD_32,
   access                BYTE,
   number_buffers        BYTE,
   buffer_size           WORD_16,
   seek                  BYTE)
```

or

```
typedef struct {
   UINT_8                 file_driver;
   UINT_8                 flags;
   UINT_8                 open_mode;
   UINT_8                 share_mode;
   UINT_32                file_ptr;
   UINT_8                 access;
   UINT_8                 number_buffers;
   UINT_16                buffer_size;
   UINT_8                 seek;
} CONNECTION_INFO_STRUCT;
```

Where:

`file_drivers`

Identifies the type of file driver associated with the connection.

Value	**File Driver**
1	Physical
2	Stream
3	DOS
4	Named
5	Remote
6	EDOS
7-max	Loadable file drivers, including NFS. The ID for these drivers can vary; it is assigned in the order the driver is loaded.

`flags` Indicates the kind of connection this is.

Bits	**Meaning**
7-3	Reserved
2	If 1, the connection is a device connection
1	If 1, the connection can be opened
0	Reserved

`open_mode`

Indicates how the connection was opened. This applies only to file connections.

Value	**Meaning**
0	Closed
1	Open for reading only
2	Open for writing only
3	Open for both reading and writing

share_mode

Indicates who can share the device or file connection.

Value	Meaning
0	Cannot be shared
1	Share with readers only
2	Share with writers only
3	Share with all

file_ptr A 32-bit offset from the beginning of the file to where the next I/O operation is performed.

access The access rights for the connection. This applies only to connections for named files (including remote and DOS), and the interpretation of this field depends upon whether the file is a data file or a directory. Access is represented as a bit mask shown in these tables; access is granted if a bit is set to 1:

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

For remote iRMX-NET files, `access` is interpreted as follows:

Bits	Data File	Directory
7-4	Reserved	Reserved
3	Write	Ignored (set same as bit 2)
2	Write	Write (set same as bit 3)
1	Read	Display
0	Ignored	Ignored

For NFS files, access bits can be mapped differently for different OSs.

See also: Accessing NFS files, Chapter 17, *System Concepts*

number_buffers

The number of buffers used with this connection.

buffer_size

The size, in bytes, of each buffer used with the connection.

seek Indicates whether the **seek** function can be used with this connection. 0 means no; 0FFH means yes.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

For DOS files, the World user always has read (list) access to DOS files and directories; write (delete, append, update, add-entry and change-entry) access is optional. The EIOS does not check access rights before returning status information.

When the status of a file connection to an iRMX-NET 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. The remote file's access rights are checked during operations on the connection. This won't affect your programs if you follow these guidelines:

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

Do not delete a task while it is using this system call.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONN_NOT_OPEN	0034H	The connection was opened by **a_open**, not **s_open**.
E_EXIST	0006H	The connection parameter is not a token for an existing job.
E_IFDR	002FH	An invalid file driver request occurred.
E_LIMIT	0004H	At least one of these is true: • The calling task has reached its object limit. • Either the calling task's job, or the job's default user object, is involved in 255 I/O operations. • The calling task's job is not an I/O job.
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_NOT_CONNECTION	8042H	The connection parameter is not a connection object token.
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 a task in the calling task's job.

s_get_directory_entry

Returns a filename (or subdirectory) entry from a specified named or DOS directory.

Syntax, PL/M and C

```
CALL rq$s$get$directory$entry (dir_name_ptr, entry_num,
      name_ptr, except_ptr);
```

```
rq_s_get_directory_entry (dir_name_ptr, entry_num, name_ptr,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
dir_name_ptr	POINTER	STRING far *
entry_num	WORD_16	UINT_16
name_ptr	POINTER	STRING far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

`dir_name_ptr`

A pointer to a STRING containing the directory pathname. This pathname can be up to 255 characters long.

`entry_num`

The entry number of the desired filename. Entries are numbered sequentially starting from 0. The E_EMPTY_ENTRY condition code returns if there is no directory entry associated with the number.

`name_ptr`

A pointer to a STRING locating the entry name specified by `entry_num`. This name has a maximum length of 14 bytes. The filename is left-justified and padded with blanks to the right.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The calling task must have list (read) access to the designated directory. The DOS World user always has read (list) access to DOS files and directories; write (delete, append, update, add-entry and change-entry) access is optional.

The alternative to this call is to open and read a directory file.

S_get_directory_entry is not supported for iRMX-NET remote directories. Use BIOS calls **a_open**, **a_read**, or **s_open**, and **s_read_move**.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
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 caller's default user object is not qualified for list access to the directory.
E_FTYPE	0027H	The specified connection does not refer to a directory.
E_IFDR	002FH	One of these is true: • This system call applies only to named and DOS directories, but the STRING pointed to by `dir_name_ptr` specifies another type of file. • This system call is not supported for remote files.
E_IO	002BH	An I/O error occurred that might have prevented the operation from completing.
E_LIMIT	0004H	The calling task's job has 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_CONFIGURED	0008H	This system call is not part of the present configuration.

s_get_file_status

Obtains information about a file of any type. This call can be used with any file, including those created by the BIOS.

Syntax, PL/M and C

```
CALL rq$s$get$file$status (path_ptr, info_ptr, except_ptr);
```

```
rq_s_get_file_status (path_ptr, info_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`path_ptr`	POINTER	STRING far *
`info_ptr`	POINTER	S_FILE_STATUS_STRUCT far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameter

`path_ptr`

A pointer to a STRING that contains the path for the file. The format of this path varies depending on the file type.

See also: Path syntax, *System Concepts*

`info_ptr`

A pointer to this structure where the EIOS returns file status information. The information returned depends on the type of file specified. For all types of files, the first part of this structure through the `device_connection` field returns. If the contents of the `named_file` field indicate a named file, the second part (from `file_ID` on) returns. The `create_time`, `access_time`, `modify_time`, and `owner_access` elements have different meaning for DOS files.

```
DECLARE file_info STRUCTURE(
   device_share            WORD_16,
   number_connections      WORD_16,
   number_readers          WORD_16,
   number_writers          WORD_16,
   share                   BYTE,
   named_file              BYTE,
   device_name(14)         BYTE,
   file_drivers            WORD_16,
   functions               BYTE,
   flags                   BYTE,
   device_granularity      WORD_16,
   device_size             WORD_32,
   device_connections      WORD_16,
   file_id                 WORD_16,
   file_type               BYTE,
   file_granularity        BYTE,
   owner_id                WORD_16,
   create_time             WORD_32,
   access_time             WORD_32,
   modify_time             WORD_32
   file_size               WORD_32,
   file_blocks             WORD_32,
   volume_name(6)          BYTE,
   volume_granularity      WORD_16,
   volume_size             WORD_32,
   accessor_count          WORD_16,
   owner_access            BYTE);
```

or

```
typedef struct {
   UINT_16                device_share;
   UINT_16                number_connections;
   UINT_16                number_readers;
   UINT_16                number_writers;
   UINT_8                 share;
   UINT_8                 named_file;
   UINT_8                 device_name[14];
   UINT_16                file_drivers;
   UINT_8                 functions;
   UINT_8                 flags;
   UINT_16                device_granularity;
   UINT_32                device_size;
   UINT_16                device_connections;
   UINT_16                file_id;
   UINT_8                 file_type;
   UINT_8                 file_granularity;
   UINT_16                owner_id;
   UINT_32                creation_time;
   UINT_32                access_time;
   UINT_32                modify_time;
   UINT_32                file_size;
   UINT_32                file_blocks;
   UINT_8                 volume_name[6];
   UINT_16                volume_granularity;
   UINT_32                volume_size;
   UINT_16                accessor_count;
   UINT_8                 owner_access;
} S_FILE_STATUS_STRUCT;
```

Where:

`device_share`

This is always set to 1, indicating that all devices can be shared.

`number_connections`

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

`number_readers`

The number of connections currently open for reading. For remote and NFS files a 0 indicates either no connection or a connection open for writing only, and a 1 indicates an open readable or read/writable connection.

`number_writers`

The number of connections currently open for writing. For remote files a 0 indicates either no connection or a connection open for reading only, and a 1 indicates an open writable or read/writable connection.

`share` The current shared status of the file; possible values are

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

If a remote or NFS file is open, the share mode used to open the connection is returned, but if the file connection is not open, share mode 3 is indicated.

`named_file`

Indicates if this structure contains any information beyond the `device_connections` field.

Value	**Meaning**
0	No
0FFH	Yes

`device_name`

The physical device name where this file resides. This name is padded with blanks. Device names should not exceed 14 characters in length.

For remote files, this is the name of the remote server on which the file resides. For NFS files, this is the host name and path used when the device was attached.

file_drivers

Indicates what kinds of files can reside on this device. When the device is formatted, this value is copied into the device volume label.

File Type Bit	**File Type**
7-6	Reserved
5	EDOS file
4	Remote (iRMX-NET) or NFS file
3	Named file
2	DOS
1	Stream file
0	Physical file

functions Describes the functions supported by the device where this file resides. A bit set to 1 indicates the corresponding function is supported. This field is not supported by the remote file driver; 0 is always returned for remote files.

Bit	**Function**
7	F_CLOSE
6	Reserved
5	F_DETACH_DEV
4	Reserved
3	F_ATTACH_DEV
2	F_SEEK
1	F_WRITE
0	F_READ

`flags` Meaningful only for diskette drives. This field is not supported by iRMX-NET or the NFS file driver; 0 returns for such remote files.

Bits	Value	Function
7-5		Reserved; set to 0
4	0	Standard diskette, for MBI only; track 0 is single-density, 128-byte sectors
	1	Uniform diskette or not a diskette
3	0	High (quad) density
	1	Low (double) density For 8" diskettes, set to 0
2	0	Single sided
	1	Double sided
1	0	Single density
	1	Not single density

Disk Size	Bit 1	Bit 3
3.5D	1	1
3.5Q	1	0
5.25D	1	1
5.25Q	1	0
8S	0	0
8D	1	0

Bits	Value	Function
0	0	This field is undefined
	1	Bits 7-1 are valid

See also: Supporting the standard diskette format, *Driver Programming Concepts*

`device_granularity`

The granularity, in bytes, of the device where this file resides.

`device_size`

The storage capacity of the device, in bytes.

`device_connections`

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

`file_id` An fnode number that distinguishes this file from all other files on the same device.

`file_type` The file type.

Value	Meaning
6	Directory file
8	Data file

`file_granularity`
The file granularity as a multiple of `volume_granularity`. For remote and NFS files, 1 is always returned.

`owner_id` The first ID in the creating task's default user object.

`create_time, access_time, modify_time`
The time and date when the file was created, accessed, or modified. For DOS files, only `creation_time` or `modify_time` returns. For ICU-configurable systems, an ICU option determines whether the OS maintains these fields.

See also: TF parameter, *ICU User's Guide and Quick Reference*

`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 `volume_granularity` bytes of data.

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

`volume_granularity`
The volume granularity, in bytes.

`volume_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. User IDs for NFS files may be mapped differently for different OSs.

`owner_access`

The access rights to this file that are currently held by the owner. In this table, access is granted if a bit is set to 1. Access rights for NFS files may be mapped differently for different OSs:

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

See also: Accessing NFS files, Chapter 17, *System Concepts*

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

For asynchronous BIOS calls, some returned information might be inaccurate. For instance, if the application invokes **s_get_file_status** while the BIOS is processing an **a_write** call for the same file, the values returned in the file size fields might be incorrect. The EIOS cannot check such values and does not check access before returning file status information.

See also: BIOS call **a_write**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_ALREADY_ATTACHED	0038H	The EIOS is unable to attach the device containing the file because the BIOS has done so.
E_CONTEXT	0005H	The calling task's job is not an I/O job.
E_DEV_DETACHING	0039H	The device containing the specified file is being detached.
E_DEVFD	0022H	The EIOS attempted to physically attach a device that had formerly been only logically attached. It found that the device and the device driver specified in the logical attachment were incompatible.

E_FNEXIST	0021H	At least one of these is true: • A file in the specified path, or the target file itself, does not exist or is marked for deletion. • The physical device specified in the call was not found.
E_FTYPE	0027H	A path component is not a directory file.
E_ILLVOL	002DH	The EIOS attempted to physically attach a device that had formerly been only logically attached. It found that the volume does not contain named files. The named, remote, DOS, or EDOS file driver was requested during logical attachment.
E_INVALID_FNODE	003DH	The fnode for the specified file is invalid. The file cannot be accessed; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_MODE	0056H	A tape drive attempted a read (write) before the previous write (read) completed.
E_IO_NO_DATA	0055H	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 I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_IO_MEM	0042H	The BIOS job does not currently have a block of memory large enough to allow this system call to complete.

E_LIMIT	0004H	At least one of these is true: • The user object or the calling task's job is involved in 255 I/O operations. • The calling task's job is not an I/O job. • The calling task's object limit has been reached.
E_LOG_NAME_NEXIST	0045H	The specified path contains a logical name, but the call was unable to find this name in the object directories of the calling task's local job, the global job, or the root job.
E_LOG_NAME_SYNTAX	0040H	The specified logical name contains at least one of these syntax errors: • The logical name was missing matching colons. • Contains a logical name that exceeds 12 characters, has no characters, or contains invalid characters.
E_MEDIA	0044H	The device containing the specified file is off-line.
E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E_NOPREFIX	8022H	The default prefix for the calling task's job is undefined, or is not a valid device or file connection.
E_NOT_CONFIGURED	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_NOT_LOG_NAME	8040H	The specified path contains a logical name that refers to an object that is not a device connection or a file connection.
E_NOUSER	8021H	The calling task's job does not have a default user, or is not a user object.

E_PARAM	8004H	The EIOS attempted to physically attach a device that had formerly been only logically attached. It found that the logical attachment referred to a file driver that is not configured into your system. See also: For ICU-configurable systems, DFD parameter, *ICU User's Guide and Quick Reference*
E_PATHNAME_SYNTAX	003EH	The specified pathname contains invalid characters.
E_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF. The server's UDF must have World read permission.

get_logical_device_status

Provides status information about logical names that represent devices. The EIOS does not check access before returning status information.

Syntax, PL/M and C

```
CALL rq$get$logical$device$status (log_name_ptr, dev_info_ptr,
      except_ptr);
```

```
rq_get_logical_device_status (log_name_ptr, dev_info_ptr,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
log_name_ptr	POINTER	STRING far *
dev_info_ptr	POINTER	DEVICE_INFO_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

log_name_ptr

A pointer to a STRING of 12 or fewer characters, optionally delimited with colons, containing the logical name under which the logical device object is cataloged in the root object directory.

dev_info_ptr

A pointer to a structure, where status information returns, of this form:

```
DECLARE device_info STRUCTURE(
   device_name(15)        BYTE,
   file_driver            BYTE,
   num_conns              WORD_16,
   owner_id               WORD_16);
```

or

```
typedef struct {
   UINT_8                 device_name[15];
   UINT_8                 file_driver;
   UINT_16                num_conns;
   UINT_16                owner_id;
} DEVICE_INFO_STRUCT;
```

Where:

`device_name`
The physical name associated with the device. The first byte is the length of the field, the second is a colon, then up to 12 bytes for the name, followed by a colon. For ICU-configurable systems, this name is established during system configuration.

See also: **attachdevice**, *Command Reference*,
DPN parameter, *ICU User's Guide and Quick Reference*

`file_driver`
The type of file driver associated with the device. Possible values include:

Value	**File Driver**
1	Physical
2	Stream
3	DOS
4	Named
5	Remote
6	EDOS
7-max	Loadable file drivers, including NFS. The IDs can vary, depending on which driver is loaded first.

`num_conns` The current number of connections to the device.

`owner_id` The owner ID for this device. This ID is the first ID listed in the default user object of the attaching task's job.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The device connection corresponding to the logical name is being deleted.
E_LIMIT	0004H	Either the user object or the calling task's job is involved in 255 I/O operations.
E_LOG_NAME_NEXIST	0045H	The logical name was not found in the root object directory.

E_LOG_NAME_SYNTAX	0040H	The syntax of the specified logical name is incorrect. At least one of these is true: • The logical name was missing matching colons. • The STRING pointed to by the `log_name_ptr` parameter has a length of 0 or greater than 12. • The logical name contains invalid characters.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NOT_DEVICE	8041H	The specified logical name does not represent a valid device connection.

s_get_path_component

Returns the name of a named file (including remote and DOS), as cataloged in its parent directory.

Syntax, PL/M and C

```
CALL rq$s$get$path$component (connection, name_ptr,
      except_ptr);
```

```
rq_s_get_path_component (connection, name_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
name_ptr	POINTER	STRING far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection
A token for the file connection whose name is sought.

name_ptr
A pointer to a STRING where the OS returns the path component. The maximum length of the STRING is 14 bytes.

except_ptr
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

A null STRING returns if a stream or physical file, or the root directory of a named or remote file is referenced.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The name_ptr parameter is a null pointer.
E_FNEXIST	0021H	The file is marked for deletion. The STRING is undefined.

E_INVALID_FNODE	003DH	The fnode for the specified file is invalid. The file cannot be accessed; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*
E_IO	002BH	An I/O error might have prevented completion of the operation.
E_IO_MEM	0042H	Memory available to the EIOS is not sufficient to complete the call.
E_NOT_FILE_CONN	0032H	For remote files, the connection parameter must be a file connection, not a device connection.

get_user_ids

Returns the user ID(s) associated with a user defined in the UDF.

Syntax, PL/M and C

```
CALL rq$get$user$ids (name_ptr, ids_ptr, except_ptr);
```

```
rq_get_user_ids (name_ptr, ids_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
name_ptr	POINTER	STRING far *
ids_ptr	POINTER	IDS_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

name_ptr

A pointer to a STRING containing the user name. Only the first 8 characters are significant.

ids_ptr

A pointer to this structure where the ID(s) associated with the user name is placed:

```
DECLARE ids STRUCTURE (
   length         WORD_16,
   count          BYTE,
   IDs(*)         BYTE);
```

or

```
typedef struct {
   UINT_16        length;
   UINT_16        count;
   UINT_16        ids[_NUM_IDS]; /* adjust to count value */
 } IDS_STRUCT;
```

Where:

`length`	Should be set by the caller to the maximum number of ID(s) desired.
`count`	The number of valid IDs in the ID array after **get_user_ids** returns to the caller. This value will never be greater than the `length` parameter. The calling task does not need to initialize this value.
`IDs`	An array of IDs obtained from the UDF. The length of this array is contained in `count`. The calling task does not need to initialize this array.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This system call searches the user definition file *:config:udf* for the user name pointed to by the `name_ptr` parameter and if found, returns that user's ID(s).

See also: *:config:udf* file, *Command Reference,*
for ICU-configurable systems, I/O Users screen and CD parameter,
ICU User's Guide and Quick Reference

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_CALL	8005H	A task wrote over the interface library or the EIOS job.
E_CONTEXT	0005H	The calling job is not an I/O job.
E_DEV_DETACHING	0039H	An I/O operation could not be performed on the device *:sd:* because it was being detached.
E_DEVFD	0022H	The device *:sd:* cannot be used with the file driver as specified in the preceding logical attach operation.
E_UDF_FORMAT	0048H	The UDF is not in the correct format.
E_FACCESS	0026H	User does not have access rights for the requested operation.
E_FLUSHING	002CH	The device *:sd:* is being detached.

E_FNEXIST	0021H	At least one of these is true: • The UDF or a file in *:config:* does not exist. • The specified physical device containing *:config:udf* was not found.
E_FTYPE	0027H	A path component is not a directory file.
E_ILLVOL	002DH	The file driver in the volume label conflicts with the file driver specified in the preceding logical attach operation.
E_INVALID_FNODE	003DH	The fnode for the specified file is invalid. The file cannot be accessed; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_MEM	0042H	The BIOS job did not have enough memory to perform the requested function.
E_IO_OPRINT	0053H	The device is off-line; operator intervention is required.
E_IO_SOFT	0051H	A soft error occurred. The BIOS has retried the operation and failed; a retry is not possible.
E_IO_UNCLASS	0050H	An unclassified I/O error occurred.
E_IO_WR_PROT	0054H	The volume specified in this call is write protected.
E_LIMIT	0004H	The root job object directory is full.
E_LOG_NAME_NEXIST	0045H	The logical name was not found in the caller's object directory, the global job object directory, or the root job object directory.
E_MEDIA	0044H	The device associated with the system call is off-line.
E_NAME_NEXIST	0049H	The name specified in this call is not defined. Only dynamic logon creates verified users.
E_NOPREFIX	8022H	The caller's job does not have a default prefix, or is invalid.

E_NOUSER	8021H	The caller's job does not have a default user, or is invalid.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	At least one of these is true: • The `name_ptr` parameter is a null pointer. • The `length` field of the `ids` structure is 0. • The name contains invalid characters.
E_SHARE	0028H	The file is not sharable with the requested access.

hybrid_detach_device

Temporarily removes the correspondence between a logical name and a physical device established with **logical_attach_device**. This system call does not remove the logical name from the root object directory.

Syntax, PL/M and C

```
CALL rq$hybrid$detach$device (log_name_ptr, except_ptr);
```

```
rq_hybrid_detach_device (log_name_ptr, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`log_name_ptr`	POINTER	STRING far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`log_name_ptr`

A pointer to a STRING of 12 or fewer characters, optionally delimited with colons, containing the logical name under which the logical device object is cataloged in the root object directory.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The EIOS detaches the device by issuing the BIOS **a_physical_detach_device** call. The EIOS specifies the hard detach option which deletes all connections to files on the device.

Reattach a device in one of two ways.

- A task can issue the BIOS call **a_physical_attach_device**.
- A task can use the device's logical name as the prefix portion of a pathname when issuing an EIOS call. The EIOS physically attaches the device using the parameters originally specified when the logical name was established in **logical_attach_device**.

A task cannot use **logical_attach_device** to reattach a device that **hybrid_detach_device** detached until it issues **logical_detach_device**.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The device connection corresponding to the logical name is being deleted.
E_LIMIT	0004H	Either the user object or the calling task's job is involved in 255 I/O operations.
E_LOG_NAME_NEXIST	0045H	The logical name was not found in the root object directory.
E_LOG_NAME_SYNTAX	0040H	The syntax of the specified logical name is incorrect. At least one of these is true: • The STRING pointed to by the `log_name_ptr` parameter is of length 0 or greater than 12 characters, or is missing matching colons. • The logical name contains invalid characters.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NOT_DEVICE	8041H	The specified logical name does not represent a valid device connection.
E_NOT_OWNER	0046H	The user specified by the default user object is not the user that attached the device.

logical_attach_device

Assigns a logical name to a physical device. Any task that uses this system call loses its device independence. Only a few selected tasks should perform all device attaching and detaching.

Syntax, PL/M and C

```
CALL rq$logical$attach$device (log_name_ptr, dev_name,
      file_driver, except_ptr);
```

```
rq_logical_attach_device (log_name_ptr, dev_name, file_driver,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`log_name_ptr`	POINTER	STRING far *
`dev_name`	POINTER	STRING far *
`file_driver`	BYTE	UINT_8
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`log_name_ptr`

A pointer to a STRING of 12 or fewer characters, possibly delimited with colons, that contains the logical name to be assigned to a device. The OS removes the colons so that a logical name with colons is the same as one without; *:F0:* is the same as *F0*. Colons do not count in the length of the name. When you subsequently use this name in other system calls, specify colons.

`dev_name`

A pointer to a STRING containing the device name to which the logical name is assigned. This is the name of a Device-Unit Information Block (DUIB) specified during system configuration. For all file types except NFS, device names longer than 14 characters are truncated by the call to 14 characters.

See also: **attachdevice**, *Command Reference*,
for ICU-configurable systems, Logical Names screen, *ICU User's Guide and Quick Reference*

`file_driver`

Specifies which type of BIOS file driver to use with the device:

Value	**File Driver**
1	Physical
2	Stream
3	DOS
4	Named
5	Remote
6	EDOS
7-max	Loadable file drivers, including NFS. The IDs can vary, depending on which driver is loaded first. To find what ID is currently assigned to a specific loadable driver, first call **rq_get_file_driver_status**.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This system call creates a logical device object that corresponds to a physical device. This logical device object is cataloged in the root object directory under the logical name pointed to by `log_name_ptr`. The logical device object must be cataloged before the EIOS can make connections to files on the device.

The first EIOS call that uses the logical name as a prefix in a pathname causes the physical device to be attached. The logical name can be used as a prefix in other system calls and can be deleted by **logical_detach_device**.

The EIOS uses the BIOS call **a_physical_attach_device**. Some condition codes that result because of errors in **logical_attach_device** are not returned until the EIOS tries to attach the device with **a_physical_attach_device**.

Depending on your system configuration, if the first attempt to attach the device fails, the EIOS will try again. The EIOS will continue trying to attach the device until the device is attached successfully or the configured number of retries has been reached.

See also: BIOS call **a_physical_attach_device**,
For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The root object directory already contains an entry with the name pointed to by the log_name_ptr parameter.
E_LIMIT	0004H	At least one of these is true: • The calling task's job object directory is full. • The root object directory is full. • The calling task's job is not an I/O job.
E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete this call.
E_LOG_NAME_SYNTAX	0040H	The specified logical name is incorrect. At least one of these is true: • The STRING pointed to by the `log_name_ptr` parameter is length 0 or greater than 12 characters. • The logical name contains invalid characters.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

logical_detach_device

Removes the correspondence between a logical name and a physical device, and removes the logical name from the root object directory.

Syntax, PL/M and C

```
CALL rq$logical$detach_device (log_name_ptr, except_ptr);

rq_logical_detach_device (log_name_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
log_name_ptr	POINTER	STRING far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

log_name_ptr

A pointer to a STRING of 12 or fewer characters, optionally delimited with colons, containing the logical name under which the logical device object is cataloged in the root object directory.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Logical_detach_device is issued by the task that used **logical_attach_device** to create the logical name, some other task in the same job as the attaching task, another job having the same owner ID in its default user object, or the system manager.

After **logical_detach_device** is issued, users cannot create new connections using the logical name as a prefix. When the last file connection on the physical device is deleted, the EIOS detaches the device by issuing the BIOS call **a_physical_detach_device**.

Logical_detach_device closes all open file connections but does not flush the associated EIOS file buffers. These buffers will be flushed by issuing **s_close** before **logical_detach_device**.

⚠ **CAUTION**

Data will be lost if you do not flush the buffers. If a job with open file connections and active EIOS file buffers is deleted, the EIOS buffers will be flushed as part of the job deletion process.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The device connection corresponding to this logical name is being deleted.
E_LIMIT	0004H	One of these is true: • The job has reached the object limit of the calling task's object directory. • Either the user object or the calling task's job is involved in 255 I/O operations. • The calling task's job is not an I/O job.
E_LOG_NAME_NEXIST	0045H	The logical name was not found in the root object directory.
E_LOG_NAME_SYNTAX	0040H	The syntax of the specified logical name is incorrect. At least one of these is true: • The STRING pointed to by the `log_name_ptr` parameter is length 0 or greater than 12. • The logical name contains invalid characters.
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_NOT_DEVICE	8041H	The specified logical name does not represent a valid device connection.
E_NOT_OWNER	0046H	The default user object is not the user that originally attached the device.

s_lookup_connection

Accepts a logical name from the calling task and returns a token for the associated connection.

Syntax, PL/M and C

```
connection = rq$s$lookup$connection (log_name_ptr, except_ptr);

connection = rq_s_lookup_connection (log_name_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection`	SELECTOR	SELECTOR
`log_name_ptr`	POINTER	STRING far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`connection`

The token that represents the connection associated with the logical name.

Parameters

`log_name_ptr`

A pointer to a STRING of 12 or fewer characters, optionally delimited with colons, containing the logical name to be looked up. The OS removes the colons so that a logical name with colons is the same as one without; *:F0:* is the same as *F0*. Colons do not count in the length of the name.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

After converting any lowercase letters in the logical name to uppercase, the EIOS searches for the logical name. It first checks the object directory of the local job, the global job, and finally the root job.

This system call can look up logical names created by **catalog_object**. However, **catalog_object** does not convert from lowercase to uppercase. For compatibility, use uppercase characters with **catalog_object**.

See also: Nucleus call **catalog_object**,
Search sequence, *System Concepts*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The calling task's job is not an I/O job.
E_LIMIT	0004H	The job has reached the object limit of the calling task's object directory.
E_LOG_NAME_NEXIST	0045H	The specified path contains a logical name, but the call was unable to find this name in the object directories of the calling task's local job, the global job, or the root job.
E_LOG_NAME_SYNTAX	0040H	The specified logical name contains at least one of these syntax errors: • The logical name was missing matching colons. • The specified path contains a logical name that exceeds 12 characters, has no characters, or contains invalid characters.
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_NOT_CONNECTION	8042H	The logical name does not refer to a connection object.
E_TIME	0001H	The calling task's job is not an I/O job.

s_open

Opens a file connection for any file type.

Syntax, PL/M and C

```
CALL rq$s$open (connection, mode, number_buffers, except_ptr);
```

```
rq_s_open (connection, mode, number_buffers, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection`	SELECTOR	SELECTOR
`mode`	BYTE	UINT_8
`number_buffers`	BYTE	UINT_8
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection`

A token for the file connection to be opened. The connection must have been created in the calling task's job. If the connection was created in a different job, use **s_attach_file** to obtain a new connection.

`mode`

Indicates the access and share states of the connection. Use one for remote directories.

Value	Meaning
01H	Read only; share with all.
02H	Write only; share with all.
03H	Read and write; share with all.
04H	Read only; private use.
05H	Write only; private use.
06H	Read and write; private use.
07H	Read only; share with readers.
08H	Write only; share with readers.
09H	Read and write; share with readers.
0AH	Read only; share with writers.
0BH	Write only; share with writers.
0CH	Read and write; share with writers.

`number_buffers`

Specifies the number of buffers that the EIOS should allocate for this connection. This number must be between 0 and the maximum configured value.

See also: For ICU-configurable systems, Driver screens, *ICU User's Guide and Quick Reference*

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This system call creates the number of buffers requested, sets the connection's file pointer to 0, and starts reading ahead if the number of buffers is greater than 0 and the mode parameter includes reading.

Do not delete a task while it is using this system call.

If you don't know how the connection is used, specify both reading and writing.

The DOS World user always has read (list) access to DOS files and directories; write (delete, append, update, add-entry and change-entry) access is optional.

See also: **s_attach_file**

The iRMX-NET remote file's access rights are checked only during operations on the connection. This won't affect your programs if you:

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

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONN_OPEN	0035H	The connection is open.
E_DEV_OFF_LINE	002EH	The device is off-line or an unspecified DOS error occurred.
E_EXIST	0006H	The connection parameter is not a token for an existing object.
E_FACCESS	0026H	The access rights prohibit opening the file in the specified mode. If a named (including remote and DOS) file, the mode value does not match the connection's access rights when it was created.

E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_MODE	0056H	A tape drive attempted a read (write) before the previous write (read) completed.
E_IO_NO_DATA	0055H	A tape drive attempted to read the next record, but 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 I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_IO_WRPROT	0054H	The volume is write-protected.
E_LIMIT	0004H	At least one of these is true: • DOS has run out of file handles. • The calling task's job is not an I/O job. • The calling task's job, or the job's default user object, is involved in 255 I/O operations. • Processing this call would deplete the remote server's resources.
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_NOT_CONNECTION	8042H	The connection parameter is not a connection object token.
E_NOT_FILE_CONN	0032H	The connection is a device connection.
E_PARAM	8004H	The mode parameter is set to other than 1 through 0CH.

E_SHARE	0028H	At least one of these is true: • The call attempted to open a directory file or a bit-map file for writing. • The file's share state is not compatible with the mode specified in this call. • The call attempted to open a remote directory with the mode parameter set to other than 1.
E_SUPPORT	0023H	The specified connection was not created by a task in the calling task's job.

s_read_move

Reads a number of contiguous bytes from a file to a buffer specified by the calling task.

Syntax, PL/M and C

```
bytes_read = rq$s$read$move (connection, buffer_ptr,
      bytes_desired, except_ptr);
```

```
bytes_read = rq_s_read_move (connection, buffer_ptr,
      bytes_desire, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
bytes_read	WORD_32	NATIVE_WORD
connection	SELECTOR	SELECTOR
buffer_ptr	POINTER	UINT_8 far *
bytes_desired	WORD_32	NATIVE_WORD
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

bytes_read

Indicates the actual number of bytes read from the file.

Parameters

connection

A token for the connection to the file. This connection must be open for reading or for reading and writing, and the file pointer of the connection must point to the first byte to read.

See also: **s_change_access**, BIOS call **a_change_access**

buffer_ptr

A pointer to a user-supplied buffer that receives the information read from the file. Up to 4 Gbytes can be read.

bytes_desired

Specifies the maximum number of bytes to read from the file. If the EIOS detects an EOF before reading the number of bytes requested, it returns only those bytes preceding the EOF.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

DOS directory files can only be read a multiple of 16 bytes at a time, on 16-byte boundaries. Otherwise, E_SUPPORT returns.

Do not delete a task while it is using this system call.

If a condition code other than E_OK returns, the information in the buffer and the `bytes_read` parameter are meaningless.

If your task performs random-access reads of the file, it must identify which bytes to read. Use **s_seek** to position the connection's file pointer to the first byte to read. If your task reads from the file sequentially, the EIOS maintains the connection's file pointer automatically.

See also: **s_seek**

For better performance, the priority of the invoking task should be equal to or lower (numerically greater) than 130. If the priority of the calling task is greater than 130, the OS cannot overlap the read with computation or with other I/O operations.

See also: Setting priorities, *System Concepts*

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_granularity` parameter returned by **s_get_file_status** indicates the buffer size of a remote server.

If you use an iRMX segment as your buffer, the OS will detect when a task attempts to write beyond a buffer. If you create a buffer at compilation time, the information immediately following the buffer could be overwritten.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_BUFF	8023H	One of these is true: • The specified memory buffer is not writable. • The specified memory buffer crosses a segment boundary.
E_CONN_NOT_OPEN	0034H	At least one of these is true: • The connection is not open for reading or for reading and writing. • The connection is closed. • The connection was opened by **a_open**, not **s_open**.

E_EXIST	0006H	The connection is not a token for an existing object.
E_FLUSHING	002CH	The specified device is being detached.
E_IDDR	002AH	This request is invalid for the specified device driver.
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_MODE	0056H	A tape drive attempted a read (write) before the previous write (read) completed.
E_IO_NO_DATA	0055H	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 I/O System tried to perform the operation a number of times and failed. The number of retries is a configuration parameter. Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_LIMIT	0004H	At least one of these is true: • The calling task's job, or the job's default user object, is involved in 255 I/O operations. • The calling task's job is not an I/O job.
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_NOT_CONNECTION	8042H	The connection parameter is not a token for a file connection.

E_SPACE	0029H	At least one of these is true: • This call attempted to read beyond the end of the volume. • Another task is writing to the file using the same connection and is attempting to write beyond the end of the volume or the end of the available space on the volume.
E_SUPPORT	0023H	The connection parameter was not created by a task in the calling task's job, or the request involved a DOS directory but the byte and boundary restrictions were not adhered to.

s_rename_file

Changes the pathname of a named directory or data file, including remote and DOS. It cannot be used for stream or physical files.

➠ **Note**

When you rename a directory, you change the paths for all files and other directories contained in the directory.

Syntax, PL/M and C

```
CALL rq$s$rename$file (path_ptr, new_path_ptr, except_ptr);
```

```
rq_s_rename_file (path_ptr, new_path_ptr, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`path_ptr`	POINTER	STRING far *
`new_path_ptr`	POINTER	STRING far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`path_ptr`

A pointer to a STRING that specifies the current path for an existing file or directory to be renamed.

`new_path_ptr`

A pointer to a STRING that specifies the new path for the file. This path must comply with the syntax and semantics of paths for named files. This path cannot refer to an existing file.

See also: Paths, *System Concepts*

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

A task can change any aspect of a directory's or file's path so long as it remains on the same volume.

DOS users cannot rename a directory as a subdirectory. The DOS World user must have write access to the file to rename it; write (delete, append, update, add-entry, and change-entry) access is optional.

The iRMX default user object of the calling task's job must have deletion access to the original file and add-entry access to the file's new parent directory.

See also: **s_change_access**, BIOS call **a_change_access**

S_rename_file cannot rename these iRMX-NET entries:

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

The remote directory's or file's access rights are checked during operations on the connection. This won't affect your programs if you:

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

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_ALREADY_ATTACHED	0038H	The EIOS is unable to attach the device containing the file because the BIOS has done so.
E_CONTEXT	0005H	The calling task's job is not an I/O job.
E_DEV_DETACHING	0039H	The device containing the specified file is being detached.
E_DEVFD	0022H	The EIOS attempted to physically attach a device that had been only logically attached, and found that the device and the device driver specified in the logical attachment were incompatible.

E_FACCESS	0026H	At least one of these is true: • The call is trying to rename a bit-map file or the root directory. • The default user object associated with the calling task's job does not have add-entry access to the parent directory of the `new_path_ptr` file. • The default user object associated with the calling task's job does not have delete access to the file being renamed.
E_FEXIST	0020H	The new_path_ptr parameter refers to a file that already exists.
E_FNEXIST	0021H	A file in the specified path, or the file being renamed, does not exist or is marked for deletion.
E_FTYPE	0027H	A path component is not a directory file.
E_IFDR	002FH	The specified file is a stream or physical file.
E_ILLOGICAL_RENAME	003BH	The call attempted to rename a directory to a new path containing itself.
E_ILLVOL	002DH	The EIOS attempted to physically attach a device that had formerly been only logically attached. It found that the volume does not contain named files. The named file driver was requested during logical attachment.
E_INVALID_FNODE	003DH	The fnode for the specified file is invalid. The file cannot be accessed; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_OPRINT	0053H	The device was off-line. Operator intervention is required.

E_IO_SOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_IO_WRPROT	0054H	The volume is write-protected.
E_IO_MEM	0042H	The BIOS job does not currently have a block of memory large enough to allow this system call to complete.
E_LIMIT	0004H	At least one of these is true: • The user object or the calling task's job is involved in 255 I/O operations. • The calling task's job is not an I/O job. • The calling task's object limit has been reached. • Processing this call would deplete the remote server's resources.
E_LOG_NAME_NEXIST	0045H	At least one of the specified paths contains a logical name, but the call was unable to find this name in the object directories of the calling task's local job, the global job, or the root job.
E_LOG_NAME_SYNTAX	0040H	At least one of the specified paths contain one or more of these logical name syntax errors: • The logical name was missing matching colons. • A path contains a logical name that exceeds 12 characters, has no characters, or contains invalid characters.
E_MEDIA	0044H	The device containing the specified file is off-line.
E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.

E_NAME_NEXIST	0049H	The user object does not represent a verified user or is not properly defined in the remote server's UDF. Only dynamic logon creates verified users.
E_NOPREFIX	8022H	The default prefix for the calling task's job is undefined, or is not a valid device or file connection.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NOT_FILE_CONN	0032H	In the specified path, the subpath portion is null and the prefix portion is not a file connection.
E_NOT_LOG_NAME	8040H	At least one of the specified paths contains a logical name that refers to an object that is not a device connection or a file connection.
E_NOT_SAME_DEV	003AH	The two paths refer to different devices.
E_NOUSER	8021H	The calling task's job does not have a default user object, or the object cataloged in *r?iouser* is not a user object.
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	One or both of the specified pathnames contain invalid characters.
E_PARAM	8004H	The specified task_priority for an I/O job is unequal to 0 and is greater than the max_priority of the I/O job.
E_SPACE	0029H	At least one of these is true: • The volume is full. • 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_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF. The server's UDF must have World read permission.
E_SUPPORT	0023H	The task attempted to rename a physical or stream file.

s_seek

Moves the file pointer for any open physical or named file (including remote and DOS) connection. This system call cannot be used with stream files.

Syntax, PL/M and C

```
CALL rq$s$seek (connection, mode, move_count, except_ptr);

rq_s_seek (connection, mode, move_count, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
mode	BYTE	UINT_8
move_count	WORD_32	UINT_32
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection

A token for an open connection whose file pointer you wish to move.

mode

Describes the movement of the file pointer:

Value	File Pointer Movement
1	Back by move_count bytes; if the pointer moves past the beginning of the file, it is set to 0 (first byte).
2	Set to the position specified by move_count. Moving beyond the EOF is valid for named files only.
3	Forward by move_count bytes. Moving beyond the EOF is valid for named files only.
4	Move to the EOF and then back by move_count bytes; if the pointer moves beyond the beginning of the file, it is set to 0 (first byte). This option is not supported for DOS directories; E_SUPPORT returns.

move_count

Specifies how far, in bytes, to move the file pointer.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If your tasks are performing sequential I/O on a file, they do not need to use this system call. Otherwise, when performing random I/O, use this system call to position the file pointer before using **s_read_move**, **s_truncate_file**, or **s_write_move**. It is possible to position the file pointer beyond the EOF for a named file.

Do not delete a task while it is using this system call.

The connection must be open for reading only, writing only, or reading and writing. If not, use **s_open** to open the file. The DOS World user always has read (list) access to DOS files and directories; write (delete, append, update, add-entry and change-entry) access is optional.

The connection must have been created by a task within the calling task's job. If not, use the existing connection as a prefix, and have the calling task obtain a new connection by invoking **s_attach_file**.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_BUFF	8023H	One of these is true: • The specified memory buffer is not writable. • The specified memory buffer crosses a segment boundary.
E_CONN_NOT_OPEN	0034H	At least one of these is true: • The connection is not open. • The connection was opened by **a_open**, not **s_open**.
E_EXIST	0006H	The connection parameter is not a token for an existing object.
E_FLUSHING	002CH	The specified device is being detached.
E_IDDR	002AH	This request is invalid for the specified device driver.
E_IFDR	002FH	**S_seek** cannot be used with stream files.
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_MODE	0056H	A tape drive attempted a read (write) before the previous write (read) completed.

E_IO_NO_DATA	0055H	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 I/O System tried to perform the operation a number of times and failed (the number of retries is a configured option). Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_LIMIT	0004H	At least one of these is true: • Either the calling task's job, or the job's default user object, is involved in 255 I/O operations. • The calling task's job is not an I/O job.
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_NOT_CONNECTION	8042H	The connection parameter is not a token for a file connection.
E_PARAM	8004H	At least one of these is true: • The `mode` parameter is not in the range 1-4. • The calling task was attempting to seek past the end of a physical file.
E_SPACE	0029H	This seek forced the EIOS to attempt to empty the connection's buffer(s) by writing their contents to the volume. However, the volume is full.
E_SUPPORT	0023H	The connection parameter refers to a connection that was created by a task outside of the calling task's job.

s_set_file_status

Changes the owner and/or time stamps of a file.

Syntax, PL/M and C

```
CALL rq$s$set$file$status (path_ptr, set_info_ptr, except_ptr);
```

```
rq_s_set_file_status (path_ptr, set_info_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
path_ptr	POINTER	STRING far *
set_info_ptr	POINTER	SET_FILE_STATUS_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

path_ptr

A pointer to a STRING that contains the path for the file.

set_info_ptr

A pointer to this structure:

```
DECLARE set_file_status_struct STRUCTURE(
   select                 WORD_16,
   owner                  WORD_16,
   create_time            WORD_32,
   modify_time            WORD_32,
   access_time            WORD_32);
```

or

```
typedef struct {
   UINT_16                select;
   UINT_16                owner;
   UINT_32                create_time;
   UINT_32                modify_time;
   UINT_32                access_time;
} SET_FILE_STATUS_STRUCT
```

Where:

`select` Specifies the file attributes to set; encoded as follows:

Bit	Meaning
0	Change owner
1	Set creation time
2	Set last modified time
3	Set last access time
bits 4-15	Reserved, must be 0

`owner` File owner ID

`create_time`
The date and time the file was created.

`modify_time`
The date and time the file was last modified.

`access_time`
The date and time the file was last accessed.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

You must have write access to the specified file, since **s_set_file_status** attaches the file and opens it for writing. If the file is currently open and share privileges have not been granted, **s_set_file_status** fails and returns E_SHARE.

Not all file drivers support this system call due to file system limitations. This is the level of support provided by each standard file driver:

File Driver	Support
Physical	None
Stream	None
DOS	Only last modified time
Named	Full support
Remote	Local full support, remote support is system-dependent
EDOS	Only last modified time
NFS	Fully supported except you cannot change the owner

Condition Codes

E_OK	0000H	No exceptional conditions occurred.

E_ALREADY_ATTACHED	0038H	The EIOS is unable to attach the device containing the file because the BIOS has done so.
E_CONTEXT	0005H	The calling task's job is not an I/O job.
E_DEV_DETACHING	0039H	The device containing the specified file is being detached.
E_DEVFD	0022H	The EIOS attempted to physically attach a device that had formerly been only logically attached. It found that the device and the device driver specified in the logical attachment were incompatible.
E_FACCESS	0026H	The default user does not have write access to the file.
E_SHARE	0028H	The file's current share mode will not allow a connection to be opened with write access.
E_SUPPORT	0023H	The file driver associated with the specified connection does not support this system call.
E_FNEXIST	0021H	At least one of these is true: • A file in the specified path, or the target file itself, does not exist or is marked for deletion. • The physical device specified in the call was not found.
E_FTYPE	0027H	A path component is not a directory file.
E_ILLVOL	002DH	The EIOS attempted to physically attach a device that had formerly been only logically attached. It found that the volume does not contain named files. The named, remote, NFS, DOS, or EDOS file driver was requested during logical attachment.
E_INVALID_FNODE	003DH	The fnode for the specified file is invalid. The file cannot be accessed; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.

E_IO_MODE	0056H	A tape drive attempted a read (write) before the previous write (read) completed.
E_IO_NO_DATA	0055H	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 I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_MEM	0042H	The BIOS job does not currently have a block of memory large enough to allow this system call to complete.
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_IO_WRPROT	0054H	The volume is write-protected.
E_LIMIT	0004H	At least one of these is true: • The user object or the calling task's job is involved in 255 I/O operations. • The calling task's job is not an I/O job. • The calling task's object limit has been reached.
E_LOG_NAME_NEXIST	0045H	The specified path contains a logical name, but the call was unable to find this name in the object directories of the calling task's local job, the global job, or the root job.
E_LOG_NAME_SYNTAX	0040H	The specified logical name contains at least one of these syntax errors: • The logical name was missing matching colons. • Contains a logical name that exceeds 12 characters, has no characters, or contains invalid characters.
E_MEDIA	0044H	The device containing the specified file is off-line.

E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E_NOPREFIX	8022H	The default prefix for the calling task's job is undefined, or is not a valid device or file connection.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NOT_FILE_CONN	0032H	For remote and NFS files, the connection parameter must be a file connection, not a device connection.
E_NOT_LOG_NAME	8040H	The specified path contains a logical name that refers to an object that is not a device connection or a file connection.
E_NOUSER	8021H	The calling task's job does not have a default user, or is not a user object.
E_PARAM	8004H	The EIOS attempted to physically attach a device that had formerly been only logically attached. It found that the logical attachment referred to a file driver that is not configured into your system. See also: For ICU-configurable systems, DFD parameter, *ICU User's Guide and Quick Reference*
E_PATHNAME_SYNTAX	003EH	The specified pathname contains invalid characters.
E_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF. The server's UDF must have World read permission.

s_special

Enables tasks to communicate with devices, device drivers, and the stream file driver to perform operations that are less device-independent than other EIOS operations. This call is not valid for devices accessed through NFS.

Syntax, PL/M and C

```
CALL rq$s$special (connection, function, data_ptr, iors_ptr,
      except_ptr);
```

```
rq_s_special (connection, function, data_ptr, iors_ptr,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
function	WORD_16	UINT_16
data_ptr	POINTER	void far *
iors_ptr	POINTER	IORS_DATA_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection

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

function

Specifies the special function being requested. Each function is described in detail after the Additional Information section.

⇒ **Notes**

Bits 8 and 12 of the function field are reserved; do not use values that manipulate these bits in your applications or device drivers. Mask bits 8 and 12 when your device driver receives a function code from the I/O system.

Only function code 2 (Notify) is supported for remote servers and the DOS or EDOS file driver.

This table summarizes the values assigned to this parameter:

Function Code	File Type	Description
0	Physical	Format track
0	Stream	Query
1	Stream	Satisfy
2	Physical/Named	Notify (The only function supported for remote servers.)
3	Physical	Get disk data
4	Physical	Get terminal data
5	Physical	Set terminal data
6	Physical	Set signal
7	Physical	Rewind tape
8	Physical	Read tape file mark
9	Physical	Write tape file mark
10	Physical	Retension tape
11		Reserved
12	Physical	Set bad track/sector information
13	Physical	Get bad track/sector information
14-15		Reserved
16	Physical	Get terminal status
17-19		Reserved
20	Named/DOS/EDOS	Get device free space data
21-32767		Reserved
32768-65535		Available for user devices, except for values that use bits 8 or 12.
17	Physical	Cancel terminal I/O: Use BIOS call a_special, function code 17
18	Physical	Resume terminal I/O: Use a_special, function code 17
19	Physical	Perform disk mirroring: Use BIOS call a_special, function code 19

See also: BIOS call **a_special**

`data_ptr`

A pointer to a parameter block that your task uses to exchange information with the EIOS. The contents and form of the parameter block depend on the function being requested. Many of these data structures are identical to those in the BIOS call **a_special**; refer to the corresponding **a_special** function code for a complete description of the structure. If the function requires no parameter block, set `data_ptr` to null.

`iors_ptr`

A pointer to a structure described below. The EIOS uses this structure to return information to the calling task. If you set this pointer to null, the EIOS does not return the information. Most applications do not need this information.

```
DECLARE iors_data STRUCTURE(
   actual                 WORD_32,
   device                 WORD_16,
   unit                   BYTE,
   funct                  BYTE,
   subfunct               WORD_16,
   device_loc             WORD_32,
   buf_ptr                POINTER,
   count                  WORD_32,
   aux_ptr                POINTER)
```

or

```
typedef struct {
   NATIVE_WORD            actual;
   UINT_16                device;
   UINT_8                 unit;
   UINT_8                 funct;
   UINT_16                subfunct;
   UINT_32                device_loc;
   UINT_8 far *           buf_ptr;
   NATIVE_WORD            count;
   void far *             aux_ptr;
} IORS_DATA_STRUCT;
```

Where:

`actual` Number of bytes transferred during the function, if any.

`device` Device number identifying the device.

See also: For ICU-configurable systems, Device Driver screens, *ICU User's Guide and Quick Reference*

`unit` Number of the unit that contains the file on which the special function is being performed.

`funct` Code recognized by the driver, usually meaning that this is a special operation.

`subfunct` User-provided function code.

`device_loc`

Location on the device where the operation was performed.

buf_ptr Pointer to a buffer used for this operation, if any. For flat model applications only, treat this parameter as two separate fields in the structure. The first field has the name listed above and is a near pointer. The second field has the same name with _seg appended at the end. It is a segment selector for the pointer.

count Number of bytes transferred, if any.

aux_ptr Same as the call parameter data_ptr.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Do not delete a task while it is using this system call.

Only function code 2 (Notify) is supported for remote servers. When a task invokes **s_special** with a connection to a remote server and function 2, the calling task is notified of a communication failure immediately after an unsuccessful attempt to access the remote server, or when 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.

To restore the availability of a remote server, perform these 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.

Each of the special functions is described below. Descriptions appear in numerical order of the function parameter.

Format a Track (Function Code 0)

Call **s_special** with an open physical file connection, function set to 0, and data_ptr pointing to a structure of the form:

```
DECLARE format_track STRUCTURE(
   track_number           WORD_16,
   interleave             WORD_16,
   track_offset           WORD_16,
   fill_char              BYTE)
```

or

```
typedef struct {
   UINT_16                  track_number;
   UINT_16                  interleave;
   UINT_16                  track_offset;
   UINT_8                   fill_char;
} FORMAT_TRACK_STRUCT;
```

Where:

`track_number`
The number of the track to be formatted: from 0 to 1 less than the number of tracks on the volume. When formatting a tape or a RAM-disk, set to 0.

`interleave`
The interleave factor for the track: the number of physical sectors to skip when locating the next logical sector. 0 or 1 skips no physical sectors between logical sectors. If the specified interleave factor is greater than the number of physical sectors on a track, the OS divides the specified value by the number of physical sectors and uses the remainder as `interleave`. This field does not apply to tapes or RAM-disks.

`track_offset`
The number of physical sectors to skip between the index mark and the first logical sector. This does not apply to tapes or to RAM-disks.

`fill_char` A character with which each sector is written. Some drivers ignore this value and fill the sector with a character they establish.

See also: Function Code 3, Get Special Disk Data

Query For Information About Stream File Operation (Function Code 0)

Call **s_special** with a token for the connection to a stream file, `function` set to 0, and `data_ptr` set to null. Use this function to find out what is being requested by another task using the same stream file. For example, the task reading a stream file might need to know how many bytes are being sent by a task writing to the same file.

If a task is reading from or writing to the stream file, the EIOS returns this information to the `IORS_DATA_STRUCT` structure referenced by the `iors_ptr` parameter:

`actual` Number of bytes already transferred.

`count` Number of bytes remaining to be transferred.

`buf_ptr` A pointer to the memory location to be used for the next byte to be transferred.

`funct` Indicates the purpose of the queued request. 0 means read; 1 means write.

If no task is reading or writing, the EIOS queues this request. It remains queued until a task issues a read or write. If another arrives, the EIOS cancels both requests and returns E_STREAM_SPECIAL condition code to the calling tasks.

Satisfy Stream File Transactions (Function Code 1)

Call **s_special**, with a stream file connection, `function` set to 1, `data_ptr` and `iors_ptr` set to null. The only information that your task can obtain is the condition code. Use this function to force a stream file transaction to complete, even if the number of bytes written does not match the number of bytes read.

When one task tries to read or write a stream file, the task does not ordinarily run again until the complementary task issues a matching request. For example, suppose that Task A wants to read 512 bytes, but Task B writes only 256 bytes. Task A stops running until Task B supplies at least 256 more bytes.

Request Notification That Volume Is Unavailable (Function Code 2)

A volume mounted on a drive becomes unavailable because an operator opens a flexible disk drive door or presses the STOP button on other mass storage drives. A task can use **s_special** to request notification of this event. For most drives, notification occurs immediately. This function applies to named and physical files only. The `data_ptr` parameter points to this structure:

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

or

```
typedef struct {
   SELECTOR             mailbox;
   SELECTOR             object;
} NOTIFY_STRUCT;
```

Where:

`mailbox` A token for a mailbox.

`object` A token for an object. When the BIOS detects that the volume is unavailable, the object is sent to the mailbox. To cancel a request for notification, make a dummy request using the same connection with a null selector value in the mailbox parameter.

After a task has made a request for notification, the BIOS remembers the object and mailbox tokens until the volume is detected as being unavailable or until the device is detached by **a_physical_detach_device**. A task should be dedicated to waiting at the mailbox. If the volume is detected as being unavailable, the BIOS will not execute I/O requests to the volume's device. Such requests return with the IORS `status` field set to E_IO and the `unit_status` field set to IO_OPRINT; operator intervention is required.

See also: IORS, Chapter 1,
Accessing the IORS, *Programming Techniques*

If a task issues a subsequent notification request for the same device connection, the BIOS replaces the old mailbox and object values with the new ones. It does not return a condition code.

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

1. Close the door or restart the drive.
2. Call **a_physical_detach_device** to do a hard detach of the device.
3. Call **a_physical_attach_device** to reattach the device.
4. Create a new file connection.

See also: **a_physical_detach_device**, **a_physical_attach_device**

Get Disk Data (Function Code 3)

When a disk is formatted, you may place some special device data into the iRMX volume label. To get this data, call **s_special** with a token for a device connection, `function` set to 3, and `data_ptr` pointing to this structure:

```
DECLARE disk_label_data STRUCTURE(
   label_data(8)          BYTE);
```

or

```
typedef struct {
   UINT_8                 label_data[8];
} DISK_LABEL_DATA_STRUCT;
```

Where:

`label_data` The last eight bytes of the label on the iRMX named volume.

Get Terminal Characteristics (Function Code 4)
Set Terminal Characteristics (Function Code 5)

Use function code 4 to get the current characteristics before setting the terminal characteristics. Modify the returned structure to reflect the changes. Then use function code 5 to set the characteristics, using the modified structure as input.

Some attributes in this function can also be set with OSC sequences. You can use the OSC Query sequence when debugging, to ensure that your tasks invoked **s_special** correctly.

See also: Line editing, OSC sequences, translation, *Driver Programming Concepts*

Call **s_special** with a token for a connection to a terminal; get or set the terminal characteristics with `function` set to 4 or 5. `Data_ptr` points to this structure. Zero for any of the `connection_flags` through `scroll_lines` fields causes the I/O System to skip over the zeroed field, leaving it at its previous setting.

```
DECLARE terminal_attributes STRUCTURE(
   num_words               WORD_16,
   num_used                WORD_16,
   connection_flags        WORD_16,
   terminal_flags          WORD_16,
   in_baud_rate            WORD_32,
   out_baud_rate           WORD_32,
   scroll_lines            WORD_16,
   page_width              BYTE,
   page_length             BYTE,
   cursor_offset           BYTE,
   overflow_offset         BYTE,
   special_modes           WORD_16,
   high_water_mark         WORD_16,
   low_water_mark          WORD_16,
   fc_on_char              WORD_16,
   fc_off_char             WORD_16,
   link_parameter          WORD_16,
   spc_hi_water_mark       WORD_16,
   special_char(4)         BYTE);
```

or

```
typedef struct {
   UINT_16                 num_words;
   UINT_16                 num_used;
   UINT_16                 connection_flags;
   UINT_16                 terminal_flags;
   NATIVE_WORD             in_baud_rate;
   NATIVE_WORD             out_baud_rate;
   UINT_16                 scroll_lines;
   UINT_8                  page_width;
   UINT_8                  page_length;
   UINT_8                  cursor_offset;
   UINT_8                  overflow_offset;
   UINT_16                 special_modes;
   UINT_16                 high_water_mark;
   UINT_16                 low_water_mark;
   UINT_16                 fc_on_char;
   UINT_16                 fc_off_char;
   UINT_16                 link_parameter;
   UINT_16                 spc_hi_water_mark;
   UINT_8                  special_char[4];
} TERM_ATTRIB_STRUCT;
```

See also: Function codes 4 and 5 of BIOS call **a_special** for a description of the fields in this data structure

Set Signal Characters for Signaling from Terminal Keyboard (Function Code 6)

This function associates a keyboard character with a semaphore, so that whenever the character is entered at the terminal, the BIOS automatically sends a unit to the semaphore. Character-semaphore pairs are called signals. Up to 12 signal characters, each character being associated with a different semaphore, are allowed. Call **s_special** with a device connection, `function` set to 6, and `data_ptr` pointing to this structure:

```
DECLARE signal_pair STRUCTURE(
   semaphore               TOKEN,
   character               BYTE);
```

or

```
typedef struct {
   SELECTOR                semaphore;
   UINT_8                  character;
} SIGNAL_PAIR_STRUCT;
```

Where:

semaphore A token for the semaphore to be associated with the character. To delete a signal character, use a null selector.

character The signal character.

Value	**Meaning**
0-1FH, 7FH	TSC sends a unit to the associated semaphore when it receives this ASCII value.
20H-40H	Type-ahead buffer (and input buffer if a buffered device) is cleared and a unit is sent to the associated semaphore when it receives a character in the 0 to 1FH range (add 20H to desired control character).

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

Call **s_special** with a physical file connection, using these function codes and `data_ptr` values to perform four different operations on tape drives only:

Code	**data_ptr**	**Function**
7	Nil	The tape drive rewinds a tape to its load point. This function also terminates tape read and write operations. If a write operation, the tape drive writes a file mark before rewinding the tape.
8	Valid	The tape drive moves the tape to the next file mark. This function also terminates tape read operations. The value of the search byte in the read_file_mark structure (see below) determines the direction of the search.
9	Nil	The tape drive writes a file mark at the current position. This function also terminates tape write operations.
10	Nil	The tape drive fast-forwards to the end of the tape and then rewinds to the load point (retensioning).

If using Function Code 8, `data_ptr` points to this structure:

```
DECLARE read_file_mark STRUCTURE (search BYTE);
```

or

```
typedef struct {
   UINT_8                 search;
} READ_FILE_MARK_STRUCT;
```

Where:

`search` A value indicating the direction of the search:

Value	Meaning
00	search forward
0FFH	search backward (for start/stop drives only)

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

Use these functions to set (write) or get (read) the bad track/sector information of a volume. Any information already existing in the volume's Bad Track/Sector Information Block will be overwritten. If you wish to modify existing information, get, modify, then set the Bad Track/Sector Information. The `data_ptr` parameter must point to this structure:

```
DECLARE bad_track_info STRUCTURE(
   reserved                 WORD_16,
   count                    WORD_16,
   bad_tracks(1024)         WORD_32),
   badtracks(1024)          STRUCTURE (
   cylinder                 WORD_16,
   head                     BYTE,
   sector                   BYTE)
   AT (@bad_track_info.bad_tracks);
```

or

```
typedef struct {
   UINT_16                  cylinder;
   UINT_8                   head;
   UINT_8                   sector;
} BAD_TRACK_STRUCT;

typedef struct {
   UINT_16                  reserved;
   UINT_16                  count;
   BAD_TRACK_STRUCT         bad_tracks[1024];
} BAD_TRACK_INFO_STRUCT;
```

Where:

`reserved` Reserved for use by the device driver.

`count` The number of bad tracks/sectors listed in the `bad_tracks` structure, up to the maximum of 1024. A 0 in the `count` field indicates that no valid information is available (get) or that there are no bad tracks (set).

`bad_tracks`
A structure used to store the bad track/sector list. For each entry, a sub-structure defines the cylinder, head, and sector for each bad track. List bad tracks in ascending order.

Get Terminal Status (Function Code 16)

This function gets the status of a terminal that is being driven by a terminal device driver. Call **s_special** with a physical connection to terminal, `function` set to 16, and `data_ptr` pointing to this structure:

```
DECLARE term_status STRUCTURE(
   terminal_flags         WORD_16,
   input_conn_flags       WORD_16,
   input_state            WORD_16,
   input_conn             TOKEN,
   input_count            WORD_32,
   input_actual           WORD_32,
   raw_buf_count          WORD_16,
   type_ahead_count       BYTE,
   num_input_requests     BYTE,
   output_conn_flags      WORD_16,
   output_state           WORD_16,
   output_conn            TOKEN,
   output_count           WORD_32,
   output_actual          WORD_32,
   out_buf_count          WORD_16,
   num_output_requests    BYTE);
```

or

```
typedef struct {
   UINT_16                 terminal_flags;
   UINT_16                 input_conn_flags;
   UINT_16                 input_state;
   SELECTOR                input_conn;
   NATIVE_WORD             input_count;
   NATIVE_WORD             input_actual;
   UINT_16                 raw_buf_count;
   UINT_8                  type_ahead_count;
   UINT_8                  num_input_requests;
   UINT_16                 output_conn_flags;
   UINT_16                 output_state;
   SELECTOR                output_conn;
   NATIVE_WORD             output_count;
   NATIVE_WORD             output_actual;
   UINT_16                 out_buf_count;
   UINT_8                  num_output_requests;
} TERM_STATUS_STRUCT;
```

See also: Function code 16, BIOS call **a_special**, for descriptions of the fields in this data structure

Get Device Free Space Data (Function Code 20)

This function returns information about the free space available on the specified device.

Call **s_special** with an open file connection, `function` a WORD_16 set to 20, and `data_ptr` pointing to a structure of this form. Set `iors_ptr` to null.

```
DECLARE device_free_struct STRUCTURE(
   sector_size                WORD_16
   device_size                WORD_32
   bytes_free                 WORD_32
   files_free                 WORD_32
   reserved(2)                WORD_32);
```

or

```
typedef struct {
   UINT_16                 sector_size;
   UINT_32                 device_size;
   UINT_32                 bytes_free;
   UINT_32                 files_free
   UINT_32                 reserved[2];
}DEVICE_FREE_STRUCT;
```

Where:

`sector_size`
The minimum I/O transfer size for the device.

`device_size`
The total number of bytes available on the device (when empty).

`bytes_free`
The number of bytes available in the device file system.

`files_free`
The number of files available in the device file system. A returned value of `0FFFFFFFFH` indicates that this file does not apply — the number of files in the file system is limited only by the space on the device (DOS and EDOS file drivers).

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONN_NOT_OPEN	0034H	At least one of these is true: • The connection is not open. • The connection was opened by **a_open**, not **s_open**.
E_EXIST	0006H	The connection parameter is not a token for an existing object.
E_FLUSHING	002CH	The specified device is being detached.
E_IDDR	002AH	The requested function is not supported by the device containing the specified file.
E_IFDR	002FH	The EIOS does not support the requested function for the file driver associated with the connection.
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.

E_IO_MEM	0042H	The BIOS memory pool on the remote server does not have a block of memory large enough to allow the system call to complete.
E_IO_MODE	0056H	A tape drive attempted a read (write) before the previous write (read) completed.
E_IO_NO_DATA	0055H	The 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 I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_IO_WRPROT	0054H	The volume is write-protected.
E_LIMIT	0004H	At least one of these is true: • Either the calling task's job or the job's default user object is involved in 255 I/O operations. • The calling task's job is not an I/O job.
E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E_NOT_CONFIGURED	0008H	This call is not part of the present configuration.
E_NOT_CONNECTION	8042H	The connection parameter is not a file connection token.
E_PARAM	8004H	The function code is not a legitimate value.
E_SPACE	0029H	At least one of these is true: • The call attempted to format a track that is beyond the end of the volume. • When formatting a RAM-disk or a tape, the call attempted to format a track other than track 0.

E_STREAM_SPECIAL	003CH	At least one of these is true: • The calling task is attempting to satisfy a stream file request, but there is no request queued at the stream file. • The calling task is attempting to satisfy a stream file request, but the only queued request is a query. • The calling task is querying a stream file, but the only request queued at the file is another query. The EIOS removes both queries from the queue.
E_SUPPORT	0023H	The specified connection was created by a task outside of the calling task's job.

start_io_job

Starts the execution of the initial task in an I/O job. The task was not started when the I/O job was created.

Syntax, PL/M and C

```
CALL rq$start$io$job (io_job, except_ptr);

rq_start_io_job (io_job, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`io_job`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`io_job`

A token for the I/O job to be started. This is the same token that was returned to **create_io_job**.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

When you call **create_io_job**, use the `task_flags` parameter to specify that the task in the new job not run until **start_io_job** is issued. Then initialize any items that need to be set before the initial task runs. For example, you can create a job, catalog a logical name in the new job's object directory, and then issue **start_io_job**.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TIME	0001H	The job cannot be started yet, probably because the OS has not finished processing **create_io_job**.

s_truncate_file

Removes information from the end of a named (including DOS and remote) data file.

Syntax, PL/M and C

```
CALL rq$s$truncate$file (connection, except_ptr);
```

```
rq_s_truncate_file (connection, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection

A token for a connection to the named data file which is to be truncated. The current file pointer for this connection indicates where to truncate the file. The byte indicated by the pointer is the first byte to be dropped from the file.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Use **s_seek** to position the pointer before using **s_truncate_file**.

Truncation will occur immediately, regardless of the status of other connections to the same file unless the pointer is at or beyond the EOF.

File pointers for other connections to the file are not adjusted by the truncation and may be beyond the new EOF after **s_truncate_file**. If a task invokes **a_read** or **s_read_move** in this case, the BIOS behaves as though the read began at the EOF.

Do not delete a task while it is using this system call.

The connection must be open for writing only or for both reading and writing. If not, use **s_open** to open the connection.

The connection must have update access to the file. The EIOS computes a connection's access when the connection is created.

See also: **s_change_access**, BIOS call **a_change_access**

The DOS World user always has read (list) access to DOS files and directories; write (delete, append, update, add-entry and change-entry) access is optional.

The connection must have been created by a task within the calling task's job. If not, use the existing connection as a prefix, and invoke **s_attach_file**.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONN_NOT_OPEN	0034H	At least one of these is true: • The connection is open in the wrong mode. It must be open for writing or for both reading and writing. • The connection is not open. • The connection was opened by **a_open**, not **s_open**.
E_FACCESS	0026H	The connection does not have update access to the file.
E_EXIST	0006H	The connection parameter is not a token for an existing object.
E_IFDR	002FH	**S_truncate_file** can be used only on named files, not stream or physical files.
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_OPRINT	0053H	The device was off-line. Operator intervention is required.
E_IO_SOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_IO_WRPROT	0054H	The volume is write-protected.

E_LIMIT	0004H	At least one of these is true: • The calling task's job is not an I/O job. • Either the calling task's job, or the job's default user object, is involved in 255 I/O operations.
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_NOT_CONNECTION	8042H	The connection parameter is not a token for a file connection.
E_SPACE	0029H	The truncation required writing the contents of a buffer to the file, but the volume was full.
E_SUPPORT	0023H	The connection was created by a task outside the calling task's job.

s_uncatalog_connection

Deletes a logical name that was added by **s_catalog_connection** from the object directory of a job. Do not delete a task while it is using this system call.

Syntax, PL/M and C

```
CALL rq$s$uncatalog$connection (job, log_name_ptr, except_ptr);
```

```
rq_s_uncatalog_connection (job, log_name_pt, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`job`	SELECTOR	SELECTOR
`log_name_ptr`	POINTER	STRING far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`job` A token for a job. Setting this parameter to a null selector specifies the calling task's job.

`log_name_ptr`

A pointer to a STRING of 12 or fewer characters, optionally delimited with colons, containing the logical name to uncatalog. The OS removes the colons so that a logical name with colons is the same as one without; *:F0:* is the same as *F0*. Colons do not count in the length of the name.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The job parameter is not a token for an existing object.
E_LIMIT	0004H	The calling task's job is not an I/O job.
E_LOG_NAME_NEXIST	0045H	The call could not find the logical name in the job's object directory.

E_LOG_NAME_SYNTAX	0040H	The syntax of the specified logical name is incorrect because at least one of these is true: • The STRING pointed to by the `log_name_ptr` parameter is length 0 or greater than 12. • The logical name contains invalid characters.
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_TYPE	8002H	The job parameter is not a token for a job object.

verify_user

Validates a user's name and password, then modifies the user object to indicate verification.

Syntax, PL/M and C

```
CALL rq$verify$user (user_t, name_ptr, password_ptr,
      except_ptr);
```

```
rq_verify_user (user_t, name_ptr, password_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`user_t`	SELECTOR	SELECTOR
`name_ptr`	POINTER	STRING far *
`password_ptr`	POINTER	STRING far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`user_t`

A token for the user object to be verified. For DOS files, the EIOS ignores this parameter because the user is always World.

`name_ptr`

A pointer to a STRING containing the user name. This name is typically entered from the console during dynamic logon. Only the first eight characters are used; any additional characters are ignored.

`password_ptr`

A pointer to a STRING containing the unencrypted user password. This password is typically entered from the console at the same time as the `name_ptr` parameter. Only the first eight characters are used.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This system call searches the *:config:udf* file for a matching user name and password. The name must be the same as it appears in the UDF. The password parameter is encrypted and then compared to the encrypted version in the UDF. The ID defined in the UDF is also compared with the ID contained in the user object.

If a matching name, password, and ID are found, the user object is modified to indicate the user has been verified. Otherwise, an exceptional condition code returns and the user object is not modified.

See also: For ICU-configurable systems, I/O Users screen and CD parameter, *ICU User's Guide and Quick Reference,*
:config:udf file, *Command Reference*

If iRMX-NET is configured into your system and the **verify_user** call succeeds, the user also gains access to iRMX-NET remote files.

⟹ **Note**
The iRMX-NET remote file driver will reject all user tokens created by **create_user** unless **verify_user** is used to verify the user tokens created.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_CALL	8005H	A task wrote over the interface library or over the EIOS job.
E_CONTEXT	0005H	The user token has been verified.
E_DEVFD	0022H	The device cannot be used with the file driver as specified in the preceding logical attach operation.
E_DEVICE_DETACHING	0039H	An I/O operation could not be performed on the device because it was being detached.
E_EXIST	0006H	The user token parameter is not valid.
E_FACCESS	0026H	The user does not have the proper access rights for the requested operation.
E_FLUSHING	002CH	The device is being detached.

E_FNEXIST	0021H	One of these is true: • The file or a file in its path does not exist. • The specified physical device was not found.
E_FTYPE	0027H	A path component is not a directory file.
E_ILLVOL	002DH	The file driver in the volume label conflicts with the file driver specified in the preceding logical attach operation.
E_INVALID_FNODE	003DH	The fnode for the specified file is invalid. The file cannot be accessed; delete it or fix it with **diskverify**. See also: **diskverify**, *Command Reference*
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_MEM	0042H	The BIOS job did not have enough memory to perform the requested function.
E_IO_OPRINT	0053H	The device is off-line; operator intervention is required.
E_IO_SOFT	0051H	A soft error occurred and the BIOS has retried the operation and failed; a retry is not possible.
E_IO_UNCLASS	0050H	An unclassified I/O error occurred.
E_IO_WR_PROT	0054H	The volume is write protected.
E_LIMIT	0004H	The caller's job is not an I/O job.
E_LOG_NAME_NEXIST	0045H	The logical name was not found in the caller's object directory, the global job object directory, or the root job object directory.
E_LOG_NAME_SYNTAX	0040H	One of these was true: • The logical name was missing matching colons. • The logical name STRING has a length of 0 or more than 12 characters. • The logical name STRING contains invalid characters.
E_MEDIA	0044H	The device associated with the system call is off-line.

E_MEM	0002H	The caller's job does not have enough memory to perform the requested operation.
E_NAME_NEXIST	0049H	The name specified in this call is not defined. Only dynamic logon creates verified users.
E_NOPREFIX	8022H	The caller's job does not have a default prefix, or it is invalid.
E_NOT_CONFIGURED	0008H	This call is not part of the present configuration.
E_NOT_LOG_NAME	8040H	The token referred to by the logical name supplied does not refer to a valid device or file connection.
E_NOUSER	8021H	The caller's job does not have a default user, or is invalid.
E_PARAM	8004H	The name or the password contain invalid characters or the name length is 0.
E_PASSWORD_MISMATCH	004BH	The password is incorrect.
E_SHARE	0028H	The file cannot be shared using the requested access.
E_TYPE	8002H	The user_t parameter is not a user object token.
E_UDF_FORMAT	0048H	The UDF is not in the correct format.
E_UID_NEXIST	004AH	The user ID present in the user token does not match that specified in the UDF.

s_write_move

Writes a collection of bytes from a buffer to a file.

Syntax, PL/M and C

```
bytes_written = rq$s$write$move (connection, buf_ptr, count,
      except_ptr);

bytes_written = rq_s_write_move (connection, buf_ptr, count,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
bytes_written	WORD_32	NATIVE_WORD
connection	SELECTOR	SELECTOR
buf_ptr	POINTER	UINT_8 far *
count	WORD_32	NATIVE_WORD
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

bytes_written

Indicates the number of bytes that were actually written to the file. This number is always less than or equal to the number specified in the count parameter.

Parameters

connection

A token for the connection to the file where the information is written.

buf_ptr

A pointer to a contiguous buffer of up to 4 Gbytes that is to be written to the specified file.

count

Specifies the number of bytes to be written from the buffer to the file.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

To write information into a file, the connection must have been created by a task within the calling task's job and be open for writing or for both reading and writing. The connection may also have access rights for updating, appending, or both.

See also: **s_change_access**, BIOS call **a_change_access**

The DOS World user always has read (list) access to DOS files and directories; write (delete, append, update, add-entry and change-entry) access is optional.

The EIOS returns a condition code and writes fewer bytes than requested by the task (on return from the call, `bytes_written` is less than `count`) under two circumstances.

- When the EIOS encounters an I/O error
- When the volume to which your task is writing becomes full

The EIOS writes the first byte starting at the byte pointed to by the file pointer and updates the pointer. After the write is complete, the file pointer points to the byte immediately following the last byte written. Use **s_seek** to position the file pointer if you are performing random-access operations.

If your task is using a connection that has append access, the task can start a write beyond, rather than at, the EOF. The EIOS extends the file and performs the write. If the file is extended, the extended section contains unknown, random information. You can write data into this area later.

For better performance, the priority of the invoking task should be equal to or lower than (numerically greater than) 130. If the priority of the calling task is greater than 130, the OS cannot overlap the write with computation or with other I/O operations.

Do not delete a task while it is using this system call.

See also: **s_attach_file**, **s_seek**, **s_create_file**,
Setting priorities, *System Concepts*

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_granularity` parameter returned by **s_get_file_status** indicates the buffer size of a remote server.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_BUFF	8023H	One of these is true: • The specified source memory buffer is not readable. • The specified source memory buffer crosses segment boundaries.
E_CONN_NOT_OPEN	0034H	At least one of these is true: • The connection is not open or not open for writing. • The connection was opened with **a_open**, not **s_open**.
E_EXIST	0006H	The connection parameter is not a token for an existing object.
E_FACCESS	0026H	The call tried to write beyond the EOF, but the connection specified does not have append access to the file.
E_FLUSHING	002CH	The specified device is being detached.
E_FNODE_LIMIT	003FH	The file cannot be created or extended to this size because it has reached the maximum number of volume blocks. See also: File driver limitations, System Concepts manual
E_FRAGMENTATION	0030H	The disk is too fragmented to extend the file.
E_IO_HARD	0052H	A hard error occurred; the BIOS cannot retry the request.
E_IO_MODE	0056H	A tape drive attempted a read (write) before the previous write (read) completed.
E_IO_OPRINT	0053H	The device was off-line. Operator intervention is required.

E_IO_SOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed. The number of retries is a configuration parameter. Another retry might be successful. See also: For ICU-configurable systems, RPA parameter, *ICU User's Guide and Quick Reference*
E_IO_UNCLASS	0050H	An unknown I/O error occurred.
E_IO_WRPROT	0054H	The volume is write-protected.
E_LIMIT	0004H	At least one of these is true: • The calling task's job is not an I/O job. • The calling task's job, or the job's default user object, is involved in 255 I/O operations.
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_NOT_CONNECTION	8042H	The connection parameter is not a token for a file connection.
E_PARAM	8004H	The calling task is attempting to write beyond the end of a physical file.
E_SPACE	0029H	The volume is full.
E_SUPPORT	0023H	The connection parameter refers to a connection that was created by a task outside of the calling task's job.

□ □ □

Human Interface System Calls 5

c_backup_char

Moves the parsing buffer pointer back one character (byte) for each occurrence of the call. The parsing buffer receives the call's parameters when the operator invokes an HI command.

Syntax, PL/M and C

```
CALL rq$c$backup$char (except_ptr);
```

```
rq_c_backup_char (except_ptr);
```

Parameter	PL/M Data Type	C Data Type
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameter

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_LIMIT	0004H	The parsing buffer's pointer is at the start of the command.
E_CONTEXT	0005H	The calling job is not an I/O job.

c_create_command_connection

Returns a token for an iRMX command connection object. This object is required in order to invoke commands from a program using the **c_send_command** system call.

See also: **c_send_command**

Syntax, PL/M and C

```
command_conn = rq$c$create$command$connection (default_ci,
      default_co, flags, except_ptr);

command_conn = rq_c_create_command_connection (default_ci,
      default_co, flags, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
command_conn	SELECTOR	SELECTOR
default_ci	SELECTOR	SELECTOR
default_co	SELECTOR	SELECTOR
flags	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

command_conn

A token for the new command connection.

Parameters

default_ci

A token for a connection used as the *:ci:* (console input) for any commands invoked using this command connection.

default_co

A token for a connection used as the *:co:* (console output) for any commands invoked using this command connection.

`flags`

Indicates if the HI should return an E_ERROR_OUTPUT condition code if the system call **c_send_eo_response** is used by any task.

Value	**Meaning**
0	Do not return a code.
1	Return the condition code.

See also: HI CLI, *System Concepts*

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Although a job can contain multiple command connections, the tasks in a job cannot create command connections simultaneously. Attempts to do this result in an E_CONTEXT condition code. Only one task should create the command connections for all tasks in the job.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_ALREADY_ATTACHED	0038H	While creating a stream file, the EIOS was unable to attach the *:stream:* device because another task in the same job had already invoked a BIOS call to attach the device.
E_CONTEXT	0005H	At least one of these is true: • Two command connections were being created simultaneously by two tasks in the same job. • The calling task's job was not created by the HI. See also: I/O jobs, *System Concepts*
E_DEV_DETACHING	0039H	The *:stream:* device, the default_ci device, or the default_co device was being detached.
E_DEVFD	0022H	The EIOS attempted the physical attachment of the *:stream:* device. This device had formerly been only logically attached. The EIOS found that the device and the device driver specified in the logical attachment are incompatible. The *:stream:* device is not properly configured.

E_EXIST	0006H	The default_ci or default_co parameter is not a token for an existing object.
E_FNEXIST	0021H	The *:stream:* file does not exist or is marked for deletion.
E_IFDR	002FH	The EIOS attempted to obtain information about the default_ci or default_co connection. This resulted in an invalid file driver request.
E_INVALID_FNODE	003DH	The fnode associated with the file being used for the redirected *:ci:* or *:co:* information is invalid.
E_IO_MEM	0042H	The BIOS job does not currently have a block of memory large enough to allow the HI to create a stream file.
E_LIMIT	0004H	At least one of these is true: • The object directory of the calling task's job has already reached the maximum size. • The calling task's job has exceeded its object limit. • The calling task's job or that job's default user object is already involved in 255 I/O operations. • The calling task's job was not created by the HI. See also: I/O jobs, *System Concepts*
E_LOG_NAME_NEXIST	0045H	The call was unable to find the logical name *:stream:* in the object directories of the local job, the global job, or the root job.
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's job does not have a valid default prefix.
E_NOT_CONNECTION	8042H	The default_ci or default_co parameter is a token for an object that is not a connection to a file.
E_NOT_LOG_NAME	8040H	The logical name *:stream:* refers to an object that is not a file or device connection.
E_NOUSER	8021H	The calling task's job does not have a valid default user object.

E_PARAM	8004H	The system call forced the EIOS to attempt the physical attachment of the *:stream:* device, which had formerly been only logically attached. The physical attachment is not possible; the stream file driver is not properly configured.
E_SUPPORT	0023H	The default_ci or default_co device connection was not created by this job.

c_delete_command_connection

Deletes a command connection object from a previous **c_create_command_connection** call and frees the memory used by the connection.

Syntax, PL/M and C

```
CALL rq$c$delete$command$connection (command_conn, except_ptr);
```

```
rq_c_delete_command_connection (command_conn, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`command_conn`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`command_conn`

A token for a valid command connection.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The command_conn parameter is not a token for an existing object.
E_TYPE	8002H	The command_conn parameter is a not token for a command connection object.

c_format_exception

Creates a default message for a given condition code and writes that message into a user-provided STRING.

Syntax, PL/M and C

```
CALL rq$c$format$exception (buff_ptr, buff_max, exception_code,
      reserved_byte, except_ptr);
```

```
rq_c_format_exception (buff_ptr, buff_max, exception_code,
      reserved_byte, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
buff_ptr	POINTER	STRING far *
buff_max	WORD_16	UINT_16
exception_code	WORD_16	UINT_16
reserved_byte	BYTE	UINT_8
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

`buff_ptr`

A pointer to a STRING where the HI concatenates the formatted exception message.

`buff_max`

Specifies the maximum number of bytes that may be contained in the STRING pointed to by `buff_ptr`.

`exception_code`

The condition code value for which a message is to be created.

`reserved_byte`

Reserved. Set to 1.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The call concatenates the message to the end of the STRING pointed to by the `buff_ptr` pointer and updates the count byte to reflect the addition. If a STRING is not already present in the buffer, the first byte of the buffer must be 0. The message added by **c_format_exception** will not be longer than 30 characters, not including the length byte.

The condition code message created by **c_format_exception** consists of the condition code value and condition code mnemonic in this format:

```
value : mnemonic
```

The mnemonics are provided by the HI from an internal table.

See also: Internal table, *Command Reference*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_PARAM	8004H	An undefined condition code value was specified.
E_STRING	8084H	The message to be returned exceeds the length limit of 255 characters.
E_STRING_BUFFER	0081H	The buffer pointed to by the buff_ptr parameter is not large enough to contain the exception message.

c_get_char

Gets a character from the parsing buffer and moves the pointer to the next character. Consecutive calls to **c_get_char** return consecutive characters.

Syntax, PL/M and C

```
gchar = rq$c$get$char (except_ptr);

gchar = rq_c_get_char (except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`char`	BYTE	UINT_8
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`gchar` The next character from the parsing buffer. A null character returns when the end of the buffer is reached.

Parameter

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The calling task's job was not created by the HI. See also: I/O jobs, *System Concepts*
E_LIMIT	0004H	At least one of these occurred: • The object directory of the calling task's job has already reached the maximum size. • The calling task's job has exceeded its object limit. • The calling task's job was not created by the HI.
E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.

c_get_command_name

Obtains the pathname of the command entered by the operator. This information is available to each command and is stored in a separate buffer from the parsing buffer. This call does not obtain information from the parsing buffer, nor does it move the parsing buffer pointer.

Syntax, PL/M and C

```
CALL rq$c$get$command$name (path_name_ptr, name_max,
      except_ptr);
```

```
rq_c_get_command_name (path_name_ptr, name_max, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`path_name_ptr`	POINTER	STRING far *
`name_max`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`path_name_ptr`

A pointer to a STRING that receives the pathname of the current command.

`name_max`

Specifies the maximum length in bytes, including the length byte, of the STRING pointed to `path_name_ptr`.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If the operator invokes the **c_get_command_name** command without specifying a logical name, the HI automatically searches a configured number of directories for the command. In such cases, the value returned by this command also includes the directory name (such as *`:system:`*, *`:prog:`*, or *`:$:`*) as a prefix to the command name.

See also: For ICU-configurable systems, HI Logical Names screen, *ICU User's Guide and Quick Reference*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_LIMIT	0004H	The calling task's job was not created by the HI.
E_PATHNAME_SYNTAX	003EH	The specified pathname contains invalid characters.
E_STRING_BUFFER	0081H	The buffer pointed to by the path_name_ptr parameter is not large enough to contain the command name.
E_TIME	0001H	The calling task's job was not created by the HI.

c_get_input_connection

Returns an EIOS connection to the specified input file. This call causes an error message to appear at the operator's terminal (*:co:*) whenever the OS encounters an exceptional condition. This condition can be one of those listed for this call or the EIOS calls **s_attach_file** and **s_open**.

Syntax, PL/M and C

```
connection = rq$c$get$input$connection (path_name_ptr,
      except_ptr);
```

```
connection = rq_c_get_input_connection (path_name_ptr,
      except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
connection	SELECTOR	SELECTOR
path_name_ptr	POINTER	STRING far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

connection

The token for the connection to the specified file.

Parameters

path_name_ptr

A pointer to a STRING that specifies the path and filename.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The connection obtained by **c_get_input_connection** has these attributes: read only, accessible to all users, and has two 1024-byte buffers (default size).

These messages can be displayed by this call:

`*pathname*, file does not exist`
The input file does not exist.

`*pathname*, invalid file type`
The input file was a data file and a directory was required, or vice versa.

`*pathname*, invalid logical name`
The input pathname contains a logical name longer than 12 characters, or contains unmatched colons, invalid characters, or 0 characters.

`*pathname*, logical name does not exist`
The input pathname contains a logical name that does not exist.

`*pathname*, READ access required`
The user does not have read access to the input file.

`*pathname*, *exception value*:*exception mnemonic*`
If an exceptional condition occurs when **c_get_input_connection** attempts to obtain the input connection, the *exception value* and *exception mnemonic* portions of the message indicate the condition code encountered.

See also: Condition Codes in EIOS calls **s_attach_file** and **s_open**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_ALREADY_ATTACHED	0038H	The device containing the file specified in the path_name_ptr parameter is already attached.
E_CONTEXT	0005H	The calling task's job was not created by the HI. See also: I/O jobs, *System Concepts*
E_DEV_DETACHING	0039H	The device specified in the path_name_ptr parameter is being detached.
E_DEVFD	0022H	The call attempted the physical attachment of a device that had formerly been only logically attached and the device and the device driver specified in the logical attachment were incompatible.

E_EXIST	0006H	The specified device does not exist.
E_FACCESS	0026H	The specified connection does not have read access to the file.
E_FNEXIST	0021H	At least one of these is true: • The target file does not exist or is marked for deletion. • While attaching the file pointed to by the path_name_ptr parameter, the call attempted the physical attachment of the device as a named device. The device specified when the logical attachment was made was not properly configured.
E_FTYPE	0027H	The path pointed to by the path_name_ptr parameter contained a file name that should have been the name of a directory, but is not.
E_ILLVOL	002DH	The call attempted the physical attachment of the specified device as a named device. This device had formerly been only logically attached and the volume did not contain named files.
E_INVALID_FNODE	003DH	The fnode associated with the file being used for the redirected *:ci:* or *:co:* information is invalid.
E_IO_HARD	0052H	While attempting to access the parent directory of the file pointed to by the path_name_ptr parameter, the call detected a hard I/O error. A retry is probably useless.
E_IO_MEM	0042H	The BIOS job does not currently have a block of memory large enough to allow this call to complete.
E_IO_NOT_READY	0053H	At least one of these is true: • While attempting to access the file specified in the `path_name_ptr` parameter, the call found that the device was off-line. Operator intervention is required. • Communication failed between the local system and the remote server. Operator intervention is required.

E_IO_SOFT	0051H	While attempting to access the file specified in the path_name_ptr parameter, the call detected a soft I/O error. Another try might be successful.
E_IO_UNCLASS	0050H	An unknown I/O error occurred while this call tried to access the file given in the path_name_ptr parameter.
E_LIMIT	0004H	At least one of these is true: • The calling task's job or the job's default user object is already involved in 255 I/O operations. • The calling task's job was not created by the HI. • The object limit of the calling job has been reached. • Processing this call would deplete the remote server's resources.
E_LOG_NAME_NEXIST	0045H	The pathname for the specified device contains an explicit logical name. The call was unable to find this name in the object directories of the local job, the global job, or the root job.
E_LOG_NAME_SYNTAX	0040H	The pathname pointed to by the path_name_ptr parameter contains a logical name. This logical name contains an unmatched colon, is longer than 12 characters, has 0 characters, or contains invalid characters.
E_MEDIA	0044H	The specified device was off-line or removable media were not in place.
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's job does not have a valid default prefix.
E_NOT_LOG_NAME	8040H	The logical name specified by the path_name_ptr parameter does not refer to a file or device connection.
E_NOUSER	8021H	The calling task's job does not have a valid default user.

E_PARAM	8004H	At least one of these is true: • The system call forced the EIOS to attempt the physical attachment of the device referenced by the `path_name_ptr` parameter. This device had formerly been only logically attached. The physical attachment is not possible; the file driver is not properly configured. • The connection to the specified file cannot be opened for reading.
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 specified pathname contains invalid characters.
E_SHARE	0028H	The file sharing attribute currently does not allow new connections to the file to be opened for reading.
E_STREAM_SPECIAL	003CH	The call attempted to attach a stream file with an invalid stream file request.
E_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF. The server's UDF must have World read permission.

c_get_input_pathname

Gets a pathname from the list of input pathnames in the parsing buffer.

Syntax, PL/M and C

```
CALL rq$c$get$input$pathname (path_name_ptr, path_name_max,
      except_ptr);
```

```
rq_c_get_input_pathname (path_name_ptr, path_name_max,
      except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`path_name_ptr`	POINTER	STRING far *
`path_name_max`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`path_name_ptr`

A pointer to a STRING that receives the next pathname in parsing buffer. A zero-length STRING indicates that there are no more pathnames.

`path_name_max`

Specifies the maximum length, up to 256 bytes including the length byte, of the STRING pointed to by the `path_name_ptr` parameter.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The first call to **c_get_input_pathname** retrieves the entire input pathname list and moves the parsing buffer pointer to the next parameter. **C_get_input_pathname** stores the list in an internal buffer and returns the first pathname in the STRING pointed to by the `path_name_ptr` parameter. Succeeding calls to **c_get_input_pathname** return additional pathnames from the input pathname list but do not move the parsing buffer pointer.

C_get_input_pathname accepts wildcard characters in the last component of a pathname; it treats such a pathname as a list of pathnames. To obtain each pathname, it searches in the parent directory of the component containing the wildcard, comparing the wildcard name with the names of all files in the directory. It returns the next pathname that matches.

The pathname returned by **c_get_input_pathname** can be used for any purpose. It is most often used in a call to **c_get_input_connection**, to obtain a connection.

See also: **c_get_input_connection**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_ALREADY_ATTACHED	0038H	The device containing the file pointed to by the path_name_ptr parameter is already attached.
E_CONTEXT	0005H	At least one of these is true: • The calling task's job was not created by the HI. See also: I/O jobs, *System Concepts* • The task called **c_get_output_pathname** before calling **c_get_input_pathname**. See also: **c_get_output_pathname**
E_DEV_DETACHING	0039H	The device pointed to by the path_name_ptr parameter is being detached.
E_DEVFD	0022H	The EIOS attempted the physical attachment of a device that had formerly been only logically attached. The EIOS found that the device and the device driver specified in the logical attachment were incompatible.

E_EXIST	0006H	At least one of these is true: • The connection to the parent directory of the file pointed to by the `path_name_ptr` parameter is not a token for the existing job. • The calling task's job was not created by the HI.
E_FACCESS	0026H	The connection used to open the directory does not have read access to the directory.
E_FLUSHING	002CH	The device containing the directory was being detached.
E_FNEXIST	0021H	At least one of these is true: • The target file does not exist or is marked for deletion. • While attaching the parent directory of the file pointed to by the path_name_ptr parameter, the I/O System attempted the physical attachment of the device as a named device. The device specified when the logical attachment was made was not properly configured.
E_FTYPE	0027H	The path pointed to by the path_name_ptr parameter contained a file name that should have been the name of a directory, but is not.
E_IFDR	002FH	The specified file is a stream or physical file.
E_ILLVOL	002DH	The call attempted the physical attachment of the specified device as a named device. This device had formerly been only logically attached and the volume did not contain named files.
E_INVALID_FNODE	003DH	The fnode associated with the file being used for the redirected *:ci:* or *:co:* information is invalid.
E_IO_HARD	0052H	While attempting to access the parent directory of the file pointed to by the path_name_ptr parameter, the call detected a hard I/O error. A retry is probably useless.
E_IO_MEM	0042H	The BIOS job does not currently have a block of memory large enough to allow this call to complete.

E_IO_NOT_READY	0053H	At least one of these is true: • While attempting to access the file specified in the `path_name_ptr` parameter, the call found that the device was off-line. Operator intervention is required. • Communication failed between the local system and the remote server. Operator intervention is required.
E_IO_SOFT	0051H	While attempting to access the file specified in the path_name_ptr parameter, the call detected a soft I/O error. Another try might be successful.
E_IO_UNCLASS	0050H	An unknown I/O error occurred while this call tried to access the parent directory of the file pointed to by the path_name_ptr parameter.
E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its object limit. • The calling task's job or the job's default user object is already involved in 255 I/O operations. • The calling task's job was not created by the HI. • Processing this call would deplete the remote server's resources.
E_LIST	0085H	The last value of the input pathname list is missing. For example: *able,baker,* has no value following the second comma.
E_LOG_NAME_NEXIST	0045H	The pathname for the specified device contains an explicit logical name. The call was unable to find this name in the object directory of the local job, the global job, or the root job.
E_LOG_NAME_SYNTAX	0040H	The pathname pointed to by the path_name_ptr parameter contains a logical name that has an unmatched colon, is longer than 12 characters, has 0 characters, or contains invalid characters.
E_MEDIA	0044H	The specified device was off-line or removable media were not in place.

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's job does not have a valid default prefix.
E_NOT_LOG_NAME	8040H	The logical name specified by the path_name_ptr parameter does not refer to a file or device connection.
E_NOUSER	8021H	The calling task's job does not have a valid default user object.
E_PARAM	8004H	At least one of these is true: • The EIOS attempted the physical attachment of the device pointed to by the `path_name_ptr` parameter. This device had formerly been only logically attached. The physical attachment is not possible; the file driver is not properly configured. • The connection to the parent directory cannot be opened for reading.
E_PARSE_TABLES	8080H	The call detected an error in an internal table used by the HI.
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 specified pathname contains invalid characters.
E_SHARE	0028H	The connection to the parent directory cannot be opened for reading.
E_STREAM_SPECIAL	003CH	The EIOS issued an invalid stream file request when an attempt to attach a stream file failed.
E_STRING	8084H	The pathname to be returned exceeds the length limit of 255 characters.
E_STRING_BUFFER	0081H	The buffer pointed to by the path_name_ptr parameter was not large enough for the pathname to return.

E_SUPPORT	0023H	This call attempted to read the parent directory of the pathname pointed to by the path_name_ptr parameter. The file driver corresponding to that directory does not support this operation.
E_WILDCARD	0086H	The pathname to be returned contains an invalid wildcard specification.
E_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF. The server's UDF must have World read permission.

c_get_output_connection

Parses the command line and returns an EIOS connection to the requested output file.

Syntax, PL/M and C

```
connection = rq$c$get$output$connection (path_name_ptr,
      preposition, except_ptr);
```

```
connection = rq_c_get_output_connection (path_name_ptr,
      preposition, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection	SELECTOR	SELECTOR
path_name_ptr	POINTER	STRING far *
preposition	BYTE	UINT_8
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

`connection`

A connection to the output file.

Parameters

`path_name_ptr`

A pointer to a STRING containing the pathname of the file to be accessed.

`preposition`

Defines which preposition to use to create the output file. Use these values to specify the preposition mode:

Value	Meaning
0	Use the preposition returned by the last **c_get_output_pathname** call
1	TO
2	OVER
3	AFTER
4-255	Reserved, results in an error

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The connection obtained by **c_get_output_connection** is open for writing and has these attributes: write only, accessible to all, and has 2 1024-byte buffers.

If the call to **c_get_output_connection** specifies the TO preposition and the output file already exists, **c_get_output_connection** issues this message to the terminal (*:co:*):

pathname, already exists, OVERWRITE?

If the operator enters Y, y, R, or r, **c_get_output_connection** returns a connection to the existing file, enabling the command to write over the file. Any other response causes **c_get_output_connection** to return an E_FACCESS condition code.

C_get_output_connection causes an error message to appear at the operator's terminal (*:co:*) whenever an exceptional condition occurs. The exceptional condition that causes the error message can be one of those listed below or one associated with an EIOS call. These messages can occur:

pathname, DELETE access required
The user does not have delete access to an existing file.

pathname, directory ADD entry access required
The user does not have add entry access to the parent directory.

pathname, file does not exist
The output file does not exist.

pathname, invalid file type
The output file was a data file and a directory was required, or vice versa.

pathname, invalid logical name
The output pathname contains a logical name longer than 12 characters, contains unmatched colons, contains invalid characters, or 0 characters.

pathname, logical name does not exist
The output pathname contains a logical name that does not exist.

pathname, *exception value*:*exception mnemonic*
If an exceptional condition occurs when **c_get_output_connection** attempts to obtain the output connection, the *exception value* and *exception mnemonic* portions of the message indicate the condition code encountered.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_ALREADY_ATTACHED	0038H	The EIOS was unable to attach the device containing the file because the BIOS has already attached the device.
E_CONTEXT	0005H	The calling task's job was not created by the HI.
E_DEV_DETACHING	0039H	The device referred to by the path_name_ptr parameter was being detached.
E_DEVFD	0022H	The call attempted the physical attachment of a device that had formerly been only logically attached and the device and the device driver specified in the logical attachment were incompatible.
E_EXIST	0006H	The connection parameter for the device containing that file is not a token for an existing object.
E_FACCESS	0026H	At least one of these is true: • The default user for the calling task's job did not have update access to an existing file and/or add-entry access to the parent directory. • The TO or OVER preposition was specified and the default user for the calling task's job could not truncate the file.
E_FNEXIST	0021H	At least one of these is true: • The target file does not exist or is marked for deletion. • While attaching the file pointed to by the path_name_ptr parameter, the EIOS attempted the physical attachment of the device as a named device. The device specified when the logical attachment was made was not properly configured.
E_FTYPE	0027H	The path pointed to by the path_name_ptr parameter contained a file name that should have been the name of a directory, but is not.

E_IFDR	002FH	The call requested information about the specified file, but the request was an invalid file driver request.
E_ILLVOL	002DH	The call attempted the physical attachment of the specified device as a named device. This device had formerly been only logically attached and the volume did not contain named files.
E_INVALID_FNODE	003DH	The fnode associated with the file being used for the redirected *:ci:* or *:co:* information is invalid.
E_IO_HARD	0052H	While attempting to access the file specified in the path_name_ptr parameter, the call detected a hard I/O error. A retry is probably useless.
E_IO_MEM	0042H	The BIOS job does not currently have a block of memory large enough to allow this call to complete.
E_IO_NOT_READY	0053H	At least one of these is true: • While attempting to access the file specified in the `path_name_ptr` parameter, the call found that the device was off-line. Operator intervention is required. • Communication failed between the local system and the remote server. Operator intervention is required.
E_IO_SOFT	0051H	While attempting to access the file specified in the path_name_ptr parameter, the call detected a soft I/O error. Another try might be successful.
E_IO_UNCLASS	0050H	An unknown I/O error occurred while this call tried to access the file given in the path_name_ptr parameter.

E_IO_WRPROT	0054H	While attempting to obtain an input connection to the file specified in the path_name_ptr parameter, this call found that the volume containing the file is write-protected. • The calling task's job or the job's default user object is already involved in 255 I/O operations. • The calling task's job was not created by the HI. • The calling task's job has reached its object limit. See also: I/O jobs, *System Concepts* • Processing this call would deplete the remote server's resources.
E_LOG_NAME_NEXIST	0045H	The specified pathname contains an explicit logical name. The call was unable to find this name in the object directory of the local job, the global job, or the root job.
E_LOG_NAME_SYNTAX	0040H	The pathname pointed to by the path_name_ptr parameter contains a logical name. The logical name contains unmatched colons, is longer than 12 characters, contains invalid characters, or contains 0 characters.
E_MEDIA	0044H	The specified device was off-line or removable media were not in place.
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's job does not have a valid default prefix.
E_NOT_LOG_NAME	8040H	The logical name specified by the path_name_ptr parameter does not refer to a file or device connection.
E_NOUSER	8021H	The calling task's job does not have a valid default user object.

E_PARAM	8004H	The system call forced the EIOS to attempt the physical attachment of the device referenced by the path_name_ptr parameter. The device had formerly been only logically attached. The physical attachment is not possible; the file driver is not properly configured.
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 specified pathname contains invalid characters.
E_PREPOSITION	0087H	One of these is true: • The command line contained a preposition value greater than 3. • The command line contained a 0 as the preposition value. This indicated that the same preposition was to be used as in the last call to **c_get_output_connection**. This is the first call to **c_get_output_connection**.
E_SHARE	0028H	The new connection cannot be opened for writing.
E_SPACE	0029H	One of these is true: • The volume is full. • The volume already contains the maximum number of files.
E_STREAM_SPECIAL	003CH	The EIOS issued an invalid stream file request when an attempt to attach a stream file failed.
E_UDF_IO	02D0H	An error occurred while accessing the remote server's UDF. The server's UDF must have World read permission.

c_get_output_pathname

Gets a pathname from the list of output pathnames in the parsing buffer.

Syntax, PL/M and C

```
preposition = rq$c$get$output$pathname (path_name_ptr,
      path_name_max, default_output_ptr, except_ptr);
```

```
preposition = rq_c_get_output_pathname (path_name_ptr,
      path_name_max, default_output_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`preposition`	BYTE	UINT_8
`path_name_ptr`	POINTER	STRING far *
`path_name_max`	WORD_16	UINT_16
`default_output_ptr`	POINTER	STRING far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`preposition`

One of these preposition type values. You can pass this value to **c_get_output_connection** when obtaining an output connection to the file.

Value	Meaning
0	The preposition returned by the last **c_get_output_pathname** call
1	TO
2	OVER
3	AFTER
4-255	Reserved

Parameters

`path_name_ptr`

A pointer to a STRING that receives the next pathname in the pathname list. A null STRING indicates that there are no more pathnames.

`path_name_max`

Specifies the maximum length, up to 256 bytes including the length byte, of the STRING pointed to by the `path_name_ptr` parameter.

`default_output_ptr`

A pointer to a STRING containing the command's default standard output. The text must specify TO, OVER, or AFTER for the output mode.
For example: `TO :co:` or `TO :lp:`

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Do not call **c_get_output_pathname** before first calling **c_get_input_pathname**.

The first call to **c_get_output_pathname** retrieves the preposition (TO/OVER/AFTER) and the entire output pathname list; it then moves the parsing buffer pointer to the next parameter. If the parsing buffer does not contain a preposition and pathname list, **c_get_output_pathname** uses the default pointed to by the `default_output_ptr` parameter and does not move the parsing buffer pointer.

After retrieving the pathname list, **c_get_output_pathname** stores it in an internal buffer, returns the first pathname in the STRING pointed to by the `path_name_ptr` parameter, and returns the preposition in the preposition parameter. Succeeding calls to **c_get_output_pathname** return additional pathnames from the output pathname list as well as the preposition, but they do not move the parsing buffer pointer.

C_get_output_pathname accepts characters with a wildcard as the last component of a pathname. It generates each output pathname based on this pathname and wildcard, the corresponding pathname and wildcard that was input to **c_get_input_pathname**, and the most recent input pathname returned by **c_get_input_pathname**.

The pathname returned by **c_get_output_pathname** can be used for any purpose. It is most often used in a call to **c_get_output_connection** to obtain a connection to the file. In such a case, **c_get_output_connection** processes the TO/OVER/AFTER preposition. If the pathname is used as input to a system call other than **c_get_output_connection**, the interpretation of the TO/OVER/AFTER preposition is the user's responsibility.

See also: **c_get_input_pathname**, **c_get_output_connection**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The calling task's job was not created by the HI.
E_DEFAULT_SO	8083H	The default output STRING pointed to by default_output_ptr contained an invalid preposition or pathname.
E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its object limit. • The calling task's job was not created by the HI. • The calling task's job or the job's default user object is already involved in 255 I/O operations.
E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E_PATHNAME_SYNTAX	003EH	The specified pathname contains invalid characters.
E_STRING	8084H	The pathname to be returned exceeds the length limit of 255 characters.
E_STRING_BUFFER	0081H	The buffer pointed to by the path_name_ptr parameter was not large enough for the pathname to return.
E_UNMATCHED_LISTS	008BH	The numbers of files in the input and output lists are not the same.
E_WILDCARD	0086H	The output pathname contains an invalid wildcard specification.

c_get_parameter

Retrieves one parameter from the parsing buffer and moves the parsing buffer pointer to the next parameter.

Syntax, PL/M and C

```
more = rq$c$get$parameter (name_ptr, name_max, value_ptr,
      value_max, index_ptr, predict_list_ptr, except_ptr);
```

```
more = rq_c_get_parameter (name_ptr, name_max, value_ptr,
      value_max, index_ptr, predict_list_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`more`	BYTE	UINT_8
`name_ptr`	POINTER	STRING far *
`name_max`	WORD_16	UINT_16
`value_ptr`	POINTER	STRING_TABLE_STRUCT far *
`value_max`	WORD_16	UINT_16
`index_ptr`	POINTER	UINT_8 far *
`predict_list_ptr`	POINTER	STRING_TABLE_STRUCT far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`more` Indicates whether or not the current call to **c_get_parameter** returned a parameter.

Value	Meaning
00H	Indicates that there are no more parameters and no parameter returned
0FFH	Indicates that a parameter returned.

Parameters

`name_ptr`

A pointer to a STRING that receives the keyword portion of the parameter. If this parameter does not contain a keyword portion, the HI returns a null STRING.

`name_max`

Specifies the maximum length, up to 256 bytes including the length byte, of the STRING pointed to by the `name_ptr` parameter.

`value_ptr`

A pointer to a STRINGTABLE that receives the value portion of the parameter. If the value portion contains a list of values separated by commas, the HI returns the values to the STRINGTABLE one value per string.

See also: Data types, STRINGTABLE, in this manual

`value_max`

Specifies the maximum length in bytes of the STRINGTABLE pointed to by the `value_ptr` parameter. The maximum length is 65535 bytes.

`index_ptr`

A pointer to location that receives an index into the STRINGTABLE pointed to by `predict_list_ptr`. This index identifies the `name_ptr` keyword as a preposition from the list of possible prepositions. If the STRINGTABLE is empty, or if the keyword name is not in the list, the system call returns 0 for the index.

`predict_list_ptr`

A pointer to a STRINGTABLE that specifies the acceptable preposition values. A null pointer indicates that you do not intend to retrieve parameters that use prepositions. Without this list, **c_get_parameter** cannot determine whether groups of characters separated by spaces are separate parameters or a single parameter that uses a preposition.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The parameter retrieved by **c_get_parameter** can be one of these:

- Keyword/value-list parameter using parentheses
- Keyword/value-list parameter using an equal sign
- Keyword/value-list parameter with the keyword as a preposition
- Value-list without a keyword

See also: Types, format, and syntax of parameters, *System Concepts*

When **c_get_parameter** retrieves a parameter from the parsing buffer, it obtains the next group of characters that are separated by spaces. These characters are checked against those in the `predict_list_ptr` list. If the characters match a value in the list, **c_get_parameter** realizes that the characters represent a preposition and not an entire parameter; it then obtains the next group of characters separated by spaces as the value portion of the parameter.

Condition Codes

<table>
<tr><td>E_OK</td><td>0000H</td><td>No exceptional conditions occurred.</td></tr>
<tr><td>E_CONTEXT</td><td>0005H</td><td>The calling task's job was not an I/O job.
See also: I/O jobs, *System Concepts*</td></tr>
<tr><td>E_CONTINUED</td><td>0083H</td><td>The call found a continuation character in the parsing buffer. Command lines should not contain continuation characters.</td></tr>
<tr><td>E_LIMIT</td><td>0004H</td><td>At least one of these is true:
• The calling task's job has already reached its object limit.
• The calling task's job was not an I/O job. See also: I/O jobs, *System Concepts*</td></tr>
<tr><td>E_LIST</td><td>0085H</td><td>At least one of these is true:
• The parameter contains an unmatched parenthesis.
• A value in the value list is missing or an improper value was entered, for example
<table>
<tr><th>Value</th><th>Comments</th></tr>
<tr><td>A,B,</td><td>No value following second comma.</td></tr>
<tr><td>A,B=C,D</td><td>The equal sign must be between quotes. 'B=C' is valid.</td></tr>
<tr><td>A,B(C,E),F</td><td>The parentheses must be between quotes or set off by commas. A,B,(C,E),F is valid.</td></tr>
</table></td></tr>
<tr><td>E_LITERAL</td><td>0080H</td><td>The call found a literal (quoted string) in the parsing buffer with no closing quote.</td></tr>
<tr><td>E_MEM</td><td>0002H</td><td>The memory available to the calling task's job is not sufficient to complete the call.</td></tr>
<tr><td>E_PARAM</td><td>8004H</td><td>The predict_list_ptr parameter pointed to a STRINGTABLE, but the index_ptr parameter was set to 0.</td></tr>
<tr><td>E_PARSE_TABLES</td><td>8080H</td><td>The call found an error in an internal table used by the HI.</td></tr>
<tr><td>E_SEPARATOR</td><td>0082H</td><td>The call found an invalid command separator in the parsing buffer. These are invalid command separators: ><, <>, ||, |, [, and].</td></tr>
</table>

E_STRING	8084H	The STRING returned as the parameter name or one of the parameter values exceeds 255 characters.
E_STRING_BUFFER	0081H	The STRING returned as the parameter name or one of the parameter values exceeds the buffer size provided in the call.

c_send_command

Stores a command line in the command connection created by the **c_create_command_connection** call, concatenates the command line with any others already stored there, and (if the command invocation is complete) invokes the command. The command can be any standard HI command or a command that you create. Use this system call to invoke a command from a program.

See also: **c_create_command_connection**

Syntax, PL/M and C

```
CALL rq$c$send$command (command_conn, line_ptr,
      command_except_ptr, except_ptr);
```

```
rq_c_send_command (command_conn, line_ptr, command_except_ptr,
      except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`command_conn`	SELECTOR	SELECTOR
`line_ptr`	POINTER	STRING far *
`command_except_ptr`	POINTER	UINT_16 far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`command_conn`
A token for the command connection that receives the command line.

`line_ptr`
A pointer to a STRING containing a command line to execute.

`command_except_ptr`
A pointer to a location where the condition code indicating the status of the invoked command returns. This parameter is undefined if an E_OK condition code is not returned to the location pointed to by `except_ptr`.

`except_ptr`
A pointer to a variable declared by the application where the call returns the condition code indicating the status of the **c_send_command.**

Additional Information

A command invocation can contain several & (continuation marks), indicating that the command line is continued on the next line. In this case, the HI returns an E_CONTINUED condition code and does not invoke the command. Call **c_send_command** as often as needed to send the continuation lines.

C_send_command concatenates the original command line and all continuation lines into a single command line in the command connection. It removes all continuation marks and comments from this command line.

See also: Continuing input lines and comments, *Command Reference*

When the command invocation is complete, the HI parses the command pathname from the command line. If no exception conditions occur, the HI requests the AL to load and execute the command.

NOTE When a **c_send_command** call is made, the HI sets the <Ctrl-C> semaphore to the default HI <Ctrl-C> handler. If you previously set the <Ctrl-C> handler, it must be set again after making this call.

See also: **rq_c_set_control_c** system call,
<Ctrl-C>, *System Concepts*

⚠ **CAUTION**

Do not use this system call to launch any commands that require user input. The request for input does not get redirected to the user. See the list below for commands that you cannot launch, or cannot launch if you use a form of the command that requires input.

For example, you can use the **copy** command with the `over` parameter, because it will copy over any existing files without question. But you cannot use the **copy** command with the `to` parameter, because if a file exists with the same name, the command prompts the user whether to overwrite the file. The user will not receive the prompt when the command is launched from **rq_c_send_command**, so the command never completes.

Never launch these commands	**Launch only forms that do not require input**		
backup	accounting	deletedir	locdata
pause	addloc	dir	permit
restore	copy	diskverify	remini
psh (without any parameter)	copydir	format	rename
telnet	date	ftp	time
rlogin	delete	help	
tftp			

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_ALREADY_ATTACHED	0038H	The EIOS was unable to attach the device containing the object file because the BIOS has already attached the device.
E_BAD_GROUP	0061H	The object file represented by the command's pathname contained an invalid group definition record.
E_BAD_HEADER	0062H	The object file represented by the command's pathname does not begin with a header record for a loadable object module.
E_BAD_SEGDEF	0063H	The object file represented by the command's pathname contains an invalid segment definition record.
E_CHECKSUM	0064H	At least one record of the object file represented by the command's pathname contains a checksum error. This can occur if the CHECKSUM amount calculated during the read operation did not match the CHECKSUM field of the record being read.
E_CONTEXT	0005H	The calling task's job was not created by the HI.
E_CONTINUED	0083H	The OS detected a continuation character while scanning the command line pointed to by the line_ptr parameter. **C_send_command** must be invoked again to send the next portion of the command.
E_DEV_DETACHING	0039H	The device containing the object file was being detached.
E_DEVFD	0022H	The EIOS attempted the physical attachment of a device that had formerly been only logically attached. The EIOS found that the device and the device driver specified in the logical attachment were incompatible.
E_EOF	0065H	The AL encountered an unexpected EOF on the object file represented by the command's pathname.

E_EXIST	0006H	At least one of these is true: • The call detached the device containing the object file before completing the loading operation. • The command_conn parameter is not a token for a command connection.
E_FACCESS	0026H	The default user for the calling task's job does not have read access to the object file.
E_FLUSHING	002CH	The device containing the object file was being detached.
E_FNEXIST	0021H	At least one of these is true: • The file in the command's pathname is either marked for deletion or does not exist. • While attaching the file specified in the line_ptr parameter, the EIOS attempted the physical attachment of the device as a named device. The device specified when the logical attachment was made was not properly configured.
E_FTYPE	0027H	The path pointed to by the line_ptr parameter contained a file name that should have been the name of a directory, but is not. Except for the last component, each file in a pathname must be a named directory.
E_ILLVOL	002DH	The call attempted the physical attachment of the specified device as a named device. This device had formerly been only logically attached and the volume did not contain named files.
E_INVALID_FNODE	003DH	The fnode associated with the file being used for the redirected *:ci:* or *:co:* information is invalid.
E_IO_HARD	0052H	While attempting to access the object file, this call detected a hard I/O error.
E_IO_MEM	0042H	The BIOS job does not currently have a block of memory large enough to allow this call to complete.
E_IO_NOT_READY	0053H	While attempting to access the object file, this call found that the device was off-line. Operator intervention is required.

E_IO_SOFT	0051H	While attempting to access the file specified in the line_ptr parameter, the call detected a soft I/O error. Another try might be successful.
E_IO_UNCLASS	0050H	An unknown I/O error occurred while this call tried to access the object file.
E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its object limit. • The calling task's job, or the job's default user object, is already involved in 255 I/O operations. • The new I/O job, or its default user, is already involved in 255 I/O operations. • The calling task's job was not created by the HI. See also: I/O jobs, *System Concepts*
E_LITERAL	0080H	The call found a literal (quoted string) with no closing quote while scanning the contents of the command line pointed to by the line_ptr parameter.
E_LOG_NAME_NEXIST	0045H	The command's pathname contains an explicit logical name, but the call was unable to find this name in the object directory of the local job, the global job, or the root job.
E_LOG_NAME_SYNTAX	0040H	The pathname pointed to by the line_ptr parameter contains a logical name. The logical name contains an unmatched colon, is longer than 12 characters, has 0 characters, or contains invalid characters.
E_MEDIA	0044H	The specified device was off-line or removable media were not in place.
E_MEM	0002H	The memory available to the calling task's job, the new I/O job, or the BIOS job is not sufficient to complete the call.

E_NO_LOADER_MEM	0067H	At least one of these is true: • The memory pool of the newly created I/O job does not currently have a block of memory large enough to allow the AL to run. • The memory pool of the BIOS' job does not currently have a block of memory large enough to allow the AL to run.
E_NOPREFIX	8022H	The calling task's job does not have a valid default prefix.
E_NO_START	006CH	The object file represented by the command pathname does not specify the entry point for the program being loaded.
E_NOT_CONNECTION	8042H	The default_ci or default_co parameter is a token for an object that is not a file connection.
E_NOT_LOG_NAME	8040H	The command pathname contains a logical name of an object that is neither a device connection nor a file connection.
E_NOUSER	8021H	The calling task's job does not have a valid default user.
E_PARAM	8004H	The EIOS attempted the physical attachment of a device containing the object file. This device had formerly been only logically attached. The physical attachment is not possible; the file driver is not properly configured.
E_PARSE_TABLES	8080H	The call found an error in an internal table.
E_PATHNAME_SYNTAX	003EH	The command's pathname contains invalid characters.
E_REC_FORMAT	0069H	At least one record in the object file contains a record format error.
E_REC_LENGTH	006AH	The object file contains a record longer than the AL's configured maximum record length.
E_REC_TYPE	006BH	At least one of these is true: • At least one record in the file being loaded is of a type that the AL cannot process. • The AL has encountered records in a sequence that it cannot process.

E_SEG_BOUNDS	0070H	The AL created multiple segments in which to load information. One of the data records in the object file specified a load address outside of the created segments.
E_SEPARATOR	0082H	The call found an invalid separator while scanning the command line. The invalid command separators are: ><, <>, ||, |, [, and].
E_STRING	8084H	The STRING returned as the parameter name or one of the parameter values exceeds 255 characters.
E_STRING_BUFFER	0081H	The size of the command's pathname exceeds the size of the command name buffer specified during HI configuration.
E_TIME	0001H	The calling task's job was not created by the HI.
E_TYPE	8002H	The command_conn parameter is a token for an object that is not a command connection.

c_send_co_response

Sends messages to *:co:* and receives messages from *:ci:*; commands such as **submit** can redirect this input from *:ci:* and output to *:co:* to a file.

Syntax, PL/M and C

```
CALL rq$c$send$co$response (response_ptr, response_max,
      message_ptr, except_ptr);
```

```
rq_c_send_co_response (response_ptr, response_max, message_ptr,
      except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`response_ptr`	POINTER	STRING far *
`response_max`	WORD_16	UINT_16
`message_ptr`	POINTER	STRING far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`response_ptr`

A pointer to a STRING that receives the operator's response from *:ci:*.

`response_max`

Specifies the maximum length in bytes of the STRING pointed to by the `response_ptr` parameter. The value in `response_max` must be equal to the length of the STRING plus 1. If `response_max` is 0 or 1, no response from *:ci:* will be requested; control returns to the calling task immediately.

`message_ptr`

A pointer to a STRING containing the message to be sent to *:co:*. If null, no message is sent.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The operations performed by **c_send_co_response** depend on the values of the `message_ptr` and `response_max` parameters, as follows:

message_ptr	**response_max**	**Action**
null	0	Perform no I/O
null	not 0	Send no message, wait for input
NOT null	not 0	Send message, wait for input
NOT null	0	Send message, don't wait

If **c_send_co_response** requests a response from *:ci:*, output from other tasks can appear at *:co:* while the system waits for a response from *:ci:*.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The calling task's job was not created by the HI.
E_CONN_OPEN	0035H	At least one of these is true: • The connection to *:ci:* was not open for reading or the connection to *:co:* was not open for writing. • The connection to *:ci:* or *:co:* was not open. • The connection to *:ci:* or *:co:* was opened with **a_open**, not **s_open**. See also: BIOS call **a_open**, EIOS call **s_open**
E_EXIST	0006H	The token for *:ci:* or *:co:* is not a token for an existing object.
E_FLUSHING	002CH	The device containing the *:ci:* and *:co:* files was being detached.
E_IO_HARD	0052H	While attempting to access the *:ci:* or *:co:* file, the OS detected a hard I/O error.
E_IO_NOT_READY	0053H	While attempting to access the *:ci:* or *:co:* file, this call found that the device was off-line. Operator intervention is required.
E_IO_SOFT	0051H	While attempting to access the *:ci:* or *:co:* file, this call detected a soft I/O error. Another try might be successful.

E_IO_UNCLASS	0050H	An unknown I/O error occurred while this call tried to access the *:ci:* or *:co:* file.
E_IO_WRPROT	0054H	While attempting to obtain a connection to the *:co:* file, this call found that the volume containing the file is write-protected.
E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its object limit. • The calling task's job, or the job's default user object, is already involved in 255 I/O operations. • The calling task's job was not created by the HI.
E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E_NOT_CONNECTION	8042H	The call obtained a token for an object that should have been a connection to *:ci:* or *:co:*, but was not a file connection.
E_PARAM	8004H	The call attempted to write beyond the end of a physical file.
E_SPACE	0029H	One of these is true: • The output volume is full. • The call attempted to write beyond the end of a physical file.
E_STREAM_SPECIAL	003CH	When attempting to read or write to *:ci:* or *:co:*, the EIOS issued an invalid stream file request.
E_SUPPORT	0023H	The connection to *:ci:* or *:co:* was not created by this job.
E_TIME	0001H	The calling task's job was not created by the HI.

c_send_eo_response

Sends messages to and receives messages from the operator's terminal; input and output cannot be redirected to another device.

Syntax, PL/M and C

```
CALL rq$c$send$eo$response (response_ptr, response_max,
      message_ptr, except_ptr);
```

```
rq_c_send_eo_response (response_ptr, response_max, message_ptr,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
response_ptr	POINTER	STRING far *
response_max	WORD_16	UINT_16
message_ptr	POINTER	STRING far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

response_ptr

A pointer to a STRING that receives the operator's response from the terminal.

response_max

Specifies the maximum length in bytes of the STRING pointed to by the response_ptr parameter. The value must equal the STRING length plus 1. If response_max is 0 or 1, no response from the operator's terminal will be requested; control returns to the calling task immediately.

message_ptr

A pointer to a buffer containing the message to be sent to the operator's terminal. If null, no message is sent.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The operations performed by **c_send_eo_response** depend on the values of the `message_ptr` and `response_max` parameters:

message_ptr	**response_max**	**Action**
null	0	Perform no I/O
null	not 0	Send no message, wait for input
NOT null	not 0	Send message, wait for input
NOT null	0	Send message, don't wait

If **c_send_eo_response** requests a response from the terminal, no other output can appear at the terminal until **c_send_eo_response** receives a line terminator from the operator. The operator can choose to ignore the displayed message by entering a line terminator only.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONN_OPEN	0035H	At least one of these is true: • Either the connection to the operator's terminal was not open for reading or it was not open for writing. • The connection to the operator's terminal was not open. • The connection to the operator's terminal was opened with **a_open** not **s_open**.
E_CONTEXT	0005H	The calling task's job was not created by the HI.
E_ERROR_OUTPUT	8085H	The method used to call **send_eo_response** was invalid.
E_EXIST	0006H	The token values for the operator's terminal are not for existing objects.
E_FLUSHING	002CH	The operator's terminal was being detached.
E_IO_NOT_READY	0053H	While attempting to access the terminal, this call found that the device was off-line. Operator intervention is required.

E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its object limit. • The calling task's job or the job's default user object is already involved in 255 I/O operations. • The calling task's job was not created by the HI.
E_MEM	0002H	The memory pool of the calling task's job does not currently have a block of memory large enough to allow this system call to complete.
E_NOT_CONNECTION	8042H	The call obtained a token for an object that should have been a connection to the operator's terminal, but was not.
E_PARAM	8004H	The call attempted to write beyond the end of a physical file.
E_STREAM_SPECIAL	003CH	When attempting to read or write to the operator's terminal, the EIOS issued an invalid stream file request.
E_SUPPORT	0023H	The connection to the terminal was not created by this job.
E_TIME	0001H	The calling task's job was not created by the HI.

c_set_control_c

Changes the default response to <Ctrl-C> entered at the keyboard to a response that meets the needs of the calling task.

Syntax, PL/M and C

```
CALL rq$c$set$control$c (control_c_semaphore, except_ptr);
```

```
rq_c_set_control_c (control_c_semaphore, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
control_c_semaphore	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

control_c_semaphore

A token for a user-created semaphore that will receive units when a <Ctrl-C> is typed on the console keyboard.

⟹ **Note**

When a **c_send_command** call is made, the HI sets the <Ctrl-C> semaphore to the default HI <Ctrl-C> handler. If you previously set the <Ctrl-C> handler, it must be set again after making this call.

See also: <Ctrl-C>, *System Concepts*

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The HI's default <Ctrl-C> action is to delete the acting job, for example, any HI command.

One unit is sent to the semaphore specified by `control_c_semaphore` each time a <Ctrl-C> is typed. Any units sent to the semaphore that exceed the maximum allowable number are ignored.

A job running in background mode cannot set <Ctrl-C>.

If you use **rq_c_set_control_c** to establish a <Ctrl-C> semaphore before making UDI calls such as **dq_attach**, handling reverts to the UDI default <Ctrl-C> handler. To establish a <Ctrl-C> handler from within a UDI program, use **dq_trap_cc**.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The calling task's job was not an I/O job. See also: I/O jobs, *System Concepts*
E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its limit. • The calling task's job was not created by the HI. • The calling task's job or the job's default user object is already involved in 255 I/O operations.
E_TYPE	8002H	The token given in the parameter control_c_semaphore is not a token for a semaphore.

c_set_parse_buffer

Parses the contents of a buffer other than the command line buffer whenever the parsing system calls are used.

Syntax, PL/M and C

```
off_set = rq$c$set$parse$buffer (buff_ptr, buff_max,
      except_ptr);

off_set = rq_c_set_parse_buffer (buff_ptr, buff_max,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
off_set	WORD_16	UINT_16
buff_ptr	POINTER	STRING far *
buff_max	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

off_set

The offset into the previous parsing buffer that identifies the last byte parsed.

Parameters

buff_ptr

A pointer to a STRING containing the text to be parsed. If a null pointer, the buffer used for parsing reverts to the command line buffer and the buff_max parameter is ignored.

buff_max

Specifies the length in bytes of the STRING pointed to by the buff_ptr parameter.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Only one parsing buffer per job can be active at any given time.

This call sets the parsing buffer pointer to the beginning of the specified buffer and identifies the last byte parsed in the previous parsing buffer. This gives you the ability to change buffers at will after successive calls to **c_get_char**.

C_set_parse_buffer does not affect the buffer from which **c_get_input_pathname** and **c_get_output_pathname** retrieve pathnames. These system calls always obtain their pathnames from the command line.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The calling task's job was not created by the HI. See also: I/O jobs, *System Concepts*
E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its object limit. • The calling task's job was not created by the HI.
E_MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.

□ □ □

Nucleus System Calls 6

accept_control

Provides control of a region only if access is immediately available.

Syntax, PL/M and C

```
CALL rq$accept$control (region, except_ptr);

rq_accept_control (region, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`region`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`region`

A token for the target region.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If access is not immediately available, the E_BUSY condition code returns and the calling task remains ready.

Tasks that use regions cannot be deleted while they control the region. Once a task is in control of a region, it should not suspend or delete itself. Doing so locks the region and prevents other tasks from gaining access. Relinquish control by invoking **send_control**.

See also: **create_region,**
create_region example, Nucleus examples,
Regions, mutual exclusion, deadlock in *System Concepts*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BUSY	0003H	Another task currently has access to the protected data.
E_CONTEXT	0005H	The calling task currently has access to the region in question.
E_EXIST	0006H	The region parameter is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration. This code is returned if you make this call as an RTE call from Windows instead of from DOS.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_TYPE	8002H	The region parameter is not a token for a region.

add_reconfig_mailbox

Specifies a mailbox that will receive failure messages generated by the watchdog timer, so that the task can be notified of board failures in a Multibus II system.

Syntax, PL/M and C

```
CALL rq$add$reconfig$mailbox (mailbox, except_ptr);
```

```
rq_add_reconfig_mailbox (mailbox, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`mailbox`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`mailbox`

A token for a data mailbox created by the application, which will receive notification if another board fails in the system. The notification message sent to the mailbox has this structure:

```
DECLARE WD_MAILBOX_MESSAGE_STRUC  STRUCTURE (
   host_incar               WORD_16,
   type                     BYTE);
```

or

```
typedef struct {
   UINT_16                  host_incar;
   UINT_8                   type;
} WD_MAILBOX_MESSAGE_STRUC;
```

Where:

`host_incar`

The lower 12 bits of this value is the slot ID of the host to which this message applies (range 0-20). The upper 4 bits is the incarnation number of the host's latest existence message. See the `type` field to determine whether the incarnation number is for a failed host or is the new incarnation of a reset host.

`type` One of the following values indicates whether the message is a remote host failure or remote host reset:

Value	**Meaning**
WD_HOST_FAILURE	The watchdog timer expired without receiving an existence message from this host. This incarnation has failed.
WD_HOST_RESET	The incarnation number in the received existence message is not the same as previously received from this host, indicating that the host was reset. This incarnation is new.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This call takes advantage of the iRMX watchdog timer mechanism in a Multibus II system. Before making this call the application must create the mailbox with an **rq_create_mailbox** call. Set the flags to create a data mailbox that will use the **send_data** and **receive_data** system calls.

Configure parameters for the watchdog timer on the MBII screen of the ICU. Specify the maximum number of reconfiguration mailboxes that will be used on this board in the WDP parameter of that screen. In addition to reconfiguration mailboxes used by your application, the ARC server and each offboard client of the ARC server use one mailbox each.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_LIMIT	0004H	The maximum configured number of reconfiguration mailboxes is already in use. Increase the limit in the WDP parameter of the MBII screen.
E_TYPE	8002H	The mailbox is not a data mailbox.

alter_composite

Replaces components of composite objects.

Syntax, PL/M and C

```
CALL rq$alter$composite (extension, composite, component_index,
      replacing_obj, except_ptr);
```

```
rq_alter_composite (extension, composite, component_index,
      replacing_obj, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`extension`	SELECTOR	SELECTOR
`composite`	SELECTOR	SELECTOR
`component_index`	WORD_16	UINT_16
`replacing_obj`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`extension`

A token for the extension type object used by the composite object being altered.

`composite`

A token for the composite object being altered.

`component_index`

The position of the target token in the list of components. Values start with location 1.

`replacing_obj`

A token for the replacement component object or a null selector.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Any component in a composite object can be replaced either with a token for another object or with a place holding null selector that represents no object.

See also: CAUTION in **create_composite**, Component objects, composite objects, extension objects, and type manager in *System Concepts*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The composite parameter is not compatible with the extension parameter.
E_EXIST	0006H	The extension, composite, or replacing_obj parameter(s) is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	One or both of the extension or composite parameters is not a token for an object of the correct type.
E_PARAM	8004H	The component_index parameter refers to a nonexistent position in the component object list.

attach_buffer_pool

Makes a buffer pool's memory resources available to one or more ports.

Syntax, PL/M and C

```
CALL rq$attach$buffer$pool (buffer_pool_tkn, port_tkn,
      except_ptr);
```

```
rq_attach_buffer_pool (buffer_pool_tkn, port_tkn, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
buffer_pool_tkn	SELECTOR	SELECTOR
port_tkn	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

`buffer_pool_tkn`

A token identifying the buffer pool to be attached to the port.

`port_tkn`

A token identifying the port that will use the buffer pool.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The Nucleus allocates buffers from this buffer pool for receive operations on associated ports. The application must return these buffers to the buffer pool when they are no longer needed using the **release_buffer** system call.

See also: Ports, buffer pools, *System Concepts*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The port and the buffer pool tokens refer to objects that are not in the same job.
E_EXIST	0006H	Either the port_tkn or the buffer_pool_tkn parameter does not refer to an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_STATE	0007H	The specified port already has a buffer pool attached.
E_TYPE	8002H	Either buffer_pool_tkn or the port_tkn parameter refers to an object that is not the correct type.
E_PROTOCOL	80E0H	The port specified in the port_tkn parameter was created as a signal type. It needs to be a data transport type.

attach_port

Enables an application to monitor several ports simultaneously. After attachment, any message sent to the port specified as the source port is automatically forwarded to the port specified as the sink port. Both sink and source ports must be of the same type.

Syntax, PL/M and C

```
CALL rq$attach$port (port_tkn, sink_port, except_ptr);
```

```
rq_attach_port (port_tkn, sink_port, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
port_tkn	SELECTOR	SELECTOR
sink_port	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

port_tkn

A token for the source port that forwards its messages.

sink_port

A token for the sink port that receives the forwarded messages.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Messages already queued at the source port are not forwarded, only messages that are received after **attach_port** is invoked.

Only one level of forwarding is supported. If a source port sends a request using the **send_rsvp** system call with the `flags` set to use the `receive_reply` option, the RSVP message is not forwarded to the sink port.

A port remains attached until either **detach_port** is invoked or the sink port is deleted.

See also: **create_port, send_rsvp, detach_port**,
Message forwarding, *System Concepts*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	Either the port_tkn parameter or the sink_port parameter refers to an object that is not a port.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PROTOCOL	80E0H	The port specified in the port_tkn or sink_port parameter must be of the data communication type, not the signal type.
E_STATE	0007H	The source port is already attached to a sink port.
E_TYPE	8002H	Either port_tkn or sink_port is not an existing object.

broadcast

Sends a control message to every message-passing host.

Syntax, PL/M and C

```
CALL rq$broadcast (port_tkn, socket, control_ptr, except_ptr);
```

```
rq_broadcast (port_tkn, socket, control_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
port_tkn	SELECTOR	SELECTOR
socket	WORD_32	UINT_32
control_ptr	POINTER	UINT_8 far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

port_tkn

A token for the sending port.

socket

Identifies the receiving port with a host_ID and port_ID pair. Since this is a broadcast, the host ID is ignored; you do not need to fill it in. If you are using iRMX for Windows in short-circuit mode (local message-passing only), specify 31 for the local host ID.

control_ptr

A pointer to a control message.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This call can broadcast a message to one port on each host in a system.

See also: Broadcasting a message, *System Concepts*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The port_tkn parameter does not refer to an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NUC_BAD_BUF	80E2H	One or more of these is true: • Control_ptr is not a valid pointer to a buffer. • The buffer pointed to by control_ptr is not large enough to hold the message.
E_PROTOCOL	80E0H	The specified destination port is a signal type port.
E_TRANSMISSION	000BH	A negative acknowledgment (NACK), timeout, bus or host error, or retry expiration occurred during the transmission of the message.
E_TYPE	8002H	The port_tkn parameter refers to an object that is not a port.

cancel

Performs synchronous cancellation of RSVP message transmission..

Syntax, PL/M and C

```
CALL rq$cancel (port_tkn, trans_id, except_ptr);
```

```
rq_cancel (port_tkn, trans_id, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
port_tkn	SELECTOR	SELECTOR
trans_id	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

port_tkn

A token for the port that was specified in a previous **send_rsvp** operation.

trans_id

The transaction ID of the message transmission to be canceled.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The RSVP is canceled whether or not the receiving task has called the **receive** or **receive_reply** system call. Canceling a **send_rsvp** disassociates the RSVP buffer, if any, from the port.

See also: **send_rsvp**,
Canceling an exchange, transaction ID, *System Concepts*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The port_tkn parameter does not refer to an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PROTOCOL	80E0H	The specified destination port was created as a signal type.
E_TRANS_ID	00E8H	Either the trans_id parameter is invalid, or the entire transaction is already complete. The transaction is complete if the Nucleus has received a response.
E_TYPE	8002H	The port_tkn parameter refers to an object that is not a port.

catalog_object

Places an entry for an object in the object directory of a specific job. The entry consists of both the object's name and token.

Syntax, PL/M and C

```
CALL rq$catalog$object (job, object, name, except_ptr);
```

```
rq_catalog_object (job, object, name, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
job	SELECTOR	SELECTOR
object	SELECTOR	SELECTOR
name	POINTER	void far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

job A token identifying the job in whose object directory the object is to be cataloged.

Selector Value	Meaning
Null	Specifies the job to which the calling task belongs.
Valid	Specifies the token for the job requested.

object

A token for the object to be cataloged. A null token is allowed.

name A pointer to a STRING containing the name under which the object is to be cataloged. The name must not be over 12 characters long. Each character can be a byte consisting of any value from 0 to 0FFH.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

There may be several entries for a single object in a directory, because the object may have several names. However, in a given object directory, only one object may be cataloged under a given name. If another task is waiting, using **lookup_object**, for the object to be cataloged, that task is awakened when the entry is cataloged.

See also: **create_task** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The pointer to the name is invalid. Either the selector does not refer to a valid segment, or the offset is outside the segment boundaries. This code is not returned by the DOS Real-time Extension (RTE).
E_CONTEXT	0005H	At least one of these is true: • The name being cataloged is already in the designated object directory. • The directory's maximum allowable size is 0
E_EXIST	0006H	Either the job parameter or the object parameter is not a token for an existing object.
E_LIMIT	0004H	The designated object directory is full.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The length of the STRING pointed to by the parameter is 0 or greater than 12.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_TYPE	8002H	The job parameter is a not a token for a job.

rqe_change_descriptor

Changes the base physical address and size of descriptors in the Global Descriptor Table (GDT).

⚠ **CAUTION**
This system call can change a descriptor's address to refer to any area of physical memory, even if other descriptors already refer to that memory. Although this may be useful for aliasing purposes, do not overlap memory accidentally.

Syntax, PL/M and C

```
CALL rqe$change$descriptor (descriptor, abs_addr, seg_size,
      except_ptr);
```

```
rqe_change_descriptor (descriptor, abs_addr, seg_size,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
descriptor	SELECTOR	SELECTOR
abs_addr	WORD_32	UINT_32
seg_size	NATIVE_WORD	NATIVE_WORD
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

`descriptor`

A token for the descriptor to be changed.

`abs_addr`

Specifies a full, 32-bit address. This is the address where you want the segment represented by this descriptor to start. If 0, the segment retains its current starting address.

`seg_size`

Specifies the size of the segment. If 0, the size is 64 Kbytes. If greater than 1 Mbyte, the size is rounded up to the nearest multiple of 4 Kbytes.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

You can only adjust those GDT entries that were created with **rqe_create_descriptor**. You cannot change descriptors that represent segments, tasks, mailboxes, call gates, or other iRMX objects.

See also: **rqe_create_descriptor** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The descriptor parameter is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_TYPE	8002H	The descriptor parameter is not a token for an iRMX descriptor.

rqe_change_object_access

Changes the access rights of iRMX segments.

Syntax, PL/M and C

```
CALL rqe$change$object$access (object, access, limit_mode,
      except_ptr);
```

```
rqe_change_object_access (object, access, limit_mode,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
object	SELECTOR	SELECTOR
access	BYTE	UINT_8
limit_mode	BYTE	UINT_8
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

object

A token for an object whose access rights are being changed. This token must represent a segment.

access

Specifies the new access rights for the object. These values are valid for iRMX objects:

Data Segments	Binary Value	Hex Value
Read-only	10010000B	90H
Read/write	10010010B	92H

Code Segments	Binary Value	Hex Value
Execute-only	10011000B	98H
Execute/read	10011010B	9AH
Execute-only (conforming)	10011100B	9CH
Execute/read (conforming)	10011110B	9EH

`limit_mode`

Specifies information on segment granularity and type for use by the processor in limit checking.

Binary	**Hexadecimal**	**Meaning**
00000000B	0H	1 byte granularity, 16-bit segment
01000000B	40H	1 byte granularity, 32-bit segment
10000000B	80H	4 K-byte granularity, 16-bit segment
11000000B	0C0H	4 K-byte granularity, 32-bit segment

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The `access` field specifies a variety of information about an object. Only the segment type and access rights can be modified with this call.

If you are changing only one field (either `access` or `limit_mode`), first call **rqe_get_object_access** to get `access` or `limit_mode`, change the field, then call **rq_change_object access** to specify the changed field.

See also: **rqe_get_object_access**,
Descriptors, composite objects, *System Concepts*,
create_segment example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The object whose access is to be changed does not exist or is not a valid iRMX object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	The object parameter refers to an object that is neither a segment nor a composite object.

connect

Creates a connection between a port owned by the calling task and a remote port.

Syntax, PL/M and C

```
CALL rq$connect (port_tkn, socket, except_ptr);
```

```
rq_connect (port_tkn, socket, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`port_tkn`	SELECTOR	SELECTOR
`socket`	WORD_32	UINT_32
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`port_tkn`

A token for a port object.

`socket`

Identifies the remote port with a `host_ID` and `port_ID` value.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

While connected, a port can only exchange messages with the port specified in `socket`.

Invoking **connect** with `socket` = 0 disconnects the calling task's port.

See also: Connecting a port, *System Concepts*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The port_tkn parameter does not refer to an existing object.
E_HOST_ID	00E2H	The host_id portion of the socket does not refer to a board that is currently in message space.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PROTOCOL	80E0H	The port specified in the port_tkn parameter was created as a signal type. It needs to be a data transport type.
E_STATE	0007H	The port specified in the port_tkn parameter is already the sink port of a forwarded port. Only one level of port forwarding is supported.
E_TYPE	8002H	The port_tkn parameter refers to an object that is not a port.

create_buffer_pool

Establishes and returns a token for a buffer pool.

Syntax, PL/M and C

```
buffer_pool = rq$create$buffer$pool (maximum_buffs, flags,
      except_ptr);
```

```
buffer_pool = rq_create_buffer_pool (maximum_buffs, flags,
      except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`buffer_pool`	SELECTOR	SELECTOR
`maximum_buffs`	WORD_16	UINT_16
`flags`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`buffer_pool`

The new buffer pool token.

Parameters

`maximum_buffs`

The maximum number of buffers that can exist in the buffer pool at one time. The maximum size of the buffer pool is controlled by this parameter.

`flags`

Defines the attributes of the buffer pool:

Bits	**Value**	**Meaning**
15-2	0	Reserved, set to 0.
1	0	Only contiguous buffers (segments) are used.
	1	Data chains are supported (iRMX III OS only).
0	0	Reserved, set to 0.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Once a buffer pool has been established, tasks can request segments of memory from the buffer pool using **request_buffer** instead of creating the segments directly using **create_segment** each time memory space is needed.

When a task finishes with a buffer, it can release the buffer back to the buffer pool using **release_buffer** for later use by other tasks.

Each buffer pool can manage as many as 8192 segments that can be of 8 different sizes.

When creating data chains, the largest available buffer will be used for the first portion of the data chain, then the next buffer and so on. These available buffers may be larger than the data actually stored in them. Therefore, a data chain may use more physical space than the data would actually require.

⟹ **Note**
You can use data chains only in the iRMX III OS. The configuration of iRMX for Windows and iRMX for PCs does not allow the use of data chains.

See also: **request_buffer, release_buffer**,
Using buffer pools, memory allocation, buffer pools, data chaining, *System Concepts*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_MEM	0002H	There isn't enough memory to create the requested buffer pool.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The maximum_buffs parameter has a value greater than 8192.
E_SLOT	000CH	There is no room in the GDT for the buffer pool's descriptor.

create_composite

Creates a composite object of the specified extension type. The call accepts a list of tokens that specify the component objects and returns a token for the new composite object. Composite objects require the creation of extension objects. Jobs that create extension objects cannot be deleted until all the extension objects are deleted.

⚠ **CAUTION**
Avoid creating composite objects in HI applications. If an HI application creates extension objects, the application cannot be deleted asynchronously with <Ctrl-C> entered at the keyboard. The system must be rebooted to recover.

Syntax, PL/M and C

```
composite = rq$create$composite (extension, token_list,
      except_ptr);
```

```
composite = rq_create_composite (extension, token_list,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
composite	SELECTOR	SELECTOR
extension	SELECTOR	SELECTOR
token_list	POINTER	TOKEN_LIST_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

composite

The new composite token.

Parameters

`extension`

A token for an extension type representing a license to create a composite object.

`token_list`

A pointer to this structure:

```
DECLARE token_list STRUCTURE (
   num_slots              WORD_16,
   num_used               WORD_16,
   tokens(*)              SELECTOR);
```

or

```
typedef struct {
   UINT_16                num_slots;
   UINT_16                num_used;
   SELECTOR               tokens[_NUM_TOKENS];
                          /* adjust to fit
                          num_used */
} TOKEN_LIST_STRUCT;
```

Where:

`num_slots`

Maximum number of slots for component objects that the composite object can contain.

`num_used` Number of token elements to include in the composite.

`tokens` An array of tokens that will actually constitute the composite object.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Create_composite selects included tokens beginning with the first token in the token list. If the number of token elements (`num_used`) is less than the number of component slots (`num_slots`), **create_composite** fills the remaining slots with the a null selector value. If `num_slots` is less than `num_used`, **create_composite** ignores the remaining tokens in the token list.

See also: Component objects, composite objects, extension objects, type manager, *System Concepts*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The token_list pointer is invalid. Either the selector does not refer to a valid segment, or the offset is outside the segment boundaries.
E_EXIST	0006H	The extension parameter or one or more of the not 0 token_list parameters is not a token for an existing object.
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 insufficient to create a composite.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The specified number of components is 0.
E_SLOT	000CH	There is no room in the GDT for the composite object's descriptor.
E_TYPE	8002H	The extension parameter is not a token for an extension object.

rqe_create_descriptor

Builds a descriptor for an Intel386, Intel486 or Pentium memory segment, places the descriptor in the GDT, and returns a token for that descriptor.

⚠ **CAUTION**
This system call can set up a segment descriptor to refer to any area of physical memory, even if other descriptors already refer to that memory. Although this may be useful for aliasing purposes, do not overlap memory accidentally.

Syntax, PL/M and C

```
descriptor = rqe$create$descriptor (abs_addr, seg_size,
      except_ptr);
```

```
descriptor = rqe_create_descriptor (abs_addr, seg_size,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
descriptor	SELECTOR	SELECTOR
abs_addr	WORD_32	UINT_32
seg_size	NATIVE_WORD	NATIVE_WORD
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

`descriptor`
The new descriptor token.

Parameters

`abs_addr`
Specifies a full, 32-bit physical address. This is the address where you want the segment represented by this descriptor to start.

`seg_size`
Specifies the size of the segment. If 0, the size is 64 Kbytes. If greater than 1 Mbyte, the size is rounded up to the nearest multiple of 4 Kbytes.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Before the microprocessor can access an area of memory in Protected Mode, a descriptor for the memory segment must exist in one of the descriptor tables: the GDT or LDT. For iRMX objects such as jobs, tasks, segments, and mailboxes, the OS automatically creates descriptors as necessary. **Rqe_create_descriptor** lets you add your own descriptors to the GDT.

A segment created with this system call can be deleted by calling either **rqe_delete_descriptor** or **delete_segment**. However, segments created with **rqe_create_descriptor** are marked as descriptors, not iRMX segments. Unlike ordinary iRMX segments set up with **create_segment**, the memory associated with these segments does not return to the iRMX memory pool for reallocation when the segments are deleted. Rather, the GDT slot is returned to the memory manager for reassignment.

See also: Descriptors, *System Concepts,*
rqe_create_descriptor example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_LIMIT	0004H	Creating the requested descriptor would exceed the job's object limit.
E_MEM	0002H	The memory available to the calling task's job is insufficient to create the descriptor.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_RESOURCE_LIMIT	00E6H	An internal table limit has been reached. This table keeps track of the number of objects created by each DOS process and is a configured value. This is a DOS RTE error only.
E_SLOT	000CH	There is no room in the GDT for the new descriptor.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.

create_extension

Creates a new object type and returns a token for it.

CAUTION
Avoid creating extension objects in HI applications. If an HI application creates extension objects, the application cannot be deleted asynchronously with a <Ctrl-C> entered at the keyboard. The system must be rebooted to recover.

Syntax, PL/M and C

```
extension = rq$create$extension (type_code, deletion_mailbox,
      except_ptr);
```

```
extension = rq_create_extension (type_code, deletion_mailbox,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
extension	SELECTOR	SELECTOR
type_code	WORD_16	UINT_16
deletion_mailbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

`extension`
The new type token.

Parameters

`type_code`
Specifies a type code for the new object.

Value	Meaning
0-7FFFH	Reserved
8000-0FFFFH	Valid type codes for user-created composites.

See also: **get_type**

`deletion_mailbox`
A token for the mailbox where objects of the new type are sent whenever the extension type or their containing job is deleted. A null selector value indicates no deletion mailbox is desired.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If you specify a deletion mailbox, a task in your type manager must wait there for the tokens of objects that are to be deleted. Tokens are sent to the deletion mailbox either when their extension type or containing job is deleted; they are not sent there when being deleted by **delete_composite**. The task servicing the deletion mailbox may do anything with the composite objects sent to it, but it must delete them. If you do not specify a deletion mailbox, composite objects of that type are deleted automatically and the type manager is not informed.

A job containing a task that creates an extension object cannot be deleted until the extension object is deleted.

See also: **create_extension** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The calling task's job is being deleted.
E_EXIST	0006H	The deletion_mailbox parameter is not a token for an existing object.
E_LIMIT	0004H	The calling task's job has reached its object limit.
E_MEM	0002H	The memory available to the calling task's job is not sufficient to create an extension.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SLOT	000CH	There is no room in the GDT for the extension's descriptor.
E_PARAM	8004H	The type_code parameter is invalid.
E_TYPE	8002H	The deletion_mailbox parameter is not a token for a mailbox.

create_job

Obsolete. Creates a job with a single task and returns a token for the job. This call is identical to **rqe_create_job**. It is only used in applications written for earlier versions of the iRMX OS. This system call is not supported for flat model applications.

Syntax, PL/M and C

```
job = rq$create$job (directory_size, param_obj, pool_min,
      pool_max, max_objects, max_tasks, max_priority,
      except_handler, job_flags, task_priority, start_address,
      data_seg, stack_ptr, stack_size, task_flags, except_ptr);
```

```
job = rq_create_job (directory_size, param_obj, pool_min,
      pool_max, max_objects, max_tasks, max_priority,
      except_handler, job_flags, task_priority, start_address,
      data_seg, stack_ptr, stack_size, task_flags, except_ptr);
```

See also: **rqe_create_job**

rqe_create_job

Creates a job with an initial task and returns a token for the job. This system call is not supported for flat model applications.

Syntax, PL/M and C

```
job = rqe$create$job (directory_size, param_obj, pool_min,
      pool_max, max_objects, max_tasks, max_priority,
      except_handler, job_flags, task_priority, start_address,
      data_seg, stack_ptr, stack_size, task_flags, except_ptr);
```

```
job = rqe_create_job (directory_size, param_obj, pool_min,
      pool_max, max_objects, max_tasks, max_priority,
      except_handler, job_flags, task_priority, start_address,
      data_seg, stack_ptr, stack_size, task_flags, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
job	SELECTOR	SELECTOR
directory_size	WORD_16	UINT_16
param_obj	SELECTOR	SELECTOR
pool_min	WORD_32	UINT_32
pool_max	WORD_32	UINT_32
max_objects	WORD_16	UINT_16
max_tasks	WORD_16	UINT_16
max_priority	BYTE	UINT_8
except_handler	POINTER	EXCEPTION_STRUCT far *
job_flags	WORD_16	UINT_16
task_priority	BYTE	UINT_8
start_address	POINTER	void (far *)(void)
data_seg	SELECTOR	SELECTOR
stack_ptr	POINTER	void far *
stack_size	WORD_32	NATIVE_WORD
task_flags	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

job The new job token.

Parameters

`directory_size`

Specifies the maximum allowable number of entries a job can have in its object directory. The value 0 indicates that no object directory is desired. The maximum value is 0F00H.

`param_obj`

A token for one of these:

Value	**Meaning**
null selector	Indicates that the new job has no parameter object.
valid selector	The token for the new job's parameter object.

See also: Parameter objects, *System Concepts*

`pool_min`

Specifies the minimum size of the new job's memory pool in 16-byte paragraphs. The upper limit is 4 Gbytes. The lower limit is 32 + x +y, where x and y are calculated as follows:

```
IF task_flags indicates an initial task that uses floating point
instructions
    x = 6
ELSE
    x = 0
IF stack_ptr = NIL
    y = stack_size / 16
ELSE
    y = 0
```

`pool_max`

Specifies the maximum allowable size of the new job's memory pool in 16-byte paragraphs. The upper limit is 4 Gbytes.

`max_objects`

Specifies one of these:

Value	**Meaning**
0-0FFFEH	The maximum number of objects, created by tasks in the new job, that can exist at one time.
0FFFFH	Unlimited number of objects

`max_tasks`

Specifies these:

Value	Meaning
0	Produces the E_LIMIT exception.
1-0FFFEH	The maximum number of tasks that can exist simultaneously in the new job.
0FFFFH	Unlimited number of tasks

`max_priority`

Specifies one of these:

Value	Meaning
0	Tasks in the new job have the maximum priority of the parent job.
1-255	The maximum allowable priority of tasks in the new job; if max_priority exceeds the maximum priority of the parent job, an E_LIMIT error is returned.

`except_handler`

A pointer to this structure:

```
DECLARE exception STRUCTURE (
   exception_handler_ptr  POINTER,
   exception_mode         BYTE);
```

or

```
typedef struct {
   void far *             exception_handler_ptr;
   UINT_8                 exception_mode;
} EXCEPTION_STRUCT;
```

Where:

`exception_handler_ptr`

If not null, references the first instruction of the new job's own exception handler. If null, the new job's exception handler is the system default exception handler. The exception handler for the new task becomes the default exception handler for the job.

`exception_mode`

Indicates when control is to be passed to the exception handler. It is encoded:

Value	When Control Passes To Exception Handler
0	Never
1	On programmer errors only
2	On environmental conditions only
3	On all exceptional conditions

`job_flags`

Specifies information that the Nucleus needs to create and maintain the job.

Bits	Value	Meaning
15-2	0	Reserved, set to 0.
1	0	Nucleus checks call parameters for validity whenever a task in the new job or any of its offspring makes a system call.
	1	Nucleus will not check parameters unless an ancestor of the new job has been created with this bit set to 0.
0	0	Reserved, set to 0.

`task_priority`

Controls task priority:

Value	Meaning
0	The new job's initial task priority is equal to the new job's maximum priority.
1-255	The priority of the new job's initial task; if the task_priority parameter is greater (numerically smaller) than the new job's maximum priority, an E_PARAM error is returned.

`start_address`

A pointer to the first instruction of the new job's initial task, which is the task created with the job.

`data_seg`

A token for the data segment the new job's initial task is to use.

Value	Meaning
Valid selector	The base selector of the data segment of the new job's initial task.
Null selector	The new job's initial task assigns its own data segment.

`stack_ptr`

A pointer that specifies the location of the stack for the new job's initial task.

Value	Meaning
Valid pointer	References the base of the user-provided stack.
Null pointer	Nucleus allocates a stack for the new job's initial task; the length of the allocated segment is equal to the value of the stack_size parameter.

`stack_size`

Specifies the size of the stack for the created job. `Stack_size` must be at least 16 bytes but should be at least 1024 bytes if the new task is going to make Nucleus system calls.

See also: Stack, *Programming Techniques* `task_flags`

Specifies whether the initial task contains floating-point instructions.

Bits	Value	Meaning
15-1	0	Reserved, set to 0.
0	0	The initial task does not contain floating-point instructions.
	1	The initial task contains floating-point instructions; this requires a math coprocessor or FPU.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The new job's parent is the calling task's job. The new job and initial task each deduct an object from the parent job's object limit.

See also: Maximum tasks, maximum objects, *System Concepts*,
rqe_create_job example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	At least one of the except_handler, data_seg, or stack_ptr parameters is invalid. Either a selector does not refer to a valid segment, or an offset is outside the segment boundaries.
E_CONTEXT	0005H	The job containing the calling task is being deleted.
E_EXIST	0006H	The param_obj parameter is not a null selector and is not a token for an existing object.

E_LIMIT	0004H	At least one of these is true: • `max_objects` is larger than the unused portion of the object allotment in the calling task's job. • `max_tasks` is larger than the unused portion of the task allotment in the calling task's job. • `max_priority` is greater (numerically smaller) than the maximum allowable task priority in the calling task's job. • `directory_size` is larger than 0FF0H. • The initial task would exceed the object limit in the new job because the `max_objects` parameter is set to 0. • The initial task would exceed the task limit in the new job. The max_tasks parameter is set to 0.
E_MEM	0002H	At least one of these is true: • The memory available to the new job is not sufficient to create a job descriptor and the object directory. • The memory available to the new job is not sufficient to satisfy the `pool_min` parameter. • The memory available to the new job is not sufficient to create the task as specified.
E_PARAM	8004H	At least one of these is true: • `pool_min` is less than 16 + (number of paragraphs needed for the initial task and a system-allocated stack) + 5 if the task uses the math coprocessor. • `pool_min` is greater than `pool_max`. • `task_priority` is unequal to 0 and greater (numerically smaller) than `max_priority`. • `stack_size` is less than 16 (applies to 32-bit applications only; the OS automatically adds enough to the stack for 16-bit applications that this error cannot occur). • The exception handler mode is not valid.
E_SLOT	000CH	There isn't enough room in the GDT for the new job and task descriptors.

create_mailbox

Creates a mailbox and returns a token for the object.

See also: Mailboxes, *System Concepts*,
create_mailbox example, Nucleus examples

Syntax, PL/M and C

```
mailbox = rq$create$mailbox (mailbox_flags, except_ptr);

mailbox = rq_create_mailbox (mailbox_flags, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`mailbox`	SELECTOR	SELECTOR
`mailbox_flags`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`mailbox`

The new mailbox token.

Parameters

`mailbox_flags`

Indicates:

Bits	**Value**	**Meaning**
15-6	0	Reserved, set to 0.
5	0	This mailbox passes iRMX objects (signal type messages); use the **send_message** and **receive_message** calls.
	1	This mailbox passes up to 128 bytes of data per message (data type messages); use the **send_data** and **receive_data** calls.
4-1		If bit 5 is 0, the value placed here multiplied by 4 sets the number of message objects that can be queued on the high performance object queue (minimum size of 8 objects). Otherwise, the OS creates a data queue for three 128-byte messages, ignoring these bits.
0	0	FIFO task queuing
	1	Priority-based task queue

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

When you set up a mailbox to pass objects (not data) you can also set up a high-performance queue. This queue is a block of memory that stores objects waiting to be sent or received. It is permanently assigned to the mailbox, and the unused portion of the queue is unavailable for other uses. If the queue overflows, the Nucleus temporarily allocates another 200-object queue.

To get the best tradeoff between memory and performance, choose a size for your high-performance queue that is large enough for normal operations, and let the overflow queue handle unusual circumstances.

When you create a mailbox to pass data, you do not specify the size of the message queue. The OS automatically sets up a queue of 400 bytes.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
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 create a mailbox.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_RESOURCE_LIMIT	00E6H	An internal table limit has been reached. This table keeps track of the number of objects created by each DOS process and is a configured value. This is a DOS RTE error only.
E_SLOT	000CH	There isn't enough room in the GDT for the new job and task descriptors
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.

create_port

Creates a port object that can be used in message passing.

Syntax, PL/M and C

```
port_tk = rq$create$port (num_trans, info_ptr, except_ptr);
```

```
port_tkn = rq_create_port (num_trans, info_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
port_tkn	SELECTOR	SELECTOR
num_trans	WORD_16	UINT_16
info_ptr	POINTER	void far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

port_tkn

A token for the new port.

Parameters

num_trans

Specifies the number of simultaneous transactions allowed at this port.

info_ptr

A pointer to a structure of a form that is protocol-dependent.

For signal protocol, the structure has this form:

```
DECLARE signal_port_creation_info STRUCTURE (
   message_id              BYTE,
   reserved_a              BYTE,
   type                    BYTE,
   reserved_b              BYTE
   flags                   WORD_16);
```

or

```
typedef strut {
   UINT_8                  message_id;
   UINT_8                  reserved_a;
   UINT_8                  type;
   UINT_8                  reserved_b;
   UINT_16                 flags;
} SIGNAL_PORT_CREATION_INFO_STRUCT;
```

Where:

`message_id`
The slot ID of the remote host (equivalent to the `host_id` portion of the socket). This must be in the range of 0 to 19.

`reserved` Reserved, set to 0.

`type` The message protocol of the port as specified by:

Value	**Meaning**
0-1	Reserved for the Nucleus
2	Data Transport protocol
3	Signal protocol (specify here)
4-0FFH	Reserved, set to 0.

`flags` Defines the port's task queuing scheme.

Bits	**Meaning**
15-2	Reserved, set to 0.
1	Task queueing scheme: 0 = FIFO 1 = priority
0	Reserved, set to 0.

For transport protocol, the structure takes this form. A null pointer to this structure selects default values.

```
DECLARE data_port_creation_info STRUCTURE (
   port_id                 WORD_16,
   type                    BYTE,
   reserved                BYTE,
   flags                   WORD_16);
```

or

```
typedef struct {
   UINT_16                 port_id;
   UINT_8                  type;
   UINT_8                  reserved;
   UINT_16                 flags;
} DATA_PORT_CREATION_INFO_STRUCT;
```

Where:

port_id Identifies the port. Port ID values are:

ID Range	Explanation
0	The Nucleus assigns the port ID (default)
1-7FFH	Reserved
800H-0FFFH	Available to users
1000H-0FFFFH	Reserved

type The message protocol of the port.

Value	Meaning
0-1	Reserved for the Nucleus
2	Data transport protocol (default)
3	Signal protocol
4-0FFH	Reserved

flags Defines fragmentation control and task queuing scheme.

Bits	Meaning
15-3	Reserved, set to 0.
2	Message fragmentation: 0 = enabled (default) 1 = disabled
1	Task queueing scheme: 0 = FIFO (default) 1 = priority
0	Reserved, set to 0.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The new port counts against the object limit for the calling task's job. Tasks created within the same job can send and receive messages or signals through the same port, depending on the application.

For ports using signal protocol, only one connection can be established between any two hosts. Attempting to connect more than one port to the same host results in an E_CONTEXT condition code.

➠ **Note**
Ports using signal protocol receive messages before ports using data transport protocol. Therefore, if you create both types of ports on one host, the ports using data transport protocol will not receive messages.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	Signal protocol was specified with a message_id already associated with a port.
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 create a port.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NUC_BAD_BUF	80E2H	The info_ptr is invalid or points to a buffer that is not large enough.
E_PARAM	8004H	The message_id was greater than 19.
E_PORT_ID_USED	80E1H	The port_id specified is in use.
E_SLOT	000CH	There isn't enough room in the GDT for another descriptor.

create_region

Creates a region and returns a token for it.

⚠ **CAUTION**
Avoid using regions in HI applications. You cannot stop the application asynchronously with <Ctrl-C> entered at the keyboard while a task is in the region. To do so will require rebooting.

Syntax, PL/M and C

```
region = rq$create$region (region_flags, except_ptr);
```

```
region = rq_create_region (region_flags, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`region`	SELECTOR	SELECTOR
`region_flags`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`region`

A token for the new region.

Parameters

`region_flags`

The value of bit 0 specifies the queuing scheme of the new region:

Value	Protocol
0	FIFO
1	Priority based

The other bits are reserved; set to 0

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Tasks that use regions cannot be deleted while they are in control of the region.

See also: **accept_control**, **receive_control**,
Regions, *System Concepts*,
create_region example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_LIMIT	0004H	The calling task's job has reached its object limit.
E_MEM	0002H	The memory pool of the calling task's job is too small to satisfy the request.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration. This code is returned if you make this call as an RTE call from Windows instead of from DOS.
E_RESOURCE_LIMIT	00E6H	An internal table limit has been reached. This table keeps track of the number of objects created by each DOS process and is a configured value. This is a DOS RTE error only.
E_SLOT	000CH	There is not enough room in the GDT for another descriptor.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.

create_segment

Creates a segment and returns the token for it.

Syntax, PL/M and C

```
segment = rq$create$segment (seg_size, except_ptr);
```

```
segment = rq_create_segment (seg_size, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`segment`	SELECTOR	SELECTOR
`seg_size`	NATIVE_WORD	NATIVE_WORD
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`segment`

A token for the new segment.

Parameters

`seg_size`

Specifies the size of the segment. If 0, the size is 64 Kbytes. If greater than 1 Mbyte, the created segment is rounded up to the nearest multiple of 4 Kbytes.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The memory for the segment is taken from the portion of the free space memory pool belonging to the calling task's job, unless borrowing from the parent job is both necessary and possible. The new segment counts against the object limit for the calling task's job.

When setting up the descriptor for the new segment, the Nucleus assigns the segment as a data segment, with read/write access, at privilege level 0.

For iRMX for Windows, if you create a segment from DOS RTE, you must also delete the segment from DOS RTE. Otherwise, you will eventually receive an E_RESOURCE_LIMIT condition code.

See also: **create_segment** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
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 too small to create a segment of the specified size.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_RESOURCE_LIMIT	00E6H	An internal table limit has been reached. This table keeps track of the number of objects created by each DOS process and is a configured value. This is a DOS RTE error only.
E_SLOT	000CH	There isn't enough room in the GDT for another descriptor.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.

create_semaphore

Creates a semaphore and returns a token for it.

Syntax, PL/M and C

```
semaphore = rq$create$semaphore (initial_value, max_value,
      semaphore_flags, except_ptr);
```

```
semaphore = rq_create_semaphore (initial_value, max_value,
      semaphore_flags, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
semaphore	SELECTOR	SELECTOR
initial_value	WORD_16	UINT_16
max_value	WORD_16	UINT_16
semaphore_flags	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

semaphore

A token for the new semaphore.

Parameters

initial_value

The initial number of units to be in the custody of the new semaphore.

max_value

The maximum number of units over which the new semaphore is to have custody at any given time. If max_value is 0, an E_PARAM error is returned.

semaphore_flags

Bit 0 specifies the queuing scheme for the new semaphore's task queue; the remaining bits are reserved and should be set to 0.

Value	Meaning
0	FIFO task queue
1	Priority-based queue

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The created semaphore counts against the object limit for the calling task's job.

See also: **send_units**,
Semaphores, *System Concepts*,
create_semaphore example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
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 create a semaphore.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	At least one of these is true: • The `initial_value` parameter is larger than the `max_value` parameter. • The max_value parameter is 0.
E_RESOURCE_LIMIT	00E6H	An internal table limit has been reached. This table keeps track of the number of objects created by each DOS process and is a configured value. This is a DOS RTE error only.
E_SLOT	000CH	There is not enough room in the GDT for another descriptor.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.

create_task

Creates a task and returns a token for it.

Syntax, PL/M and C

```
task = rq$create$task (priority, start_address, data_seg,
      stack_ptr, stack_size, task_flags, except_ptr);
```

```
task = rq_create_task (priority, start_address, data_seg,
      stack_ptr, stack_size, task_flags, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
task	SELECTOR	SELECTOR
priority	BYTE	UINT_8
start_address	POINTER	void (far *)(void)
data_seg	SELECTOR	SELECTOR
stack_ptr	POINTER	void far *
stack_size	NATIVE_WORD	NATIVE_WORD
task_flags	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

task A token for the new task.

Parameters

priority

Specifies the priority of the new task.

Value	Meaning
0	Priority is the maximum allowable priority of the calling task's job.
1-255	The priority of the new task; this must not exceed the maximum allowable priority of the calling task's job.

start_address

A pointer to the first instruction of the new task.

data_seg

A token that specifies the new task's data segment.

Value	Meaning
Null selector	New task assigns its own data segment. When you create a flat model task, this parameter must be non-zero.
Valid selector	Token for the base address of the data segment.

`stack_ptr`

A pointer that specifies the location of the stack for the new task.

Value	Meaning
Null pointer	Nucleus allocates a stack to the new task; the length of the stack is equal to the value of the stack_size parameter.
Valid selector	Nucleus places the sum of the offset portion and the stack_size parameter in SP (stack pointer) register.

`stack_size`

Specifies the size of the stack area for the created task. `Stack_size` must be at least 16 bytes but at least 1024 bytes if the new task is going to make Nucleus system calls. The maximum stack size can be 4 Gbytes.

See also: Stack, *Programming Techniques*

`task_flags`

Indicates:

Bits	Value	Meaning
15-1	0	Reserved, set to 0.
0	0	The task does not contain floating-point instructions.
	1	The task contains floating-point instructions; this requires a math coprocessor or FPU.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The new task counts against the object limit for the calling task's job.

Attributes of the new task are initialized upon creation:

Attribute	Initial Value
Priority	As specified in the call
Execution state	Ready
Suspension depth	0
Containing job	Calling task's job
Exception handler	Containing job's exception handler
Exception mode	The exception mode of the containing job

See also: **create_task** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	Either the data_seg selector does not refer to a valid segment, or the offset of the stack_ptr parameter is outside the segment boundaries.
E_LIMIT	0004H	At least one of these is true: • The calling task's job has already reached its object limit or task limit. • The priority parameter is not 0 and greater (numerically smaller) than the maximum allowable priority for tasks in the calling task's job.
E_MEM	0002H	The memory available to the calling task's job is not sufficient to create a task as specified.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The stack_size parameter is less than 16.
E_SLOT	000CH	There isn't enough room in the GDT for another descriptor.

delete_buffer_pool

Deletes a buffer pool and any segments that it may contain; data in those segments will be lost. A buffer pool cannot be deleted while it is attached to a port.

Syntax, PL/M and C

```
CALL rq$delete$buffer$pool (buffer_pool, except_ptr);
```

```
rq_delete_buffer_pool (buffer_pool, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`buffer_pool`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`buffer_pool`

A token for the buffer pool to be deleted. This buffer pool must have been created with **create_buffer_pool**.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BUSY	0003H	The buffer pool is attached to a port.
E_EXIST	0006H	The buffer_pool parameter is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	The buffer_pool parameter is the token for an object that is not a buffer pool.

delete_composite

Deletes the specified composite object, but not its component objects.

See also: **create_composite**

Syntax, PL/M and C

```
CALL rq$delete$composite (extension, composite, except_ptr);
```

```
rq_delete_composite (extension, composite, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`extension`	SELECTOR	SELECTOR
`composite`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`extension`
A token for the extension type of the composite object to be deleted.

`composite`
A token for the composite object to be deleted.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The extension type does not match the composite parameter.
E_EXIST	0006H	One or both of the extension or composite parameters is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	One or both of the extension or composite parameters is not a token for an object of the correct type.

rqe_delete_descriptor

Removes an entry defined with **rqe_create_descriptor** from the GDT.

Syntax, PL/M and C

```
CALL rqe$delete$descriptor (descriptor, except_ptr);
```

```
rqe_delete_descriptor (descriptor, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`descriptor`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`descriptor`

A token for the descriptor to be deleted.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Memory that was addressed by the descriptor is not returned to the memory pool. The GDT slot is returned to the memory manager for reassignment.

See also: **rqe_create_descriptor** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	Either the descriptor parameter is not a token for an existing object, or it represents a descriptor for a job being deleted.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_TYPE	8002H	The descriptor parameter is not a token for a descriptor.

delete_extension

Deletes the specified extension object and all composites of that type, making the corresponding type code available for reuse.

Syntax, PL/M and C

```
CALL rq$delete$extension (extension, except_ptr);

rq_delete_extension (extension, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
extension	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

extension

A token for the extension object to be deleted.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Delete_extension is not completed until all of the composite objects have been deleted. If an extension has no deletion mailbox, composite objects created by **create_extension** are deleted without informing the type manager. The job containing the task that created the extension object cannot be deleted until the extension object is deleted.

See also: **create_extension**,
Type manager, *System Concepts*,
create_extension example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The extension parameter is not a token for an existing object.
E_MEM	0002H	The memory available to the calling task's job is too small to complete this operation.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	The extension parameter is not a token for an extension object.

delete_job

Deletes the specified job, all of the job's tasks, and all objects created by the tasks.

Syntax, PL/M and C

```
CALL rq$delete$job (job, except_ptr);

rq_delete_job (job, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`job`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`job` A token for the job to be deleted. A null selector specifies the calling task's job.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

During the deletion of any interrupt tasks owned by the job, the interrupt levels associated with those tasks are reset. The levels that do not have interrupt tasks associated with them will not be reset during a **delete_job** call.

During deletion, all resources that the target job had borrowed from its parent are returned. Deleting a job counts toward the object limit for the parent job.

Jobs that have created extension objects cannot be deleted until all the extension objects are deleted.

See also: **offspring**, **delete_composite**, **delete_extension**,
Job deletion, extension objects, *System Concepts,*
delete_job example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	At least one of these is true: • There are undeleted jobs or extension objects which have been created by tasks in the target job. • The deleting task has access to data guarded by a region contained in the job to be deleted.
E_EXIST	0006H	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.
E_TYPE	8002H	The job parameter is not a token for a job.

delete_mailbox

Deletes the specified mailbox.

Syntax, PL/M and C

```
CALL rq$delete$mailbox (mailbox, except_ptr);
```

```
rq_delete_mailbox (mailbox, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
mailbox	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

mailbox

A token for the mailbox to be deleted.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If any tasks are queued at the mailbox at deletion time, they are awakened with an E_EXIST exceptional condition. If there is a queue of object tokens or data messages, the queue is discarded. Deleting the mailbox counts toward the object limit for the containing job.

See also: **create_mailbox** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	Either the mailbox parameter is not a token for an existing object or it represents a mailbox for a job being deleted.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	The mailbox parameter is not a token for a mailbox.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.

delete_port

Deletes the specified port.

Syntax, PL/M and C

```
CALL rq$delete$port (port_tkn, except_ptr);
```

```
rq_delete_port (port_tkn, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
port_tkn	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

port_tkn

A token for the port to be deleted.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If any tasks are in the port's receive task queue at deletion time, they are awakened with an E_EXIST exceptional condition. Deleting the port counts toward the object limit for the containing job. Any messages queued at the port are discarded and, if the port is forwarded, forwarding is severed.

Deleting a sink port automatically detaches it from its source port.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	Either the port_tkn parameter is not a token for an existing object or it represents a port for a job being deleted.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	The port_tkn parameter is not a token for a port.

delete_region

Deletes the specified region.

Syntax, PL/M and C

```
CALL rq$delete$region (region, except_ptr);
```

```
rq_delete_region (region, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`region`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`region`

A token for the region to be deleted.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If a task that has access to the region requests that the region be deleted, the task receives an E_CONTEXT condition code. If a task requests deletion while another task has access, deletion is delayed until access is surrendered.

A deadlock can occur when two or more tasks request deletion of a region that another task has access to, or when a task attempts to delete another task that is trying to delete an occupied region. When the region is deleted, any waiting tasks awaken with an E_EXIST exceptional condition.

See also: **create_region**, **accept_control**,
Regions, mutual exclusion, deadlock, *System Concepts*,
create_region example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The deletion is being requested by a task that currently holds access to data protected by the region.
E_EXIST	0006H	The region parameter is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration. This code is returned if you make this call as an RTE call from Windows instead of from DOS.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_TYPE	8002H	The region parameter is not a token for a region.

delete_segment

Deletes segments created with **create_segment**, **rqe_create_descriptor**, and **rqv_create_segment**.

Syntax, PL/M and C

```
CALL rq$delete$segment (segment, except_ptr);
```

```
rq_delete_segment (segment, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
segment	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

segment

A token for the segment or descriptor to be deleted.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This system call returns the deleted segment to the memory pool from which it was allocated. The deleted segment counts toward the object limit for the containing job.

When deleting a descriptor, this system call does not return any memory to the memory pool. It clears the descriptor slot in the GDT and returns that slot to the memory manager for reassignment.

When deleting a virtual segment created with **rqv_create_segment**, both the virtual address space and all physical pages within the virtual segment are deallocated.

See also: **create_segment** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	At least one of these is true: • The `segment` parameter is not a token for an existing object. • The segment parameter represents a `segment` or descriptor for a job being deleted.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_TYPE	8002H	The `segment` parameter is not a token for a segment or a descriptor.

delete_semaphore

Deletes the specified semaphore.

Syntax, PL/M and C

```
CALL rq$delete$semaphore (semaphore, except_ptr);
```

```
rq_delete_semaphore (semaphore, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
semaphore	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

semaphore

A token for the semaphore to be deleted.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If there are tasks in the semaphore's queue at deletion time, they are awakened with an E_EXIST exceptional condition. The deleted semaphore counts toward the object limit for the containing job.

See also: **create_semaphore** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	One of these is true: • The `semaphore` parameter is not a token for an existing object • The semaphore parameter represents a semaphore for a job being deleted.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_TYPE	8002H	The semaphore parameter is not a token for a semaphore.

delete_task

Deletes the specified task from the system and from any queues in which the task was waiting.

Syntax, PL/M and C

```
CALL rq$delete$task (task, except_ptr);

rq_delete_task (task, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`task`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`task` A token identifying the task to be deleted.

Value	Meaning
Null selector	Delete the calling task
Valid selector	Token for the task to be deleted

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Delete_task enables any task currently within a region, or waiting in a queue for a region, to exit the region before being deleted. Deleting the task counts toward both the object and task limits for the containing job.

Any attempt to delete an interrupt task results in an E_CONTEXT condition code. Use the **reset_interrupt** system call instead.

See also: **create_task** example, Nucleus examples

➠ **Note**

Deleting a task does not delete the C resources allocated to it. Applications that delete C tasks should call **suspend_task**, then **_cstop**, before deleting the task in order to delete these resources.

See also: **_cstop** function, *C Library Reference*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The task parameter is a token for an interrupt task.
E_EXIST	0006H	One of these conditions has occurred: • The `task` parameter is not a token for an existing object. • The `task` parameter represents a task for a job that is being deleted. • More than one task is trying to delete a task which is in a region.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	The task parameter is a token for an object which is not a task.

detach_buffer_pool

Ends the association between a buffer pool and a port.

Syntax, PL/M and C

```
buffer_pool_tkn = rq$detach$buffer$pool (port_tkn, except_ptr);
```

```
buffer_pool_tkn = rq_detach_buffer_pool (port_tkn, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
buffer_pool_tkn	SELECTOR	SELECTOR
port_tkn	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

buffer_pool_tkn

A token for the buffer pool that was detached.

Parameters

port_tkn

A token identifying the port to be detached from the buffer pool.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This call does not delete the buffer pool. The token received as a result of this call can be used to attach the buffer pool to a different port, or to reattach it to the same port.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The port_tkn parameter is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PROTOCOL	80E0H	The port specified in the port_tkn parameter was created as a signal type. It needs to be a data transport type.
E_STATE	0007H	No port is associated with the specified port.
E_TYPE	8002H	The port_tkn parameter refers to an object that is not a port.

detach_port

Ends message forwarding from the specified port.

Syntax, PL/M and C

```
CALL rq$detach$port (port_tkn, except_ptr);

rq_detach_port (port_tkn, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`port_tkn`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`port_tkn`

A token for the source port to detach.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If **detach_port** is invoked with messages queued at the sink port, they remain at the sink port until removed with a receive operation.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The port_tkn parameter does not refer to an existing port.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PROTOCOL	80E0H	The port specified in the port_tkn parameter was created as a signal type. It needs to be a data transport type.
E_STATE	0007H	The port issuing the call does not have a sink port attached.
E_TYPE	8002H	The port_tkn parameter refers to an object that is not a port.

disable

Disables the specified interrupt level. It has no effect on other levels. You must not disable the level reserved for the system clock at system configuration.

See also: For ICU-configurable systems, CIL parameter, *ICU User's Guide and Quick Reference*

Syntax, PL/M and C

```
CALL rq$disable (level, except_ptr);
```

```
rq_disable (level, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`level`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`level` The interrupt level encoded:

Bits	**Value**	**Meaning**
15-7	0	Reserved, set to 0
6-4	0-7	First digit of the interrupt level
3	0	Bits 2-0 specify the second digit (slave)
	1	Bits 6-4 specify the entire level number (master)
2-0	0-7	Second digit of the interrupt level

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code. All condition codes must be processed in-line. Control does not pass to an exception handler.

Additional Information

To be disabled, a level must have an interrupt handler assigned to it. Otherwise, the Nucleus returns an E_CONTEXT condition code.

See also: **enable** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The level indicated by the level parameter is already disabled or has no interrupt handler assigned to it.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The level parameter is invalid.

disable_deletion

Increases by one the disabling depth of an object, making it immune to ordinary deletion.

⚠ **CAUTION**

Do not use **disable_deletion** (consequently there is no need to use **enable_deletion** or **force_delete**) in HI applications. If an HI application contains objects whose disabling depths are greater than one, the application cannot be deleted asynchronously with <Ctrl-C> entered at the keyboard. Your system will have to be rebooted.

Syntax, PL/M and C

```
CALL rq$disable$deletion (object, except_ptr);

rq_disable_deletion (object, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`object`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`object`

A token for the object whose deletion is to be disabled.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If an object's disabling depth is two or greater, it is also immune to forced deletion. If a task attempts to delete the object while it is immune, the task sleeps until the immunity is removed. At that time, the object is deleted and the task is awakened.

If an object has had its deletion disabled, the containing job cannot be deleted until that object has had its deletion reenabled.

Disabling deletion of a suspended task causes the calling task to sleep until the suspended task is resumed.

See also: **enable_deletion** example, **force_delete** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The object parameter is not a token for an existing object.
E_LIMIT	0004H	The object's disabling depth is already 255.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

enable

Enables a specific interrupt level which must have an interrupt handler assigned to it.

See also: **set_interrupt**, **enable** example, Nucleus examples

Syntax, PL/M and C

```
CALL rq$enable (level, except_ptr);
```

```
rq_enable (level, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`level`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`level`

Specifies the interrupt level:

Bits	Value	Meaning
15-7	0	Reserved, set to 0
6-4	0-7	First digit of the interrupt level
3	0	Bits 2-0 specify the second digit (slave)
	1	Bits 6-4 specify the entire level number (master)
2-0	0-7	Second digit of the interrupt level

A task must not enable the level associated with the system clock.

See also: For ICU-configurable systems, CIL parameter, *ICU User's Guide and Quick Reference*

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	At least one of these is true: • A non-interrupt task tried to enable a level that was already enabled. • There is not an interrupt handler assigned to the specified level. • There has been an interrupt overflow on the specified level.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The level parameter is invalid.

enable_deletion

Enables the deletion of objects that have had deletion disabled.

Syntax, PL/M and C

```
CALL rq$enable$deletion (object, except_ptr);
```

```
rq_enable_deletion (object, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`object`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`object`

A token for the object whose deletion is to be enabled.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Enable_deletion decreases by one the disabling depth of an object. If a deletion request is pending against the object, and the **enable_deletion** call makes the object eligible for deletion, the object is deleted and the task that made the deletion request is awakened.

See also: CAUTION in **disable_deletion**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The object's deletion is not disabled.
E_EXIST	0006H	The object parameter is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

end_init_task

Used by an initialization task of a first-level job to inform the root task that it has completed its synchronous initialization process.

Syntax, PL/M and C

```
CALL rq$end$init$task;
```

```
rq_end_init_task();
```

Additional Information

The initialization task of a first-level job must use **end_init_task** to inform the root task that it is finished. This enables the root task to resume execution, creating the next first-level job. You must include this system call in the initialization task of each first-level job, even if the jobs require no synchronous initialization.

For loadable jobs, this system call is ignored.

See also: For ICU-configurable systems, User Jobs screens, *ICU User's Guide and Quick Reference*

enter_interrupt

Used by interrupt handlers to load a previously-specified segment base address into the DS register.

Syntax, PL/M and C

```
CALL rq$enter$interrupt (level, except_ptr);
```

```
rq_enter_interrupt (level, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`level`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`level` Specifies the interrupt level:

Bits	Value	Meaning
15-7	0	Reserved, set to 0
6-4	0-7	First digit of the interrupt level
3	0	Bits 2-0 specify the second digit (slave)
	1	Bits 6-4 specify the entire level number (master)
2-0	0-7	Second digit of the interrupt level

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code. For this system call, all condition codes must be processed in-line. Control does not pass to an exception handler.

Additional Information

Enter_interrupt, on behalf of the calling interrupt handler, loads a base address value into the DS register. This value was specified when the interrupt handler was set up by an earlier call to **set_interrupt**.

If the handler is going to call an interrupt task, **enter_interrupt** enables the handler to place data in the CPU data segment used by the interrupt task. This enables the interrupt handler to pass data to the interrupt task.

See also: Interrupt management, *System Concepts*,
:rmx:demo/c/interrupt directory for demo using **rq_signal_interrupt**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	No segment base value has previously been specified in the call to **set_interrupt**.
E_NOT_CONFIGURED	0008H	This system call is not included in the present configuration.
E_PARAM	8004H	The level parameter is invalid.

exit_interrupt

Used by interrupt handlers to send an end-of-interrupt (EOI) signal to the hardware.

Syntax, PL/M and C

```
CALL rq$exit$interrupt (level except_ptr);

rq_exit_interrupt (level, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`level`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`level` Specifies the interrupt level:

Bits	Value	Meaning
15-7	0	Reserved, set to 0
6-4	0-7	First digit of the interrupt level
3	0	Bits 2-0 specify the second digit (slave)
	1	Bits 6-4 specify the entire level number (master)
2-0	0-7	Second digit of the interrupt level

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code. All condition codes must be processed in-line, as control does not pass to an exception handler.

Additional Information

This call prepares for re-enabling of interrupts. The re-enabling actually occurs when control passes from the interrupt handler to an application task.

See also: *:rmx:demo/c/interrupt* directory for demo using **rq_signal_interrupt**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The **set_interrupt** system call has not been invoked for the specified level.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The level parameter is invalid.

rqe_exit_interrupt

Used by interrupt handlers to send an end-of-interrupt (EOI) signal to the hardware.

Syntax, PL/M and C

```
CALL rqe$exit$interrupt (level, master_base, slave_base);
```

```
rqe_exit_interrupt (level, master_base, slave_base);
```

Parameter	PL/M Data Type	C Data Type
`level`	BYTE	UINT_8
`master_base`	WORD_16	UINT_16
`slave_base`	WORD_16	UINT_16

Parameters

`level` Specifies the interrupt level:

Bits	Value	Meaning
7	0	Reserved, set to 0
6-4	0-7	First digit of the interrupt level
3	0	Bits 2-0 specify the second digit (slave)
	1	Bits 6-4 specify the entire level number (master)
2-0	0-7	Second digit of the interrupt level

`master_base`

The base port address of the master PIC.

`slave_base`

The base port address of the slave PIC whose interrupt is being serviced.

Additional Information

This call is a high performance version of the **exit_interrupt** call. To use it you must know the level of the interrupt being serviced, the base port address of the master PIC, and, if the interrupt level is on a slave PIC, the base port address of the slave PIC.

This system call does not have an `except_ptr` parameter; it does not return status.

force_delete

Deletes objects whose disabling depths are 0 or 1.

Syntax, PL/M and C

```
CALL rq$force$delete (extension, object, except_ptr);
```

```
rq_force_delete (extension, object, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
extension	SELECTOR	SELECTOR
object	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

extension

If the object to be deleted is a composite object, this parameter is a token for the extension type associated with that composite object. Otherwise, this parameter is ignored.

object

A token for the object to be deleted.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If an object has a deletion depth of 2 or more, the calling task is put to sleep until the deletion depth is decreased to 1. At that time, the object is deleted and the task is awakened. If the wrong extension parameter is specified when deleting a composite, **force_delete** issues an E_CONTEXT condition code and returns without deleting the composite.

See also: CAUTION in **disable_deletion**,
OS extensions, *System Concepts*,
force_delete example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The wrong extension type was used in the extension parameter.
E_EXIST	0006H	One or both of the object or extension parameters is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	The extension parameter is not a token for an extension object.

rqe_get_address

Returns the 32-bit physical address of a logical pointer into regular and virtual segments.

Syntax, PL/M and C

```
phys_addr = rqe$get$address (log_addr, except_ptr);
```

```
phys_addr = rqe_get_address (log_addr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
phys_addr	WORD_32	UINT_32
log_addr	POINTER	void far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

phys_addr

The 32-bit physical address of the log_addr parameter.

Parameters

log_addr

A pointer containing the segmented address of the physical address. The segmented address must be in the form selector:offset.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

In Protected Virtual Address Mode (PVAM), the base portion of an address (a selector) does not specify the physical location of the address. Rather, it points to a descriptor table, where the 32-bit physical address is found. This system call retrieves the 32-bit physical address for the selector portion of a pointer, adds the offset part of the pointer to that value, and returns the resulting physical address.

For virtual segments, this call retrieves the physical address from the page tables.

See also: **rqe_get_address** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The segmented address is invalid. Either the selector does not refer to a valid segment, or the offset is outside the segment boundaries.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_NOT_ALLOCATED	00F2H	The offset part of the logical address does not point to a part of the virtual segment that contains physical memory.

get_buffer_limit

Gets the maximum size of a buffer starting from a pointer within a segment. **Get_buffer_limit** works with both regular segments and virtual segments. For virtual segments, **get_buffer_limit** returns the size of either virtual contiguous memory or contiguous physical memory.

Syntax, PL/M and C

```
count = rq$get$buffer$limit (seg, offset, flags_ptr,
     except_ptr);
```

```
count = rq_get_buffer_limit (seg, offset, flags_ptr
     except_ptr);
```

Parameter	PL/M Data Type	C Data Type
count	WORD_32	UINT_32
seg	SELECTOR	SELECTOR
offset	WORD_32	UINT_32
flags_ptr	POINTER	UINT_32 *
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

count

The number of contiguous bytes of memory (either physical or virtual, according to the returned flags).

Parameters

seg A token for the segment containing the buffer. The segment can be a normal segment or a virtual segment. If seg is null and the application is flat model, the parameter indicates the application's virtual segment. For a segmented model application, a null value causes an exception.

offset The offset in seg where the buffer begins.

flags_ptr

A pointer to a variable declared by the application where information about the buffer is returned:

Bit	Meaning
0	0 = normal segment 1 = virtual segment
1	0 = return value is the amount of contiguous physical memory 1 = return value is the amount of contiguous virtual memory

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Get_buffer_limit returns the number of contiguous bytes of memory, starting at seg:offset, that are contained within the segment limit. Status information is returned at flags_ptr. Bit 0 of the flag indicates the type of segment. Bit 1 of the flag indicates the type of memory found at the seg:offset pointer. For normal segments, bit 1 is always set to 0 (physical memory); for virtual segments, bit 1 indicates whether the returned value represents the amount of contiguous physical memory (0) or contiguous virtual memory (1).

The buffer itself is split into the seg and offset parameters to provide maximum flexibility, especially for flat model applications that cannot build a far pointer.

You can use **get_buffer_limit** to determine the size of a buffer or to map the presence of all physical memory within a virtual segment. Use **validate_buffer** to simply validate a buffer.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The offset parameter is beyond the end of the virtual segment.
E_EXIST	0006H	The seg parameter represents a segment that is being deleted, or seg is a null token and the caller is not a flat model application
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The flags_ptr parameter is not valid or is not writable.
E_TYPE	8002H	The seg parameter is not a token for a segment.

get_exception_handler

Returns both the address of the calling task's exception handler and the current value of the task's exception mode.

See also: **rqe_get_exception_handler** system call
get_exception_handler example, **set_exception_handler** example,
Nucleus examples

Syntax, PL/M and C

```
CALL rq$get$exception$handler (except_info_ptr, except_ptr);
```

```
rq_get_exception_handler (except_info_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
except_info_ptr	POINTER	EXCEPTION_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

except_info_ptr

For PL/M, a pointer to this structure:

```
DECLARE exception STRUCTURE (
   exception_handler_ptr  POINTER,
   exception_mode         BYTE);
```

or for C segmented compilers:

```
typedef struct {
   void far *             exception_handler_ptr;
   UINT_8                 exception_mode;
} EXCEPTION_STRUCT;
```

For C flat model compilers only, a pointer to this structure:

```
typedef struct {
   void *                 exception_handler_ptr;
   SELECTOR               exception_handler_ptr_seg;
   UINT_8                 exception_mode;
} EXCEPTION_STRUCT;
```

Where:

`exception_handler_ptr`
If not null, references the first instruction of the current exception handler. If null, the exception handler is the system default exception handler.

`exception_handler_ptr_seg`
For flat model compilers only, the selector for the pointer.

`exception_mode`
Indicates:

Value	When Control Passes To Exception Handler
0	Never, exceptions must be handled in-line
1	On programmer errors only
2	On environmental conditions only
3	On all exceptional conditions

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The pointer is invalid. Either the selector does not refer to a valid segment, or the offset is outside the segment boundaries.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

rqe_get_exception_handler

Returns the address and exception-handling mode for any of the following:

- Current task's exception handler
- Current job's exception handler
- System-wide exception handler
- System-wide hardware exception handler (trap handler)

Syntax, PL/M and C

```
CALL rqe$get$exception$handler (handler_id, info_ptr,
      except_ptr);
```

```
rqe_get_exception_handler (handler_id, info_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
handler_ID	BYTE	UINT_8
info_ptr	POINTER	EXCEPTION_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

handler_id

Specify the exception handler whose address and mode is to be returned, using one of the following values:

Value	Meaning
0	Task's exception handler
1	Job's default exception handler.
2	System-wide exception handler.
3	System-wide hardware exception handler.

info_ptr

For PL/M, a pointer to this structure declared by the application where the call returns information:

```
DECLARE exception STRUCTURE (
   exception_handler_ptr  POINTER,
   exception_mode         BYTE);
```

or for C segmented compilers:

```
typedef struct {
   void far *             exception_handler_ptr;
   UINT_8                 exception_mode;
} EXCEPTION_STRUCT;
```

For C flat model compilers only, a pointer to this structure:

```
typedef struct {
   void *                 exception_handler_ptr;
   SELECTOR               exception_handler_ptr_seg;
   UINT_8                 exception_mode;
} EXCEPTION_STRUCT;
```

Where:

`exception_handler_ptr`
References the first instruction of the exception handler specified by the `handler_id` parameter.

`exception_handler_ptr_seg`
For flat model compilers only, the selector for the pointer.

`exception_mode`
Interpret the mode according to the following table.

Value	**Meaning**
0	Handler receives control only on hardware exceptions, programmer and environmental exceptions must be handled in-line
1	Handler receives control only on programmer errors and hardware exceptions
2	Handler receives control only on environmental conditions and hardware exceptions
3	Handler receives control on all exceptional conditions
12	Hardware exception handler deletes the offending job
13	Hardware exception handler deletes the offending task
14	Hardware exception handler suspends the offending task
15	Hardware exception handler breaks to the SDM debug monitor

Modes 12 and higher apply only to the default system-wide hardware exception handler. If you have specified your own handler for the task, it receives control on a hardware exception for modes 12 - 14, but can return to the default hardware exception handler on exit. If the default hardware exception handler is set to mode 15, your exception handler does not receive control on a hardware exception.

See also   Exception Handlers, *System Concepts*

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The system-wide exception handler refers to the root job's exception handler. When you change the system-wide exception handler, it changes only the default exception handler that is inherited by first-level jobs created by the root job.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.

get_host_id

Returns the host ID of the board on which the calling task is running.

Syntax, PL/M and C

```
host_id = rq$get$host$id (except_ptr);

host_id = rq_get_host_id (except_ptr);
```

Parameter	PL/M Data Type	C Data Type
host_id	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

host_id

A number that identifies the board as a message passing host.

Parameters

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Use this system call to construct sockets to be used as return addresses for messages.

See also: Host ID, socket, *System Concepts*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

get_interconnect

Returns the contents of the specified Multibus II interconnect register.

Syntax, PL/M and C

```
value = rq$get$interconnect (slot_number, reg_number,
      except_ptr);
```

```
value = rq_get_interconnect (slot_number, reg_number,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
value	BYTE	UINT_8
slot_number	BYTE	UINT_8
reg_number	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

`value`

The contents of the interconnect register. If the cardslot is empty, 0 returns for PSB cardslots and 0FFH for Local Bus Extension (iLBX II) cardslots.

Parameters

`slot_number`

Specifies the Multibus II cardslot number of the board on which the specified interconnect register is located:

Value	Meaning
0-19	PSB slot numbers 0 to 19
20-23	Reserved, do not specify these values
24-29	iLBX II cardslot numbers 0 to 5
30	Reserved
31	Retrieve the contents of a local interconnect register (from the board where the calling task is running)

`reg_number`

Specifies the interconnect register to read. This value must be in the range 0000H to 01FFH. Refer to the hardware manual for your Multibus II board for definitions of its interconnect registers.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The Nucleus performs range-checking of the cardslot and register numbers specified in the call, but does not verify the existence of a board in any specific cardslot. Nor does it assign any meaning to the register being accessed.

See also: Interconnect space example, *Programming Techniques*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	One or more of the parameters has an illegal value.

get_level

Returns to the calling task the highest (numerically lowest) level that an interrupt handler has started servicing but has not yet finished. **Get_level** can only be called by a handler.

See also: *:rmx:demo/c/interrupt* directory for demo using **rq_signal_interrupt**

Syntax, PL/M and C

```
level = rq$get$level (except_ptr);

level = rq_get_level (except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`level`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`level`

The interrupt level indicates:

Bits	Value	Meaning
15-8	0	Reserved, set to 0
7	0	Bits 6-0 are significant (service in progress)
	1	Bits 6-0 are not significant (no service is in progress)
6-4	0-7	First digit of the interrupt level
3	0	Bits 2-0 specify the second digit (slave)
	1	Bits 6-4 specify the entire level number (master)
2-0	0-7	Second digit of the interrupt level

Parameter

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

rqe_get_object_access

Returns the access type of an object whose token is specified.

Syntax, PL/M and C

```
CALL rqe$get$object$access (object, access_ptr, except_ptr);
```

```
rqe_get_object_access (object, access_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
object	SELECTOR	selector
access_ptr	POINTER	ACCESS_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

object

A token for an object whose access rights you want to see.

access_ptr

A pointer to this structure:

```
DECLARE access_struct STRUCTURE (
   access                 BYTE,
   limit_mode             BYTE);
```

or

```
typedef struct {
   UINT_8                 access;
   UINT_8                 limit_mode;
} ACCESS_STRUCT;
```

Where:

`access` Gives the access rights for the object. These values are typical:

Data Segments	**Binary Value**	**Hex Value**
Read-only	10010000B	90H
Read/write	10010010B	92H
Code Segments	**Binary Value**	**Hex Value**
Execute-only	10011000B	98H
Execute/read	10011010B	9AH
Execute only (conforming)	10011100B	9CH
Execute/read (conforming)	10011110B	9EH

The bits are defined as:

Bits	**Meaning**
7	Present bit, 1 = valid descriptor, normally set to 1.
6-5	Descriptor privilege level (DPL), normally set to 0.
4	1 = segment descriptor.
3	0 = data segment. 1 = code segment.
2	For data segments, 1 = expand down (ED) bit. Normally set to 0. For code segments, 1 = conforming segment.
1	0 = read-only. 1 = write access.
0	Must be set to 0.

`limit_mode`

Specifies information on segment granularity and type for use by the processor in limit checking. These values are typical:

Binary	**Hex**	**Meaning**
00000000B	0	1 byte granularity, 16-bit segment
01000000B	40H	1 byte granularity, 32-bit segment
10000000B	80H	4 Kbyte granularity, 16-bit segment
11000000B	C0H	4 Kbyte granularity, 32-bit segment

The bits are defined as:

Bits	**Meaning**
7	0 = one byte segment granularity. 1 = 4 Kbyte granularity
6	0 = 16-bit segment. 1 = 32-bit segment
5	Set to 0.
4	Available for programmer use.
3-0	Set to 0.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Rqe_get_object_access lets you view an object descriptor's access rights. You can use **rqe_change_object_access** to change this information.

See also: Descriptors, composite objects, *System Concepts*,
create_segment example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The access_ptr pointer is invalid. Either the selector does not refer to a valid segment, or the offset is outside the segment boundaries.
E_EXIST	0006H	The object whose access is requested does not exist or is not a valid iRMX object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

get_pool_attrib

Obsolete. Returns information about the memory pool of the calling task's job. This system call can report pool sizes no larger than 1 Mbyte. Provided for compatibility. Use **rqe_get_pool_attrib**.

Syntax, PL/M and C

```
CALL rq$get$pool$attrib (attrib_ptr, except_ptr);
```

```
rq_get_pool_attrib (attrib_ptr, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
attrib_ptr	POINTER	POOL_ATTRIB_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

attrib_ptr

A pointer to this structure:

```
DECLARE pool_attrib STRUCTURE (
   pool_max               WORD_16,
   pool_min               WORD_16,
   initial_size           WORD_16,
   allocated              WORD_16,
   available              WORD_16);
```

or

```
typedef struct {
   UINT_16                pool_max;
   UINT_16                pool_min;
   UINT_16                initial_size;
   UINT_16                allocated;
   UINT_16                available;
} POOL_ATTRIB_STRUCT;
```

Where:

pool_max The maximum allowable size of the memory pool in 16-byte paragraphs.

pool_min The minimum allowable size of the memory pool in 16-byte paragraphs.

`initial_size`
The original value of the `pool_min` attribute.

`allocated` The number of 16-byte paragraphs currently allocated from the memory pool.

`available` The number of 16-byte paragraphs currently available in the memory pool. It does not include memory that could be borrowed from the parent job. The memory indicated may be fragmented and thus not allocatable as a single segment.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This system call cannot return accurate size information about memory pools that are larger than 1 Mbyte. To get accurate information concerning memory pools over 1 Mbyte, use the **rqe_get_pool_attrib** system call.

See also: **get_pool_attrib** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The attrib_ptr pointer is invalid. Either the selector does not refer to a valid segment, or the offset is outside the segment boundaries.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

rqe_get_pool_attrib

Returns information about the memory pool of any job you specify.

Syntax, PL/M and C

```
CALL rqe$get$pool$attrib (attrib_ptr, except_ptr);
```

```
rqe_get_pool_attrib (attrib_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
attrib_ptr	POINTER	E_POOL_ATTRIB_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

attrib_ptr

A pointer to this structure. The calling task specifies the target_job field; all other fields are filled in by the call.

```
DECLARE e_pool_attrib STRUCTURE (
   target_job            SELECTOR,
   parent_job            SELECTOR,
   pool_max              WORD_32,
   pool_min              WORD_32,
   initial_size          WORD_32,
   allocated             WORD_32,
   available             WORD_32,
   borrowed              WORD_32);
```

or

```
typedef struct {
   SELECTOR              target_job;
   SELECTOR              parent_job;
   UINT_32               pool_max;
   UINT_32               pool_min;
   UINT_32               initial_size;
   UINT_32               allocated;
   UINT_32               available;
   UINT_32               borrowed;
} E_POOL_ATTRIB_STRUCT;
```

Where:

`target_job`
The token for the job whose memory pool you want to examine. A null selector indicates the calling task's job.

`parent_job`
A token for the parent job of the specified target job.

`pool_max` The maximum allowable size of the target job's memory pool in 16-byte paragraphs.

`pool_min` The minimum allowable size of the target job's memory pool in 16-byte paragraphs.

`initial_size`
The original value of the `pool_min` attribute when the job was created.

`allocated` The number of 16-byte paragraphs currently allocated from the target job's memory pool.

`available` The number of 16-byte paragraphs currently available in the target job's memory pool. It does not include memory that could be borrowed from the parent job.

Note
The memory indicated might be fragmented and thus not allocatable as a single segment.

`borrowed` The amount of memory that the target job has borrowed from the parent job in 16-byte paragraphs.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This call is similar to **get_pool_attrib**, except that it handles pool sizes larger than 1 Mbyte, returns the amount of memory borrowed from the parent job, and can return information about any job.

See also: **rqe_get_pool_attributes** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The attrib_ptr pointer is invalid. Either the selector does not refer to a valid segment, or the offset is outside the segment boundaries.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The target_job field, of E_POOL_ATTRIB_STRUCT is not a valid token.
E_TYPE	8002H	The token for the target job is not a job token.

get_port_attributes

Returns information about how the specified port is set up.

Syntax, PL/M and C

```
CALL rq$get$port$attributes (port_tkn, info_ptr, except_ptr);
```

```
rq_get_port_attributes (port_tkn, info_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
port_tkn	SELECTOR	SELECTOR
info_ptr	POINTER	PORT_ATTRIB_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

port_tkn

A token for the port about which you need information.

info_ptr

A pointer to this structure:

```
DECLARE port_attrib STRUCTURE (
   port_id                WORD_16,
   type                   BYTE,
   reserved_a             BYTE,
   num_trans              WORD_16,
   reserved (2)           WORD_16,
   sink_port              SELECTOR,
   default_remote_socket  WORD_32,
   buffer_pool            SELECTOR,
   flags                  WORD_16,
   reserved_b             BYTE);
```

or

```
typedef struct {
   UINT_16              port_id;
   UINT_8               type;
   UINT_8               reserved_a;
   UINT_16              num_trans;
   UINT_16              reserved[2];
   SELECTOR             sink_port;
   UINT_32              default_remote_socket;
   SELECTOR             buffer_pool;
   UINT_16              flags;
   UINT_8               reserved_b;
} PORT_ATTRIB_STRUCT;
```

Where:

`port_id` Uniquely identifies the port.

`type` Specifies the type of messages that can be sent to and from this port:

Value	Meaning
0-1	Reserved
2	Data messages
3	Signal messages
4-0FFH	Reserved

`reserved_a`
Reserved, set to 0.

`num_trans` Specifies the number of simultaneous transactions that can be outstanding at this port.

`reserved[2]`
Reserved, set to 0.

`sink_port` If not 0, a token for the port that receives forwarded messages from the port you are examining. This indicates that the **attach_port** call has been invoked.

`default_remote_socket`
If not 0, specifies the default destination/source for all message exchanges at this port. This indicates that the **connect** call has been invoked.

`buffer_pool`
If not 0, a token for the buffer pool attached to this port. This indicates that the **attach_buffer_pool** system call has been invoked.

See also: **attach_port**, **connect**, **attach_buffer_pool** `flags` A value indicates:

Bits	Meaning
15-3	Reserved
2	0 = RSVP request message fragmentation enabled 1 = fragmentation disabled
1	0 = FIFO message queue 1 = priority message queue
0	Reserved

`reserved_b`

Reserved, set to 0.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The port_tkn parameter does not refer to an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NUC_BAD_BUF	80E2H	The info_ptr parameter is invalid or points to a buffer that is not large enough.
E_TYPE	8002H	The port_tkn parameter refers to an object that is not a port.

get_priority

Returns the priority of the specified task.

Syntax, PL/M and C

```
priority = rq$get$priority (task, except_ptr);

priority = rq_get_priority (task, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
priority	BYTE	UINT_8
task	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

priority

The priority of the task indicated by the task parameter.

Parameters

task One of these:

Value	Meaning
Null selector	Calling task is asking for its own priority.
Valid selector	Token for the task whose priority is being requested

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The task parameter is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	The task parameter is not a token for a task.

get_size

Returns the size, in bytes, of a regular or virtual iRMX segment.

Syntax, PL/M and C

```
seg_size = rq$get$size (segment, except_ptr);

seg_size = rq_get_size (segment, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
seg_size	NATIVE_WORD	NATIVE_WORD
segment	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

seg_size

Indicates:

Value	Meaning
0	For 16-bit tasks only, 64 Kbyte segment size
1-0FFFFH	Actual segment size in bytes
10000H	For 32-bit tasks only, 64-Kbyte segment size
10001H-0FFFFFFFFH	Actual segment size rounded up to the nearest multiple of 4 Kbytes

Parameters

segment

A token for a segment whose size is desired.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Segments can be a maximum length of 4 Gbytes for 32-bit tasks and 16 Mbytes (less 1 byte) for 16-bit tasks.

See also: **create_segment** example, Nucleus examples

For virtual segments only, the size returned by this call has no relationship to the actual amount of physical memory allocated to the virtual segment. It is simply the size of the virtual address space encompassed by the virtual segment. Use the **rq_validate_buffer** and **rq_get_buffer_limit** calls to obtain the size of physical memory allocated to a virtual segment.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The `segment` parameter is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_TYPE	8002H	The `segment` parameter is not a token for a segment.

get_task_accounting

Returns information about when a task was created and the amount of time the task has run.

See also: **rq_system accounting** to enable tracking of such information

Syntax, PL/M and C

```
CALL rq_get_task_accounting (target_task, info_ptr, reset_opt,
      except_ptr);
```

```
rq_get_task_accounting (target_task, info_ptr, reset_opt,
      except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
target_task	SELECTOR	SELECTOR
info_ptr	POINTER	TASK_ACCOUNTING_STRUCT far *
reset_opt	BYTE	UINT_8
except_ptr	POINTER to WORD_16	UINT_16 *

Parameters

target_task

The token for the task for which to return accounting information.

info_ptr

A pointer to the following structure declared by the application, where the call returns information.

```
DECLARE task_accounting_struct STRUCTURE (
   owner_job                     SELECTOR,
   next_task                     SELECTOR,
   accounting_state              WORD_16,
   usecs_per_tick                WORD_16,
   create_time_lo                WORD_32,
   create_time_hi                WORD_32,
   elapsed_time_lo               WORD_32,
   elapsed_time_hi               WORD_32,
   total_ticks_since_call_lo     WORD_32,
   running_ticks_since_call_lo   WORD_32,
   total_running_ticks_lo        WORD_32,
   total_running_ticks_hi        WORD_32);
```

or

```
typedef struct {
   SELECTOR              owner_job;
   SELECTOR              next_task;
   UINT_16               accounting_state;
   UINT_16               usecs_per_tick;
   UINT_32               create_time_lo;
   UINT_32               create_time_hi;
   UINT_32               elapsed_time_lo;
   UINT_32               elapsed_time_hi;
   UINT_32               total_ticks_since_call_lo;
   UINT_32               running_ticks_since_call_lo;
   UINT_32               total_running_ticks_lo;
   UINT_32               total_running_ticks_hi;
   } TASK_ACCOUNTING_STRUCT;
```

Where:

`owner_job` Token for the job containing the target task.

`next_task` Next task on the system task list, which allows easy scanning of all the tasks in the system.

`accounting_state`
If 0, only the fields up through `usecs_per_tick` are valid. If non-zero, all fields in the structure are valid.

`usecs_per_tick`
The number of microseconds that each Kernel tick represents.

`create_time_lo`
The low 32 bits of the target task's creation time.

`create_time_hi`
The high 32 bits of the target task's creation time.

`elapsed_time_lo`
The low 32 bits of the time that has elapsed since the target task was created.

`elapsed_time_hi`
The high 32 bits of the time that has elapsed since the target task was created.

`total_ticks_since_call_lo`
The low 32 bits of the elapsed time since the last call to **rq_get_task_accounting** for this task, as measured in Kernel ticks.

`running_ticks_since_call_lo`
The low 32 bits of the amount of time the task has run since the last call to **rq_get_task_accounting** for this task, as measured in Kernel ticks.

`total_running_ticks_lo`
The low 32 bits of the amount of time the task has run since its creation, as measured in Kernel ticks.

`total_running_ticks_hi`
The high 32 bits of the amount of time the task has run since its creation, as measured in Kernel ticks.

`reset_opt`
Specifies whether to reset this task's accounting information as part of this call or to accumulate information since the last call.

Value	**Meaning**
0	Accumulate information since the last call
0FFH	Reset the information with this call

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Although the time since the last call to **rq_get_task_accounting** is kept internally as a 64 bit value, it is only returned in the call as a 32 bit quantity. Therefore, you must either call often enough to avoid overrunning the `total_ticks_since_call` and `running_ticks_since_call` fields (which are only the lower 32 bits) or use the `total_running_ticks` fields to derive the appropriate information.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.

get_task_info

Returns information about a task, including such items as priority, exception handler, containing job, and execution state.

See also: **rq_get_task_accounting** and **rq_get_task_state** for other information

Syntax, PL/M and C

```
CALL rq_get_task_info (target_task, info_ptr, except_ptr);
```

```
rq_get_task_info (target_task, info_ptr, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`target_task`	SELECTOR	SELECTOR
`info_ptr`	POINTER	TASK_INFO_STRUCT far *
`except_ptr`	POINTER to WORD_16	UINT_16 *

Parameters

`target_task`

The token for the task for which to return information.

`info_ptr`

A pointer to the following structure declared by the application, where the call returns information.

```
DECLARE task_info_struct STRUCTURE (
   owner_job                  SELECTOR,
   next_task                  SELECTOR,
   exception_handler          POINTER,
   exception_mode             BYTE,
   fill0                      BYTE,
   static_priority            BYTE,
   dynamic_priority           BYTE,
   task_flags                 BYTE,
   interrupt_task             BYTE,
   pending_interrupts         BYTE,
   max_interrupts             BYTE,
   int_level                  WORD_16,
   task_state                 BYTE,
   suspend_depth              BYTE,
   delay_request              WORD_16,
   last_exchange              SELECTOR);
```

or

```
typedef struct {
   SELECTOR               owner_job;
   SELECTOR               next_task;
   void far *             exception_handler;
   UINT_8                 exception_mode;
   UINT_8                 fill0;
   UINT_8                 static_priority;
   UINT_8                 dynamic_priority;
   UINT_8                 task_flags;
   UINT_8                 interrupt_task;
   UINT_8                 pending_interrupts;
   UINT_8                 max_interrupts;
   UINT_16                int_level;
   UINT_8                 task_state;
   UINT_8                 suspend_depth;
   UINT_16                delay_request;
   SELECTOR               last_exchange;
   } TASK_INFO_STRUCT;
```

Where:

`owner_job` Token for the job containing the target task.

`next_task` Next task on the system task list, which allows easy scanning of all the tasks in the system.

`exception_handler`
Pointer to the task's current exception handler. For flat model applications only, treat this parameter as two separate fields in the structure. The first field has the name listed above and is a near pointer. The second field has the same name with `_seg` appended at the end. It is a segment selector for the pointer.

`fill1` Reserved

`static_priority`
The task's assigned priority when it was created.

`dynamic_priority`
The task's current priority, which can be dynamically raised by accessing a region.

`task_flags`
The task flags specified when the task was created.

`interrupt_task`
If non-zero, the task is an interrupt task and the next three fields are valid. If 0, the task is not an interrupt task; ignore the next three fields.

`pending_interrupts`

Number of interrupts currently pending at the interrupt level associated with this interrupt task.

`max_interrupts`

Maximum number of interrupts that can be pending at the interrupt level associated with this interrupt task.

`int_level`

Interrupt level associated with this interrupt task.

`task_state`

One of the following indicates the task's current state:

Value	**Meaning**
0H	Ready and running
1H	Ready and not running
2H	Asleep
3H	Waiting at an exchange object
4H	Waiting at a region
5H	Waiting at an object directory
6H	Waiting at a port
7H	Being deleted
10H	Suspended
12H	Asleep/Suspended
13H	Waiting at an exchange and Suspended
14H	Waiting at a region and Suspended
15H	Waiting at an object directory and Suspended
16H	Waiting at a port and Suspended
17H	Being deleted and Suspended
0FFH	Task state unknown

`suspend_depth`

Suspension depth of the task, which is non-zero only if the task has been overtly suspended (as opposed to being suspended by the OS).

`delay_request`

Amount of time the task has been waiting at an exchange. This field is zero if the task has been waiting at any other type of object.

`last_exchange`

The token for an exchange object (e.g., mailbox or semaphore) at which the task is waiting. This field is zero if the task is not waiting at an exchange.

```
except_ptr
```

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.

get_task_state

Returns information about the state of any task in the system, including such items as the execution state and the CPU registers for that task's execution context. Since the full task context is preserved only when the task has been pre-empted, the CPU register information is available only for tasks in the ready state or the suspended state where the task has not suspended itself. Thus the primary purpose for this call is to examine a task that has received an exception with its exception handler mode set to suspend the faulting task.

See also: **rq_get_task_accounting** and **rq_get_task_state** for other information

Syntax, PL/M and C

```
CALL rq_get_task_state (target_task, info_ptr, except_ptr);
```

```
rq_get_task_state (target_task, info_ptr, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
target_task	SELECTOR	SELECTOR
info_ptr	POINTER	TASK_STATE_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 *

Parameters

target_task

The token for the task for which to return information.

info_ptr

A pointer to the following structure declared by the application, where the call returns information.

```
DECLARE task_state_struct STRUCTURE (
   owner_job                SELECTOR,
   next_task                SELECTOR,
   task_state               BYTE,
   suspend_depth            BYTE,
   delay_request            WORD_16,
   last_exchange            SELECTOR,
   cpu_frame                CPU_FRAME_STRUCT);
```

or

```
typedef struct {
   SELECTOR                owner_job;
   SELECTOR                next_task;
   UINT_8                  task_state;
   UINT_8                  suspend_depth;
   UINT_16                 delay_request;
   SELECTOR                last_exchange;
   CPU_FRAME_STRUCT        cpu_frame
   } TASK_STATE_STRUCT;
```

Where:

`owner_job` Token for the job containing the target task.

`next_task` Next task on the system task list, which allows easy scanning of all the tasks in the system.

`task_state`

One of the following indicates the task's current state:

Value	**Meaning**
0H	Ready and running
1H	Ready and not running
2H	Asleep
3H	Waiting at an exchange object
4H	Waiting at a region
5H	Waiting at an object directory
6H	Waiting at a port
7H	Being deleted
10H	Suspended
12H	Asleep/Suspended
13H	Waiting at an exchange and suspended
14H	Waiting at a region and suspended
15H	Waiting at an object directory and suspended
16H	Waiting at a port and suspended
17H	Being deleted and suspended
0FFH	Task state unknown

`suspend_depth`

Suspension depth of the task, which is non-zero only if the task has been overtly suspended (as opposed to being suspended by the OS).

`delay_request`

Amount of time the task has been waiting at an exchange. This field is zero if the task has been waiting at any other type of object.

`last_exchange`

Token for an exchange object (e.g., mailbox or semaphore) at which the task is waiting. This field is zero if the task is not waiting at an exchange.

`cpu_frame`

The CPU register context for the task. This information is available only for tasks in the ready state or the suspended state when the task has not suspended itself. The information is returned in the following structure:

```
DECLARE cpu_frame_struct STRUCTURE (
   running_task        SELECTOR,
   fill0               UINT_16,
   reg_cr2             WORD_32,
   reg_gs              SELECTOR,
   fill1               WORD_16,
   reg_fs              SELECTOR,
   fill2               WORD_16,
   reg_es              SELECTOR,
   fill3               WORD_16,
   reg_ds              SELECTOR,
   fill4               WORD_16,
   reg_ldt             SELECTOR,
   fill5               WORD_16,
   reg_eax             WORD_32,
   reg_ecx             WORD_32,
   reg_edx             WORD_32,
   reg_ebx             WORD_32,
   reg_esp             WORD_32,
   reg_ebp             WORD_32,
   reg_esi             WORD_32,
   reg_edi             WORD_32,
   error_code          WORD_32,
   ret_eip             WORD_32,
   ret_cs              SELECTOR,
   fill6               WORD_16,
   eflags              WORD_32,
   ret_esp             WORD_32,
   ret_ss              SELECTOR,
   fill7               WORD_16);
```

or

```
typedef struct {
   SELECTOR            running_task;
   UINT_16             fill0;
   UINT_32             reg_cr2;
   SELECTOR            reg_gs;
   UINT_16             fill1;
   SELECTOR            reg_fs;
   UINT_16             fill2;
   SELECTOR            reg_es;
   UINT_16             fill3;
   SELECTOR            reg_ds;
   UINT_16             fill4;
   SELECTOR            reg_ldt;
   UINT_16             fill5;
   UINT_32             reg_eax;
   UINT_32             reg_ecx;
   UINT_32             reg_edx;
   UINT_32             reg_ebx;
   UINT_32             reg_esp;
   UINT_32             reg_ebp;
   UINT_32             reg_esi;
   UINT_32             reg_edi;
   UINT_32             error_code;
   UINT_32             ret_eip;
   SELECTOR            ret_cs;
   UINT_16             fill6;
   UINT_32             eflags;
   UINT_32             ret_esp;
   SELECTOR            ret_ss;
   UINT_16             fill7;
} CPU_FRAME_STRUCT;
```

Where:

`running_task` Token for the task whose CPU registers are provided.

`fill0` Reserved.

`reg_gs` The GS register.

`fill1` Reserved.

`reg_fs` The FS register.

`fill2` Reserved.

`reg_es` The ES register.

`fill3`	Reserved.
`reg_ds`	The DS register.
`fill4`	Reserved.
`reg_ldt`	The LDTR register.
`fill5`	Reserved.
`reg_eax`	The EAX register.
`reg_ecx`	The ECX register.
`reg_edx`	The EDX register.
`reg_ebx`	The EBX register.
`reg_esp`	The ESP register.
`reg_ebp`	The EBP register.
`reg_esi`	The ESI register.
`reg_edi`	The EDI register.
`reg_cr2`	The CR2 register. This field is only valid in the context of an exception handler.
`error_code`	Error code returned by the processor. This field is only valid in the context of an exception handler.
`ret_eip`	The EIP register.
`ret_cs`	The CS register.
`fill6`	Reserved.
`eflags`	The EFLAGS register.
`ret_esp`	The ESP register.
`ret_ss`	The SS register.
`fill7`	Reserved.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.

get_task_tokens

Returns a token for either the calling task, the calling task's job, the parameter object of the calling task's job, the root job, or the parent job of the calling task's job, depending on the encoded request.

See also: **get_task_tokens** example,
create_task example, Nucleus examples

Syntax, PL/M and C

```
object = rq$get$task$tokens (selection, except_ptr);
```

```
object = rq_get_task_tokens (selection, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`object`	SELECTOR	SELECTOR
`selection`	BYTE	UINT_8
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`object`

The requested token.

Parameters

`selection`

Selects the type of token to be returned.

Value	**Token Returned**
0	Calling task
1	Calling task's job
2	Parameter object of the calling task's job
3	Root job
4	Parent job of the calling task's job

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_PARAM	8004H	The selection parameter is outside the range 0-4.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.

get_time

Returns the date/time value from the BIOS's local clock.

Syntax, PL/M and C

```
date_time = rq$get$time (except_ptr);

date_time = rq_get_time (except_ptr);
```

Parameter	PL/M Data Type	C Data Type
date_time	WORD_32	UINT_32
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

date_time

Contains a date/time value expressed as the number of seconds since midnight, January 1, 1978.

Parameters

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The UDI and HI follow the convention that January 1, 1978 is equal to 0 seconds. When the date_time value reaches its maximum of 0FFFFFFFFH, it will stop incrementing and will not roll over to start again from 0.

See also: UDI call **dq_decode_time**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

get_type

Returns the type code for an object.

See also: **get_type** example, Nucleus examples

Syntax, PL/M and C

```
type_code = rq$get$type (object, except_ptr);
```

```
type_code = rq_get_type (object, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`type_code`	WORD_16	UINT_16
`object`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`type_code`

The encoded type of the specified object.

Value	**Type**
001H	job
002H	task
003H	mailbox
004H	semaphore
005H	region
006H	segment
007H	extension
009H	port
00AH	buffer pool
100H	user composite
101H	connection composite
300H	I/O job composite
301H	logical device composite
8000H-0FFFFH	user-created composites

See also: Composites, I/O jobs, *System Concepts*

Parameters

`object`

A token for an object whose type is to be returned.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The object parameter is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.

inspect_composite

Accepts a token for a composite object and returns a list of tokens for the components of the composite object.

Syntax, PL/M and C

```
CALL rq$inspect$composite (extension, composite,
      token_list_ptr, except_ptr);
```

```
rq_inspect_composite (extension, composite, token_list_ptr,
      except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
extension	SELECTOR	SELECTOR
composite	SELECTOR	SELECTOR
token_list_ptr	POINTER	TOKEN_LIST_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

extension

A token for the extension object used by the composite object being inspected.

composite

A token for the composite object being inspected.

token_list_ptr

A pointer to this structure:

```
DECLARE token_list STRUCTURE (
   num_slots              WORD_16,
   num_used               WORD_16,
   tokens(*)              SELECTOR);
```

or

```
typedef struct {
   UINT_16                num_slots;
   UINT_16                num_used;
   SELECTOR               tokens[_NUM_TOKENS];
                                     /* adjust # of tokens */
} TOKEN_LIST_STRUCT;
```

Where:

`num_slots` A field where the calling task specifies the number of positions available for tokens in the token list.

`num_used` The actual number of component tokens making up the composite object.

`tokens` An array of tokens that constitute the composite object.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The calling task must supply the `num_slots` value in the data structure pointed to by `token_list_ptr`. The Nucleus fills in the remaining fields. If `num_slots` is set to 0, the Nucleus fills in only the `num_used` field.

If the `num_slots` value is smaller than the actual number of component tokens, only that number (`num_slots`) of tokens will be returned.

See also: CAUTION in **create_composite**,
Component objects, composite objects, extension objects, type manager, *System Concepts*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The pointer to the token_list_ptr structure is invalid. Either the selector does not refer to a valid segment, or the offset is outside the segment boundaries.
E_CONTEXT	0005H	The composite parameter is not compatible with the extension parameter.
E_EXIST	0006H	The composite and/or extension parameter(s) is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	One or both of the extension or composite parameters is not a token for an object of the correct type.

lookup_object

Returns the token for an object after searching for its name in the specified object directory.

See also: **get_type** example, Nucleus examples

Syntax, PL/M and C

```
object = rq$lookup$object (job, name_ptr, time_limit,
      except_ptr);
```

```
object = rq_lookup_object (job, name_ptr, time_limit,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
object	SELECTOR	SELECTOR
job	SELECTOR	SELECTOR
name_ptr	POINTER	void far *
time_limit	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

object

The requested object token.

Parameters

job One of these:

Value	Meaning
Null selector	Search the calling task's object directory.
Valid selector	Token for the job whose object directory is to be searched.

name_ptr

A pointer to a STRING containing the name under which the object is cataloged. The lookup operation is case sensitive.

`time_limit`

Specifies the task's willingness to wait. If the object is not yet cataloged, the calling task has the option of waiting for another task to catalog the object.

Value	Meaning
0	Do not wait.
1-65534	Wait this number of clock intervals.
65535	Wait forever.

See also: For ICU-configurable systems, CIN parameter, *ICU User's Guide and Quick Reference*

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The pointer to the name string is invalid. Either the selector does not refer to a valid segment, or the offset is outside the segment boundaries. This code is not returned by the DOS RTE.
E_CONTEXT	0005H	The specified job has an object directory of size 0.
E_EXIST	0006H	At least one of these is true: • The `job` parameter (if not a null selector) is not a token for an existing object. • The name was found, but the cataloged object has a null token.
E_LIMIT	0004H	The specified object directory is full and the object being looked up has not yet been cataloged. This code (rather than E_TIME) is returned when a full object directory does not contain the requested object and the calling task is not willing to wait.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

E_PARAM	8004H	One of these: • The first byte of the STRING pointed to by the name_ptr parameter has a value outside the range 1-12. • The call was made as an RTE call from Windows, with a time_limit parameter greater than 1. This limitation does not apply to RTE calls made from DOS.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_TIME	0001H	One of these is true: • The calling task was willing to wait a certain amount of time, but the waiting period elapsed before the object became available. • The task was not willing to wait, the entry indicated by the name_ptr parameter is not in the specified object directory, and the object directory is not full.
E_TYPE	8002H	The job parameter is not a token for a job.

move_data

Copies bytes from one buffer to another.

Syntax, PL/M and C

```
actual = rq$move$data (src_seg, src_offset, dest_seg,
      dest_offset, count, except_ptr);
```

```
actual = rq_move_data (src_seg, src_offset, dest_seg,
      dest_offset, count, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`actual`	WORD_32	UINT_32
`src_seg`	SELECTOR	SELECTOR
`src_offset`	WORD_32	void near *
`dest_seg`	SELECTOR	SELECTOR
`dest_offset`	WORD_32	void near *
`count`	WORD_32	UINT_32
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`actual` The number of bytes actually copied.

Parameters

`src_seg`

A token for the source segment. If this parameter is null and the application is flat model, the parameter indicates the application's virtual segment. For segmented model applications, a null value is an error.

`src_offset`

The location within the source segment where copying is to begin.

`dest_seg`

A token for the destination segment. If this parameter is null and the application is flat model, the parameter indicates the application's virtual segment. For segmented model applications, a null value is an error.

`dest_offset`

The location within the destination segment where copying is to begin.

`count` The number of bytes to copy.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Move_data moves `count` bytes at `src_seg:src_offset` to a buffer starting at `dest_seg:dest_offset` and returns the actual number of bytes copied. This call can be used by a flat model application to move data to and from normal iRMX segments since flat model applications cannot build a far pointer to iRMX segments. **Move_data** fails if either offset is beyond the end of its segment. It also fails if either iRMX segment token is invalid or `dest_seg` is not writable. If the returned value is less than the requested count, the end of one of the segments was encountered.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	One of `src_seg` or `dest_seg` is an invalid token, or the `dest_seg:dest_offset` pair is not writable for some or all of its length, or one of the `offset` parameters is beyond the length or its respective segment.
E_EXIST	0006H	One of the `src_seg` or `dest_seg` parameters represents a segment that is being deleted, or one of them is a null token and the caller is not a flat model application
E_NOT_ALLOCATED	00F2H	The segment given by `src_seg` or `dest_seg` is a virtual segment that does not have physical memory allocated to somewhere between the `offset` and the `offset` plus the `count`.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	Either the `src_seg` or the `dest_seg` parameter is not a token for a segment.

offspring

Returns tokens for the child jobs of the specified job.

Syntax, PL/M and C

```
token_list = rq$offspring (job, except_ptr);

token_list = rq_offspring (job, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
token_list	SELECTOR	SELECTOR
job	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

token_list

A null selector indicates that the specified job has no children. If a valid selector, a token for a segment structure:

```
DECLARE child_jobs STRUCTURE (
   actual                 WORD_16,
   children(*)            SELECTOR);
```

or

```
typedef struct {
   UINT_16                actual;
   SELECTOR               children[_NUM_CHILDREN];
                                       /* adjust to actual */
} CHILD_JOBS_STRUCT;
```

Where:

actual — The actual number of child job tokens.

children — The child job token list.

Parameters

`job` A token for the job whose offspring are desired. A null selector specifies the calling task's job.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

By repeated use of this call, you can obtain tokens for all descendants of a job; this information is needed by a task which is attempting to delete a job that has offspring. The return segment is created in the calling task's job and counts against its object limit.

See also: **offspring** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The job parameter is not a token for an existing object.
E_LIMIT	0004H	The calling task's job has already reached its object limit and a return segment could not be created.
E_MEM	0002H	The memory available to the specified job is not sufficient to complete this call.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SLOT	000CH	There isn't enough room in the GDT for another descriptor.
E_TYPE	8002H	The job parameter is not a token for a job.

rqe_offspring

Returns tokens for the child jobs of the specified job. Unlike the **offspring** system call, **rqe_offspring** returns the list of child job tokens in a user-supplied structure rather than in a segment.

Syntax, PL/M and C

```
CALL rqe$offspring (job, list_ptr, except_ptr);
```

```
rqe_offspring (job, list_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
job	SELECTOR	SELECTOR
list_ptr	POINTER	OFFSPRING_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

job
A token for the job whose offspring are desired. A null selector specifies the calling task's job.

list_ptr
A pointer to this structure:

```
DECLARE offspring STRUCTURE (
   max_num                 WORD_16,
   actual                  WORD_16,
   children(*)             SELECTOR);
```

or

```
typedef struct {
   UINT_16                 max_num;
   UINT_16                 actual;
   SELECTOR                children[_NUM_CHILDREN];
                                        /* adjust to max_num */
} OFFSPRING_STRUCT;
```

Where:

`max_num` Specifies the maximum number of slots in this data structure for child job tokens. Before invoking the system call, set this field to greater than 0.

`actual` The actual number of tokens returned in this structure. This number will never be larger than `max_num`. If there are more tokens than slots available, the system call returns only the amount specified by `max_num`.

`children` The returned array of child job tokens.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

By repeated use of this call, you can obtain tokens for all descendants of a job. This information is needed by a task that is attempting to delete a job that has offspring.

See also: **rqe_offspring** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The list_ptr parameter is invalid. Either the selector is invalid or the offset is too small to allow room for the max_num and actual variables.
E_EXIST	0006H	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.
E_PARAM	8004H	The value of max_num is 0 or the list_ptr offset is too small to allow room for max_num, actual, and max_num child job token variables.
E_TYPE	8002H	The job parameter is not a token for a job.

receive

Accepts a message at a port.

Syntax, PL/M and C

```
data_ptr = rq$receive (port_tkn, time_limit, info_ptr,
      except_ptr);
```

```
data_ptr = rq_receive (port_tkn, time_limit, info_ptr,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
data_ptr	POINTER	UINT_8 far *
port_tkn	SELECTOR	SELECTOR
time_limit	WORD_16	UINT_16
info_ptr	POINTER	RECEIVE_INFO_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

data_ptr

A pointer that indicates the starting address of the data portion (if any) of the message after it has been received.

Parameters

port_tkn

A token for the port that is to receive the message.

time_limit

Specifies the maximum time the task will wait for the message to arrive.

Value	Meaning
0	Do not wait.
1-65534	Wait this number of clock intervals.
65535	Wait forever

`info_ptr`

A pointer to this structure:

```
DECLARE receive_info STRUCTURE (
   flags                   WORD_16,
   status                  WORD_16,
   trans_id                WORD_16,
   data_length             WORD_32,
   forwarding_port         SELECTOR,
   remote_socket           WORD_32,
   control_msg(20)         BYTE,
   reserved(4)             BYTE);
```

or

```
typedef struct {
   UINT_16                 flags;
   UINT_16                 status;
   UINT_16                 trans_id;
   UINT_32                 data_length;
   SELECTOR                forwarding_port;
   UINT_32                 remote_socket;
   UINT_8                  control_msg[20];
   UINT_8                  reserved[4];
} RECEIVE_INFO_STRUCT;
```

Where:

`flags` This field has meaning dependent upon certain bit patterns. All others not shown are reserved:

Bits	**Value**	**Meaning**
7-4	0000B	Transactionless message (**send** or similar call).
	0001B	Transmission or system status message.
	0010B	Transaction request message (**send_rsvp** or similar call).
	0100B	Transaction response message (**send_reply** or similar call).
3-0	0000B	The data_ptr parameter points to a single buffer (signal message type).
	0001B	The data_ptr parameter points to a data message buffer.

`status` The send message status. The status codes are:

Value	**Meaning**
0000H	E_OK A new message has been successfully received.
000BH	E_TRANSMISSION A NACK, timeout, bus or host error, or retry expiration occurred during the transmission of the message.
00E1H	E_CANCELLED A **send_rsvp** transaction has been remotely canceled.
00E3H	E_NO_LOCAL_BUFFER If the flags parameter indicates a transaction request message, the local port's buffer pool does not contain a buffer large enough to hold the message so the **receive_fragment** system call is required (message fragmentation). If the flags parameter indicates a transaction response message, the RSVP buffer supplied in the **send_rsvp** system call is not large enough to hold the response.
00E4H	E_NO_REMOTE_BUFFER The remote port's buffer pool does not have a buffer large enough to hold the message and message fragmentation is disabled.

`trans_id` The transaction ID for this message.

Value	**Meaning**
0	A new transactionless message has been received.
not 0	Indicates a transaction request or response message has been received, or a transmission status message has been received.

`data_length`

Indicates the length of the data message received.

If the `flags` parameter indicates a newly received message, the `data_length` parameter contains the length of the successfully received message.

If the `flags` and `status` parameters indicate request message fragmentation, the `data_length` parameter contains the length of all the message fragments that will be received using **receive_fragment**.

forwarding_port

A token identifying a port that is the source port for the port that is actually receiving the message.

remote_socket

A socket (host_ID and port_ID) that indicates the remote message source.

control_msg

The 20 byte control part of a data message.

See also: Control_ptr, **send**

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If the message contains a data portion, a pointer to the buffer used to store the data portion is returned. When the buffer is no longer required, the application should return it to the buffer pool using the **release_buffer** system call. If there is not enough buffer space, the message is rejected by the receiving host.

This system call supports short-circuit transmissions.

See also: Ports, *System Concepts*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The port_tkn parameter does not refer to an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the current configuration.
E_NUC_BAD_BUF	80E2H	The info_ptr parameter points to a buffer that either does not exist, or is not large enough.
E_PROTOCOL	80E0H	The port specified in the port_tkn parameter was created as a signal type.
E_TIME	0001H	Time_limit expired before a message was received.
E_TYPE	8002H	The port_tkn parameter is not a token for a port.

receive_control

Enables the calling task to gain access to a region.

Syntax, PL/M and C

```
CALL rq$receive$control (region, except_ptr);
```

```
rq_receive_control (region, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`region`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`region`

A token for the region to which the calling task wants access.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If no task currently has access, entry is immediate. If another task currently has access, the calling task enters the region's task queue and goes to sleep. The task remains asleep until it can access the data.

If the region has a priority-based task queue, the priority of the task currently having access is temporarily boosted, if necessary, to match that of the task at the head of the queue.

See also: `Region_flags`, CAUTION in **create_region**,
Regions, mutual exclusion, deadlock, *System Concepts*,
create_region example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The region parameter refers to a region already accessed by the calling task.
E_EXIST	0006H	The region parameter is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration. This code is returned if you make this call as an RTE call from Windows instead of from DOS.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_TYPE	8002H	The region parameter is not a token for a region.

receive_data

Receives messages from mailboxes that have been set up to pass data, not tokens. It causes the calling task either to receive the data message or to wait for the data in the task queue of the specified mailbox.

Syntax, PL/M and C

```
actual = rq$receive$data (mailbox, message_ptr, time_limit,
      except_ptr);
```

```
actual = rq_receive_data (mailbox, message_ptr, time_limit,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
actual	WORD_16	UINT_16
mailbox	SELECTOR	SELECTOR
message_ptr	POINTER	void far *
time_limit	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

`actual`

The number of bytes actually received.

Parameters

`mailbox`

A token for the mailbox from which the calling task expects to receive a message.

`message_ptr`

A pointer to a buffer where the message data is placed. The maximum message length is 128 bytes; the buffer must be at least 128 bytes long.

`time_limit`

Specifies how long the task will wait to receive the data message.

Value	Meaning
0	Do not wait.
1-65534	Wait this number of clock intervals.
65535	Wait forever.

See also: For ICU-configurable systems, CIN parameter, *ICU User's Guide and Quick Reference*

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Always use a buffer which is at least 128 bytes long, the maximum message size of the **rq_send_data** call.

If the calling task is not willing to wait, or if the task's waiting period elapses without a data message arriving, the task is awakened with an E_TIME exceptional condition.

When you create a mailbox with **create_mailbox**, you specify whether the mailbox will pass object tokens or data. **Receive_data** functions only with those mailboxes that have been set up to pass data.

See also: **receive_data** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The message_ptr pointer is invalid. Either the selector does not refer to a valid segment, or the offset is outside the segment boundaries.
E_EXIST	0006H	The mailbox parameter is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The call was made as an RTE call from Windows, with a `time_limit` greater than 1. This limitation does not apply to RTE calls made from DOS.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.

E_TIME	0001H	Either the calling task was not willing to wait and there was no message available, or the task waited in the task queue and its designated waiting period elapsed before the message arrived.
E_TYPE	8002H	Either the mailbox parameter is not a token for a mailbox or the mailbox is not a data mailbox.

receive_fragment

Accepts a message fragment of an RSVP data message. Use this call with the **receive** system call to receive a message that is sent from a remote host using a **send_rsvp** system call.

See also: **receive**, **send_rsvp**

Syntax, PL/M and C

```
CALL rq$receive$fragment (port_tkn, socket, rsvp_trans_id,
      fragment_ptr, fragment_length, flags, except_ptr);
```

```
rq_receive_fragment (port_tkn, socket, rsvp_trans_id,
      fragment_ptr, fragment_length, flags, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`port_tkn`	SELECTOR	SELECTOR
`socket`	WORD_32	UINT_32
`rsvp_trans_id`	WORD_16	UINT_16
`fragment_ptr`	POINTER	UINT_8 far *
`fragment_length`	WORD_32	UINT_32
`flags`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`port_tkn`

A token for the port receiving the fragment.

`socket`

A `host_ID` and `port_ID` value specifying the port from which the original RSVP message was sent. If the port issuing **receive_fragment** is connected, this parameter is ignored.

`rsvp_trans_id`

Identifies this particular message transaction. A transaction ID is generated each time **send_rsvp** is invoked.

`fragment_ptr`

A pointer to a buffer into which the message fragment is placed. If this pointer is null, reception of message fragments is terminated.

`fragment_length`

Specifies the length of the fragment. If 0, fragmented transmission of a request message is terminated. This value is obtained from the **receive** system call.

See also: `receive_info` structure, **receive**

`flags`

Specifies the type of message fragment and buffer that `fragment_ptr` points to.

Value	**Meaning**
0	Signal message, single buffer
1	Data message, data chain buffer

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CANCELLED	00E1H	The remote site canceled the transaction.
E_DISCONNECTED	00E9H	The socket parameter is equal to 0 and the port is not connected.
E_EXIST	0006H	The port_tkn parameter does not refer to an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the current configuration.
E_NUC_BAD_BUF	80E2H	The fragment_ptr parameter points to a buffer that either does not exist, or is not large enough.
E_PROTOCOL	80E0H	The port specified in the port_tkn parameter was created as a signal type.
E_TIME	0001H	The configured timeout value expired before the fragment was received See also: For ICU-configurable systems, RFT parameter, *ICU User's Guide and Quick Reference*
E_TRANS_ID	00E8H	The rsvp_trans_id parameter does not specify a currently valid transaction.
E_TYPE	8002H	The port_tkn parameter is not a token for a port.

receive_message

Receives an object token from the specified mailbox. This mailbox must be set up to pass objects (signal message type).

Syntax, PL/M and C

```
object = rq$receive$message (mailbox, time_limit, response_ptr,
      except_ptr);
```

```
object = rq_receive_message (mailbox, time_limit, response_ptr,
      except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`object`	SELECTOR	SELECTOR
`mailbox`	SELECTOR	SELECTOR
`time_limit`	WORD_16	UINT_16
`response_ptr`	POINTER	SELECTOR far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`object`

An object token.

Parameters

`mailbox`

A token for the mailbox at which the calling task expects to receive an object token.

`time_limit`

Specifies the maximum time the task will wait to receive the token.

Value	**Meaning**
0	Do not wait.
1-65534	Wait this number of clock intervals.
65535	Wait forever.

See also: For ICU-configurable systems, CIN parameter, *ICU User's Guide and Quick Reference*

`response_ptr`

If a valid pointer, points to a token for the mailbox to which the receiving task is to send a response. If null, indicates that no response is expected by the sending task.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If the object queue at the mailbox is not empty, the calling task immediately gets the token at the head of the queue and remains ready. Otherwise, the calling task goes into the task queue of the mailbox and goes to sleep, unless the task is not willing to wait. In the latter case, or if the task's waiting period elapses without a token arriving, the task is awakened with an E_TIME condition code.

It is possible that the token returned by **receive_message** is a token for an object that has already been deleted. To verify that the token is valid, the receiving task can invoke the **get_type** system call. However, tasks can avoid this situation by adhering to proper programming practices. One such practice is for the sending task to request a response from the receiving task and not delete the object until it gets a response. When the receiving task finishes with the object, it sends a response, the nature of which must be determined by the writers of the two tasks, to the response mailbox. When the sending task gets this response, it can then delete the original object, if it so desires.

See also: **receive_message** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The mailbox parameter is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The call was made as an RTE call from Windows, with a `time_limit` greater than 1. This limitation does not apply to RTE calls made from DOS.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.

E_TIME	0001H	One of these is true: • The calling task was not willing to wait and there was not a token available. • The task waited in the task queue and its designated waiting period elapsed before the task got the desired token.
E_TYPE	8002H	One of these is true: • The `mailbox` parameter is not a token for a mailbox. • The mailbox was set up to pass data messages, not objects.

receive_reply

Waits for a reply to an RSVP message sent previously by the calling task.

Syntax, PL/M and C

```
data_ptr = rq$receive$reply (port_tkn, rsvp_trans_id,
      time_limit, info_ptr, except_ptr);
```

```
data_ptr = rq_receive_reply (port_tkn, rsvp_trans_id,
      time_limit, info_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
data_ptr	POINTER	UINT_8 far *
port_tkn	SELECTOR	SELECTOR
rsvp_trans_id	WORD_16	UINT_16
time_limit	WORD_16	UINT_16
info_ptr	POINTER	REPLY_INFO_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

data_ptr

A pointer to the data portion (if any) of the message after it has been received. The data portion is always a contiguous block.

Parameters

port_tkn

A token for the port object that is to receive the reply. This port must not be a sink port.

rsvp_trans_id

The transaction ID returned from the associated **send_rsvp** system call.

time_limit

Specifies how long the task is willing to wait for the reply.

Value	Meaning
0	Do not wait.
1-65534	Wait this number of clock intervals.
65535	Wait forever.

`info_ptr`

A pointer to this structure:

```
DECLARE reply_info STRUCTURE (
   flags                  WORD_16,
   status                 WORD_16,
   trans_id               WORD_16,
   data_length            WORD_32,
   forwarding_port        SELECTOR,
   remote_socket          SOCKET,
   control_msg(20)        BYTE,
   reserved(4)            BYTE);
```

or

```
typedef struct {
   UINT_16                flags;
   UINT_16                status;
   UINT_16                trans_id;
   UINT_32                data_length;
   SELECTOR               forwarding_port;
   UINT_32                remote_socket;
   UINT_8                 control_msg[20];
   UINT_8                 reserved[4];
} REPLY_INFO_STRUCT;
```

Where:

`flags` This field has meaning dependent upon certain bit patterns. All others not listed below are reserved:

Bits	Value	Meaning
7-4	0001B	Transmission or system status message.
	0100B	Transaction response message.
3-0	0000B	The data_ptr parameter points to a single buffer (signal message type).
	0001B	The data_ptr parameter points to a data block (data message type).

`status` The send message status. The status codes are:

Value	Meaning
0000H	E_OK A new message has been successfully received.
000BH	E_TRANSMISSION A NACK, timeout, bus or host error, or retry expiration occurred during the transmission of the message.
00E1H	E_CANCELLED A **send_rsvp** transaction has been remotely canceled.
00E3H	E_NO_LOCAL_BUFFER If the flags parameter indicates a transaction response message, the RSVP buffer supplied in the **send_rsvp** system call is not large enough to hold the response.
00E4H	E_NO_REMOTE_BUFFER The remote port's buffer pool was not large enough to hold the message and message fragmentation is disabled.

`trans_id` The transaction ID for this message. If a valid value, the `trans_id` parameter indicates a response message has been received. Otherwise, a transmission status message has been received.

`data_length`
If the `flags` parameter indicates a newly received message, `data_length` contains the length of that message.

`forwarding_port`
This field does not apply to the **receive_reply** system call.

`remote_socket`
A socket for the remote message source.

`control_msg`
The 20-byte control part of a data message.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The port_tkn parameter does not refer to an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the current configuration.
E_NUC_BAD_BUF	80E2H	The info_ptr parameter points to a buffer that is non-existent or too small.
E_PROTOCOL	80E0H	The port specified in the port_tkn parameter was created as a signal type.
E_TIME	0001H	The time the task is willing to wait, specified in the time_limit parameter, expired before a message was received.
E_TRANS_ID	00E8H	Either an invalid transaction ID has been supplied, or the transaction was canceled before the response was received.
E_TYPE	8002H	The port_tkn parameter is not a token for a port.

receive_signal

Receives a signal from the specified port.

Syntax, PL/M and C

```
CALL rq$receive$signal (port_tkn, wait_time, except_ptr);
```

```
rq_receive_signal (port_tkn, wait_time, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`port_tkn`	SELECTOR	SELECTOR
`wait_time`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`port_tkn`

A token for the port where the signal is expected to arrive.

`wait_time`

Specifies how long the task is willing to wait for the signal.

Value	Meaning
0	Do not wait.
1-65534	Wait this number of clock intervals.
65535	Wait forever.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If a signal is already queued at the port, the calling task receives the signal. Otherwise, the task goes to the end of the receive task queue to wait a specified amount of time. If the task is not willing to wait, or if the task's waiting period elapses without a signal arriving, an E_TIME condition code returns.

When a signal arrives and there are tasks waiting in the receive task queue, the task at the head of the queue receives the signal. If the receive queue is empty, the signal is queued at the port. The next task to invoke **receive_signal** receives one of the queued signals.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The port_tkn parameter does not refer to an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PROTOCOL	80E0H	The port specified in the port_tkn parameter was created as a data transport type. It needs to be a signal type.
E_TIME	0001H	One of these is true: • The calling task was not willing to wait and no signal was queued at the port. • The task's designated waiting period elapsed before the desired signal arrived.
E_TYPE	8002H	The port_tkn parameter refers to an object that is not a port.

receive_units

Requests a specified number of units from a semaphore.

Syntax, PL/M and C

```
value = rq$receive$units (semaphore, units, time_limit,
      except_ptr);
```

```
value = rq_receive_units (semaphore, units, time_limit,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
value	WORD_16	UINT_16
semaphore	SELECTOR	SELECTOR
units	WORD_16	UINT_16
time_limit	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

value

The number of units remaining in the semaphore after the calling task's request is satisfied.

Parameters

semaphore

A token for the semaphore from which the calling task wants to receive units.

units

The number of units that the calling task is requesting.

time_limit

Specifies how long the task is willing to wait in the semaphore's task queue.

Value	Meaning
0	Do not wait.
1-65534	Wait this number of clock intervals.
65535	Wait forever.

See also: For ICU-configurable systems, CIN parameter, *ICU User's Guide and Quick Reference*

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If the units are available and the task is at the front of the queue, the task receives the units and remains ready. Otherwise, the task is placed in the semaphore's task queue and goes to sleep. If the task is not willing to wait, or if the task's waiting period elapses before the requested units are available, the task is awakened with an E_TIME condition code.

See also: **receive_units** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The semaphore parameter is not a token for an existing object.
E_LIMIT	0004H	The units parameter is greater than the maximum value specified for the semaphore when it was created.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The call was made as an RTE call from Windows, with a `time_limit` greater than 1. This limitation does not apply to RTE calls made from DOS.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_TIME	0001H	One of these is true: • The calling task was not willing to wait and the requested units were not available. • The time_limit expired while waiting in the task queue before the requested units were available.
E_TYPE	8002H	The semaphore parameter is not a token for a semaphore.

release_buffer

Returns previously allocated buffer space to the specified buffer pool.

Syntax, PL/M and C

```
CALL rq$release$buffer (buffer_pool, buffer_tkn, flags,
      except_ptr);
```

```
rq_release_buffer (buffer_pool, buffer_tkn, flags, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
buffer_pool	SELECTOR	SELECTOR
buffer_tkn	SELECTOR	SELECTOR
flags	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

buffer_pool

A token for the buffer pool that is to receive the released buffer.

buffer_tkn

A token for the buffer to be released.

flags Indicates:

Bits	Value	Meaning
15-2	0	Reserved, set to 0.
1	0	Return the buffer to the Free Space Manager.
	1	Do not return the buffer. This is for the case where the number of buffers has reached the maximum (from **create_buffer_pool**).
0	0	The buffer_tkn parameter refers to a contiguous buffer.
	1	The buffer_tkn parameter refers to a data chain (iRMX III OS only).

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If the `flags` bit 1 is set and you try to release a buffer to a full pool, the call will delete the segment and return E_OK. If the `flags` bit 1 is 0 and you try to release a buffer to a full pool, the call will not delete the segment and will return E_LIMIT.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	Either or both of the buffer_tkn and buffer_pool parameters do not refer to an existing object.
E_LIMIT	0004H	The calling task's job has already reached its object limit.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	Either buffer_pool does not refer to a buffer pool, or buffer_tkn does not refer to a segment.

request_buffer

Gets a buffer from a buffer pool created by **create_buffer_pool**. This call does not create a segment if none are available in the pool.

Syntax, PL/M and C

```
buffer_token = rq$request$buffer (buffer_pool, size,
      except_ptr);
```

```
buffer_token = rq_request_buffer (buffer_pool, size,
      except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
buffer_token	SELECTOR	SELECTOR
buffer_pool	SELECTOR	SELECTOR
size	WORD_32	UINT_32
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

buffer_token

A token identifying the buffer that fills the request. This buffer is either a single segment or a data chain block, as specified when the buffer pool was created.

See also: **create_buffer_pool**

Parameters

buffer_pool

A token for an existing buffer pool.

size Specifies the desired size of the requested buffer in bytes. This value must be in the range of 1 through 0FFFFFFFEH.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_DATA_CHAIN	000DH	A data chain has been returned. The token points to the beginning of the data chain block.
E_EXIST	0006H	The buffer_pool parameter does not refer to an existing object.
E_LIMIT	0004H	The size parameter requests a buffer size large enough to require a data chain whose number of elements exceeds the configured value for the maximum data chain elements.
E_MEM	0002H	The system could not locate enough memory to return the requested buffer from the buffer pool, either as a segment or a data chain. This error is returned if no segments are currently available in the buffer pool.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The size parameter is equal to 0, or is larger than 0FFFFFFFEH.
E_SLOT	000CH	The GDT is full.
E_TYPE	8002H	The buffer_pool parameter refers to an object that is not a buffer pool.

reset_interrupt

Cancels the assignment of the current interrupt handler to the specified interrupt level, and disables the level.

Syntax, PL/M and C

```
CALL rq$reset$interrupt (level, except_ptr);
```

```
rq_reset_interrupt (level, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`level`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`level` Specifies the interrupt level:

Bits	Value	Meaning
15-7	0	Reserved, set to 0
6-4	0-7	First digit of the interrupt level
3	0	Bits 2-0 specify the second digit (slave)
	1	Bits 6-4 specify the entire level number (master)
2-0	0-7	Second digit of the interrupt level

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If an interrupt task has also been assigned to the level, the interrupt task is deleted.

The level reserved for the system clock should not be reset and is considered invalid for this call.

See also: For ICU-configurable systems, CIN parameter, *ICU User's Guide and Quick Reference*,
:rmx:demo/c/interrupt directory for demo using **rq_reset_interrupt**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	There is no interrupt handler assigned to the specified level.
E_LIMIT	0004H	The task priority associated with the specified interrupt level exceeds the job's maximum priority.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The level parameter is invalid.

resume_task

Decreases by one the suspension depth of the specified non-interrupt task.

Syntax, PL/M and C

```
CALL rq$resume$task (task, except_ptr);
```

```
rq_resume_task (task, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`task`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`task` A token for the task whose suspension depth is to be decreased.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If the specified task is suspended or asleep-suspended, its suspension depth should be at least 1. If the suspension depth is still positive after the **resume_task** call, the task state remains unchanged. If the suspension depth goes to 0, the task is placed in the ready state (if suspended) or the asleep state (if asleep-suspended).

See also: **create_task** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The task indicated by the task parameter is an interrupt task.
E_EXIST	0006H	The task parameter is not a token for an existing object.
E_STATE	0007H	The task indicated by the task parameter was not suspended when the call was made.
E_TYPE	8002H	The task parameter is not a token for a task.

send

Sends a data message from a port to a port on another host.

Syntax, PL/M and C

```
trans_id = rq$send (port_tkn, socket, control_ptr, data_ptr,
      data_length, flags, except_ptr);
```

```
trans_id = rq_send (port_tkn, socket, control_ptr, data_ptr,
      data_length, flags, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
trans_id	WORD_16	UINT_16
port_tkn	SELECTOR	SELECTOR
socket	WORD_32	UINT_32
control_ptr	POINTER	UINT_8 far *
data_ptr	POINTER	UINT_8 far *
data_length	WORD_32	UINT_32
flags	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

`trans_id`

Identifies this particular message transmission. If no data is being sent (`data_ptr` is null), the value returned is 0.

Parameters

`port_tkn`

A token for the port to which a message is to be sent.

`socket`

Specifies a unique `host_ID:port_ID` combination that identifies the message destination. If the sending port has been connected using a **connect** system call it has a default socket and this parameter is ignored.

`control_ptr`

A pointer to the control portion of a message. If the `data_ptr` parameter is null or the `data_length` parameter is 0, the control message is 20 bytes long. Otherwise, the control message is 16 bytes.

data_ptr

A pointer to a data message.

Value	Meaning
Null pointer	There is no optional data portion for this message; send a control message.
Valid pointer	Points to either a contiguous buffer or a data chain, depending on the flags parameter.

data_length

Specifies the length of the data message.

flags A bit pattern encoded as:

Bits	Value	Meaning
15-8	0	Reserved, set to 0.
7-4	0000B	Transmission is synchronous.
	0001B	Transaction is asynchronous.
3-0	0000B	The data_ptr parameter points to a contiguous buffer.
	0001B	The data_ptr parameter points to a data chain (iRMX III OS only).

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If the remote port to which the message is sent does not have adequate buffer space to receive the message an E_NO_REMOTE_BUFFER condition code will be returned. This call does not support fragmentation.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_DISCONNECTED	00E9H	The socket parameter is 0 and the port is not connected.
E_EXIST	0006H	The port_tkn parameter does not point to an existing object.
E_HOST_ID	00E2H	The host_id portion of the socket parameter does not refer to a board that is currently in the message space. This error is not produced for host_id values in the range of 21 to 255.
E_NO_REMOTE_BUFFER	00E4H	The receiving host could not allocate a buffer to hold the message.

E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NUC_BAD_BUF	80E2H	Either the control_ptr or data_ptr parameter is invalid or points to a buffer that is not large enough.
E_PROTOCOL	80E0H	The port specified in the port_tkn parameter was created as a signal type. It needs to be a data type.
E_RESOURCE_LIMIT	00E6H	The configured number of simultaneous messages has been reached. See also: For ICU-configurable systems, MSM parameter, *ICU User's Guide and Quick Reference*
E_TRANS_LIMIT	00EAH	A transmission resource limitation has been encountered. An insufficient number of transaction buffers was specified during system configuration. See also: For ICU-configurable systems, MST parameter, *ICU User's Guide and Quick Reference*
E_TRANSMISSION	000BH	A NACK, timeout, bus or host error, or retry expiration occurred during the transmission of the message.
E_TYPE	8002H	The port_tkn parameter refers to an object that is not a port.

send_control

Releases the calling task's control of a region. Tasks cannot be deleted while they have control of the region.

Syntax, PL/M and C

```
CALL rq$send$control (except_ptr);

rq_send_control (except_ptr);
```

Parameter	PL/M Data Type	C Data Type
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameter

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If the task is in control of more than one region, **send_control** releases control of the most recently accessed region. Once control is released, the OS enables the next task in line to gain access.

If the calling task has had its priority boosted through access to a region, its priority is restored only when it gives up control of the last region. It is not sufficient to give up control of the region that raised the priority, if the task controls other regions.

See also: **create_region**, **accept_control**, **receive_control**,
Regions, *System Concepts*, **create_region** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The calling task does not have control of a region.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration. This code is returned if you make this call as an RTE call from Windows instead of from DOS.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.

send_data

Sends messages up to 128 bytes in length to mailboxes that have been set up to pass data.

Syntax, PL/M and C

```
CALL rq$send$data (mailbox, message_ptr, actual_length,
      except_ptr);
```

```
rq_send_data (mailbox, message_ptr, actual_length, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`mailbox`	SELECTOR	SELECTOR
`message_ptr`	POINTER	void far *
`actual_length`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`mailbox`

A token for the data mailbox to which the message is to be sent.

See also: **create_mailbox**

`message_ptr`

A pointer to a buffer containing the message.

`actual_length`

Specifies the length of the message between 0 and 0FFFFH. Messages are limited to 128 bytes, so any value over 128 causes only 128 bytes to be sent.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The buffer which receives the message must be at least 128 bytes long.

If there are tasks in the task queue at the mailbox, the task at the head of the queue is awakened and is given the data. Otherwise, the message data is placed at the tail of the mailbox's message queue.

See also: **receive_data** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The pointer to the message is invalid. Either the selector does not refer to a valid segment, or the offset is outside the segment boundaries. This code is not returned when using the DOS RTE.
E_EXIST	0006H	The mailbox token is not a token for an existing object.
E_MEM	0002H	The data message queue is full and the system does not have enough memory to create another.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_TYPE	8002H	Either one of these is true: • The `mailbox` parameter is not a token for a mailbox. • The specified mailbox was set up to pass tokens, not data.

send_message

Sends a message to a mailbox that has been set up to pass objects.

Syntax, PL/M and C

```
CALL rq$send$message (mailbox, object, response, except_ptr);
```

```
rq_send_message (mailbox, object, response, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`mailbox`	SELECTOR	SELECTOR
`object`	SELECTOR	SELECTOR
`response`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`mailbox`

A token for the mailbox to which an object token is to be sent.

See also: **create_mailbox**

`object`

A token identifying the object which is to be sent.

`response`

The token for the mailbox or semaphore at which the sending task waits for a response. A null selector indicates that no response is requested.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If there are tasks in the task queue at that mailbox, the task at the head of the queue is awakened and is given the token. Otherwise, the token is placed at the tail of the object queue of the mailbox.

The sending task has the option of specifying a mailbox or semaphore at which to wait for a response from the receiving task. The receiving task must be aware of whether the response token is for a semaphore or mailbox.

See also: **receive_message** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	One or more of the parameters is not a token for an existing object.
E_MEM	0002H	The high performance queue is full and the calling task's job does not contain sufficient memory to complete the call.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_TYPE	8002H	At least one of these is true: • The `mailbox` parameter is not a token for a mailbox. • The `response` parameter is a token for an object that is neither a mailbox nor a semaphore. • The specified mailbox was set up to pass data, not tokens.

send_reply

Sends responses to the **send_rsvp** system call. The reply message may be sent as a single message or as a series of message fragments.

Syntax, PL/M and C

```
trans_id = rq$send$reply (port_tkn, socket, rsvp_trans_id,
      control_ptr, data_ptr, data_length, flags, except_ptr);

trans_id = rq_send_reply (port_tkn, socket, rsvp_trans_id,
      control_ptr, data_ptr, data_length, flags, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
trans_id	WORD_16	UINT_16
port_tkn	SELECTOR	SELECTOR
socket	WORD_32	UINT_32
rsvp_trans_id	WORD_16	UINT_16
control_ptr	POINTER	UINT_8 far *
data_ptr	POINTER	UINT_8 far *
data_length	WORD_32	UINT_32
flags	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

`trans_id`

Identifies this particular message transmission. If no data is being sent (the `data_ptr` parameter is null), the value returned is 0.

Parameters

`port_tkn`

The token identifying the port from which the request is sent.

`socket`

Identifies the remote destination. If the sending port has a default socket, this parameter is ignored.

`rsvp_trans_id`

This is the `trans_id` parameter from the **send_rsvp** call that is being answered. This is used at the destination to identify the transaction.

`control_ptr`

A pointer to the control portion of the message. If the `data_ptr` parameter is null or the `data_length` parameter is zero, the control message is 20 bytes long. Otherwise, the control message is 16 bytes.

`data_ptr`

A pointer to a data message.

Value	**Meaning**
Null pointer	There is no optional data portion for this message; send a control message.
Valid pointer	Points to either a contiguous buffer or a data chain, depending on the flags parameter.

`data_length`

Specifies the length of the data message.

`flags` A bit pattern indicates:

Bits	**Value**	**Meaning**
15-10	0	Reserved, set to 0.
9	0	This message is the last fragment of a response (EOT flag).
	1	Send more fragments.
8	0	Reserved, set to 0.
7-4	0000B	Transmission is synchronous.
	0001B	Transaction is asynchronous.
3-0	0000B	The data_ptr parameter points to a contiguous buffer.
	0001B	The data_ptr parameter points to a data chain (iRMX III OS only).

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_DISCONNECTED	00E9H	The board that initiated the **send_rsvp** has been reset.
E_EXIST	0006H	The port_tkn parameter does not point to an existing object.
E_HOST_ID	00E2H	The host_id portion of the socket parameter does not refer to a host that is currently in message space. This error is not produced for host_id values in the range of 21 to 255.

E_NO_REMOTE_BUFFER	00E4H	The receiving host could not allocate a message buffer.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NUC_BAD_BUF	80E2H	Either the control_ptr or data_ptr parameter is invalid or points to a buffer that is not large enough.
E_PROTOCOL	80E0H	The port specified in port_tkn is not a data transport type.
E_RESOURCE_LIMIT	00E6H	The configured number of simultaneous messages or simultaneous transactions has been reached. See also: For ICU-configurable systems, MSM/MST parameters, *ICU User's Guide and Quick Reference*
E_TRANSMISSION	000BH	A NACK, timeout, bus or host error, or retry expiration occurred during the transmission of the message.
E_TRANS_LIMIT	00EAH	A transmission resource limitation has been encountered. See also: For ICU-configurable systems, MST parameter, *ICU User's Guide and Quick Reference*
E_TYPE	8002H	The port_tkn parameter refers to an object that is not a port.

send_rsvp

Initiates a request message transaction that has an implied response (RSVP message transmission).

Syntax, PL/M and C

```
trans_id = rq$send$rsvp (port_tkn, socket, control_ptr,
      data_ptr, data_length, rsvp_data_ptr, rsvp_data_length,
      flags, except_ptr);
```

```
trans_id = rq_send_rsvp (port_tkn, socket, control_ptr,
      data_ptr, data_length, rsvp_data_ptr, rsvp_data_length,
      flags, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`trans_id`	WORD_16	UINT_16
`port_tkn`	SELECTOR	SELECTOR
`socket`	WORD_32	UINT_32
`control_ptr`	POINTER	UINT_8 far *
`data_ptr`	POINTER	UINT_8 far *
`data_length`	WORD_32	UINT_32
`rsvp_data_ptr`	POINTER	UINT_8 far *
`rsvp_data_length`	WORD_32	UINT_32
`flags`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`trans_id`

Identifies this particular message transmission. If no data is being sent (`data_ptr` is null), 0 returns.

Parameters

`port_tkn`

The token identifying the port that sends the RSVP message.

`socket`

Identifies the `host_id` and `port_id`. If the sending port is connected, it has a default socket and this parameter is ignored.

control_ptr

A pointer to a control message. If the data_ptr parameter is null or the data_length parameter is 0, this message is 20 bytes long. Otherwise, it is 16 bytes.

data_ptr

A pointer to a data message.

Value	Meaning
Null pointer	There is no optional data portion for this message; send a control message
Valid pointer	Points to either a contiguous buffer or a data chain, depending on the flags parameter

data_length

Specifies the length of the data message.

rsvp_data_ptr

A pointer to a buffer into which the RSVP message is to be placed. This buffer must be a contiguous block.

rsvp_data_length

Defines the length of the RSVP message buffer.

flags A bit pattern encoded as:

Bits	Value	Meaning
15-9	0	Reserved, set to 0.
8	0	Use **receive_reply** system call for RSVP.
	1	Use **receive** system call for RSVP.
7-4	0000B	Transmission is synchronous.
	0001B	Transaction is asynchronous.
3-0	0000B	The data_ptr parameter points to a contiguous buffer.
	0001B	The data_ptr parameter points to a data chain (iRMX III OS only).

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Typically you use RSVP message transactions to transfer data from one host to another.

When the message cannot be delivered no action is required by the receiver and the receiving task/port is not notified of the transaction.

Note
If the specified port was created with message fragmentation enabled and the RSVP message requires fragmentation, the application at the destination must be able to handle the message fragments. If not, **send_rsvp** will sleep indefinitely.

See also: **create_port**, **receive**, **receive_fragment**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CANCELLED	00E1H	At least one of these is true: • The remote host canceled the transaction. • Receiver's port is non-existent or is being deleted. • Receiver's port has been connected to a socket other than the sender's (with **rq_connect**). • Receiver has insufficient transaction buffers (set at configuration time) or port transactions (set at **rq_create_port** time). • Receiver has terminated the fragmented request transaction via **rq_receive_fragment.**
E_DISCONNECTED	00E9H	0 was specified for the socket parameter with the port having no default socket.
E_EXIST	0006H	One of the port_tkn, control_ptr, or data_ptr parameters does not point to an existing object.
E_HOST_ID	00E2H	The host_id portion of the socket parameter does not refer to an existing host. This error is not produced for host_id values in the range of 21 to 255.
E_NO_REMOTE_BUFFER	00E4H	The receiving host could not allocate a message buffer.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_NUC_BAD_BUF	80E2H	The data_ptr parameter is invalid or points to a buffer that is not large enough, or the control_ptr parameter is null.

E_PROTOCOL	80E0H	The port specified in the port_tkn parameter is not a data transport type.
E_RESOURCE_LIMIT	00E6H	The configured number of simultaneous messages or simultaneous transactions has been reached. See also: For ICU-configurable systems, MSM, MST parameters, *ICU User's Guide and Quick Reference*
E_TRANSMISSION	000BH	A NACK, timeout, bus or host error, or retry expiration occurred during the transmission of the message.
E_TRANS_LIMIT	00EAH	A transmission resource limitation has been encountered. An insufficient number of transaction buffers was specified during system configuration. See also: For ICU-configurable systems, MST parameter, *ICU User's Guide and Quick Reference*
E_TYPE	8002H	The port_tkn parameter refers to an object that is not a port.

send_signal

Sends a signal message to a remote host through the specified port.

Syntax, PL/M and C

```
CALL rq$send$signal (port_tkn, except_ptr);
```

```
rq_send_signal (port_tkn, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`port_tkn`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`port_tkn`

A token for the port through which the signal is sent.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If a bus timeout or other bus error occurs, the calling task receives an E_TRANSMISSION condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The port_tkn parameter is not a token for an existing object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PROTOCOL	80E0H	The port specified in the port_tkn parameter is not a signal type.
E_TRANSMISSION	000BH	A NACK, timeout, bus or host error, or retry expiration occurred during the transmission of the signal.
E_TYPE	8002H	The port_tkn parameter is a token for an object that is not a port.

send_units

Sends the specified number of units to the specified semaphore.

Syntax, PL/M and C

```
CALL rq$send$units (semaphore, units, except_ptr);
```

```
rq_send_units (semaphore, units, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`semaphore`	SELECTOR	SELECTOR
`units`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`semaphore`

A token for the semaphore to which the units are to be sent.

`units`

The number of units to be sent.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If the transmission would cause the semaphore to exceed its maximum allowable supply, an E_LIMIT condition code occurs. Otherwise, the transmission is successful and the Nucleus attempts to satisfy the requests of the tasks in the semaphore's task queue, beginning at the head of the queue.

See also: **create_semaphore**, **receive_units** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The semaphore parameter is not a token for an existing object.
E_LIMIT	0004H	The number of units that the calling task is trying to send would cause the semaphore to exceed its maximum allowable supply of units.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_TYPE	8002H	The semaphore parameter is not a token for a semaphore.

set_exception_handler

Assigns an exception handler and exception mode attributes to the calling task.

See also: **rqe_create_job** to set the exception handler for the job
rqe_set_exception_handler to set any of the task, job, or system exception handlers

Syntax, PL/M and C

```
CALL rq$set$exception$handler (exception_info_ptr, except_ptr);
```

```
rq_set_exception_handler (exception_info_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
exception_info_ptr	POINTER	EXCEPTION_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

exception_info_ptr

For PL/M, a pointer to this structure:

```
DECLARE exception STRUCTURE (
   exception_handler_ptr  POINTER,
   exception_mode         BYTE);
```

or for C segmented compilers:

```
typedef struct {
   void far *             exception_handler_ptr;
   UINT_8                 exception_mode;
} EXCEPTION_STRUCT;
```

For C flat model compilers only, a pointer to this structure:

```
typedef struct {
   void *                 exception_handler_ptr;
   SELECTOR               exception_handler_ptr_seg;
   UINT_8                 exception_mode;
} EXCEPTION_STRUCT;
```

Where:

exception_handler_ptr

Points to the first instruction of the exception handler. If null, the exception handler of the calling task's parent job is assigned.

`exception_handler_ptr_seg`

For flat model compilers only, the selector for the pointer.

`exception_mode`

Indicates:

Value	When Control Passes To Exception Handler
0	Never
1	On programmer errors only
2	On environmental conditions only
3	On all exceptional conditions

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	Either the `exception_info_ptr` or `exception_handler_ptr` is invalid, or the offset part of one of the pointers is outside the segment boundaries.
E_NOT_ALLOCATED	00F2H	The base part of the `exception_info_ptr` or `exception_handler_ptr` parameter is a descriptor or virtual segment, and the offset part does not point to an area of the segment that contains physical memory.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The `exception_mode` field is greater than 3.

rqe_set_exception_handler

Assigns an exception handler and exception mode or changes the current mode for any of the following:

- Current task exception handler
- Current job exception handler
- System-wide exception handler

Syntax, PL/M and C

```
CALL rqe$set$exception$handler (info_ptr, except_ptr);
```

```
rq_set_exception_handler (info_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
info_ptr	POINTER	EXCEPTION_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

info_ptr

For PL/M, a pointer to this structure where you specify the exception handler and mode:

```
DECLARE exception STRUCTURE (
   exception_handler_ptr  POINTER,
   exception_mode         BYTE);
```

or for C segmented compilers:

```
typedef struct {
   void far *             exception_handler_ptr;
   UINT_8                 exception_mode;
} EXCEPTION_STRUCT;
```

For C flat model compilers only, a pointer to this structure:

```
typedef struct {
   void *                 exception_handler_ptr;
   SELECTOR               exception_handler_ptr_seg;
   UINT_8                 exception_mode;
} EXCEPTION_STRUCT;
```

Where:

`exception_handler_ptr`

Either points to the first instruction of your exception handler or a null pointer to use an already established handler; see the table below.

`exception_handler_ptr_seg`

For flat model compilers only, the selector for the pointer.

`exception_mode`

Indicates the exception-handling mode according to the table below.

Task Settings		
Mode	**Valid Pointer**	**Null Pointer**
0 Task will handle all exceptions in-line except hardware exceptions, which are returned to the last valid exception handler	ignored	ignored
1 Task's exception handler receives programmer errors and hardware exceptions only	Your exception handler assigned to this task	Job's current exception handler assigned to this task
2 Task's exception handler receives environmental errors and hardware exceptions only		
3 Task's exception handler receives all exceptions		
Job Settings		
Mode	**Valid Pointer**	**Null Pointer**
4 Job default exception mode returns exceptions to the offending task to handle in-line; hardware exceptions are returned to the last valid exception handler.	ignored	ignored
5 Job's exception handler receives programmer errors and hardware exceptions only	Your exception handler assigned as default for this job	Returns E_CONTEXT exception
6 Job's exception handler receives environmental errors and hardware exceptions only		
7 Job's exception handler receives all exceptions		

System Settings		
Mode	**Valid Pointer**	**Null Pointer**
8 System default exception mode returns exceptions to the offending task to handle in-line; hardware exceptions are returned to the last valid exception handler	ignored	ignored
9 System exception handler receives programmer errors and hardware exceptions only	Your exception handler assigned as default for the system	Returns E_CONTEXT exception
10 System exception handler receives environmental errors and hardware exceptions only		
11 System exception handler receives all exceptions		
System Hardware Trap Settings		
Mode	**Valid Pointer**	**Null Pointer**
12 Change system-wide hardware trap handlers to Delete_Offending_Job (same effect as setting DEH=0FFH in *rmx.ini* file)	ignored	ignored
13 Change system-wide hardware trap handlers to Delete_Offending_Task		
14 Change system-wide hardware trap handlers to Suspend_Offending_Task		
15 Change system-wide hardware trap handlers to Break_to_Monitor (same effect as setting DEH=00H in *rmx.ini* file)		

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The system-wide exception handler refers to the root job's exception handler. When you change the system-wide exception handler, it changes only the default exception handler that is inherited by first-level jobs created by the root job.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	Either the `info_ptr` or `exception_handler_ptr` is invalid, or the offset part of one of the pointers is outside the segment boundaries.
E_NOT_ALLOCATED	00F2H	The base (segment) part of the `info_ptr` or `exception_handler_ptr` parameter is a descriptor or virtual segment, and the offset part does not point to an area of the segment that contains physical memory.

set_interconnect

Changes the contents of a Multibus II interconnect register to a specified value.

⚠ **CAUTION**
It is possible to corrupt the operation of the board or system by specifying incorrect values in interconnect registers.

Syntax, PL/M and C

```
CALL rq$set$interconnect (value, slot_number, reg_number,
      except_ptr);
```

```
rq_set_interconnect (value, slot_number, reg_number,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
value	BYTE	UINT_8
slot_number	BYTE	UINT_8
reg_number	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

value

The value to which the specified interconnect register is to be changed.

slot_number

The Multibus II cardslot number of the board on which the specified interconnect register is located:

Value	Meaning
0-19	PSB slot numbers 0 to 19
20-23	Reserved, do not specify these values
24-29	iLBX II cardslot numbers 0 to 5
30	Reserved
31	Program the contents of a local interconnect register (on the board where the calling task is running)

reg_number

The interconnect register to which a value is to be written. This value must be in the range 0000H to 01FFH. Refer to the Multibus II board's hardware reference manual for an exact definition of its interconnect space.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The Nucleus checks the range validity of the cardslot and register numbers specified in the call. It does not verify the existence of a board in the specified cardslot nor does it check the read/write permission of the register before it attempts to access the register.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	One or more of the parameters has an illegal value.

set_interrupt

Assigns an interrupt handler to an interrupt level and, optionally, makes the calling task the interrupt task for that level.

Syntax, PL/M and C

```
CALL rq$set$interrupt (level, interrupt_task_flag,
      interrupt_handler, interrupt_handler_ds, except_ptr);

rq_set_interrupt (level, interrupt_task_flag,
      interrupt_handler, interrupt_handler_ds, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`level`	WORD_16	UINT_16
`interrupt_task_flag`	BYTE	UINT_8
`interrupt_handler`	POINTER	void (far *)(void)
`interrupt_handler_ds`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`level`

Specifies the interrupt level:

Bits	Value	Meaning
15-7	0	Reserved, set to 0
6-4	0-7	First digit of the interrupt level (master PIC level)
3	0	Bits 2-0 specify the second digit (slave)
	1	Bits 6-4 specify the entire level number (master)
2-0	0-7	Second digit of the interrupt level (slave PIC level)

See also: Interrupts, *System Concepts*, *Programming Techniques*, and *Programming Concepts for DOS and Windows* `interrupt_task_flag`

Specifies:

Value	Meaning
0	No interrupt task is to be associated with this level (the new interrupt handler will not call **signal_interrupt**).
not 0	The number of outstanding **signal_interrupt** requests that can exist; when this limit is reached, the associated interrupt level is disabled. The maximum value is 255 decimal. Also, indicates that the calling task becomes the interrupt task.

⚠ **CAUTION**

Do not set `interrupt_task_flag` to 0 if the designated interrupt handler is part of an HI application. If the application is stopped using a <Ctrl-C> entered at the keyboard, subsequent interrupts could cause unpredictable results.

See also: Interrupts, *System Concepts*

`interrupt_handler`

A pointer to the first instruction of the interrupt handler.

`interrupt_handler_ds`

A token identifying the interrupt handler's data segment.

Value	Meaning
Null selector	The interrupt handler loads its own data segment and may not invoke **enter_interrupt**.
Valid selector	The base of the interrupt handler's data segment.

See also: **enter_interrupt**

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The number of outstanding **signal_interrupt** requests that the handler can make before the associated interrupt level is disabled generally corresponds to the number of buffers used by the handler and interrupt task.

If there is an interrupt task, the calling task is that interrupt task. If there is no interrupt task, **set_interrupt** also enables the specified level, which must be disabled at the time of the call.

You may want an interrupt handler to pass information to the interrupt task that it calls. These PL/M statements, when included in the interrupt task's code (with the first statement listed here being the first statement in the task's code), will extract the DS register value used by the interrupt task and make it available to the interrupt handler, which in turn can access it by calling **enter_interrupt**:

```
DECLARE begin        WORD_16;   /* A DUMMY VARIABLE */
CALL rq$set$interrupt (...,SELECTOR$OF(@begin),...);
```

See also: Interrupts, *System Concepts*
:rmx:demo/c/int directory for demos using **rq_signal_interrupt**, **rq_reset_interrupt**, **rqe_timed_interrupt**, and **rq_set_interrupt**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The pointer to the interrupt handler or the selector for the data segment is invalid. Either one of the selectors does not refer to a valid segment, or the offset is outside the segment boundaries.
E_CONTEXT	0005H	One of these is true: • The task is already an interrupt task. • The specified level already has an interrupt handler assigned to it. • The job containing the calling task or the calling task itself is being deleted.
E_LIMIT	0004H	The priority parameter is not 0 and greater (numerically smaller) than the maximum allowable priority for tasks in the calling task's job.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration
E_PARAM	8004H	One of these is true: • The `level` parameter is invalid or would cause the task to have a priority not allowed by its job. • The PIC for the specified level is not part of the hardware configuration.

rqe_set_max_priority

Dynamically changes the maximum priority of tasks in a job.

⚠ **CAUTION**
Enables tasks of priority greater than tasks in system jobs to be created from a user/application job. Thus, some system tasks and real-time performance of the iRMX OS can be degraded.

Syntax, PL/M and C

```
CALL rqe$set$max$priority (job_token, max_priority,
      except_ptr);

rqe_set_max_priority (job_token, max_priority, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`job_token`	SELECTOR	SELECTOR
`max_priority`	BYTE	UINT_8
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`job_token`

A token for the job whose `max_priority` parameter is to be changed. A null selector specifies the calling task's job.

`max_priority`

The job's new maximum priority.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

`Max_priority` must not be lower (numerically greater) than the current value of the job's maximum priority.

This call is typically used by HI applications which include interrupt tasks.

See also: **create_task** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The specified job_token parameter is not a valid job token.
E_EXIST	0006H	The job_token parameter is not a token for an existing object.
E_LIMIT	0004H	The max_priority parameter contains a priority value that is lower (numerically greater) than the max_priority of the specified job.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

rqe_set_os_extension

Dynamically associates an entry point of a user-written OS extension with a call gate. It can also clear that association.

Syntax, PL/M and C

```
CALL rqe$set$os$extension (gate_number, start_address,
      except_ptr);
```

```
rqe_set_os_extension (gate_number, start_address, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`gate_number`	WORD_16	UINT_16
`start_address`	POINTER	void (far *)(void)
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`gate_number`

Specifies the entry number in the GDT of the call gate to be associated with the OS extension. The call gate must have been reserved for this purpose during system configuration.

See also: GSN parameter, *ICU User's Guide and Quick Reference*,
for iRMX for PCs and iRMX for Windows, see OSN in *System Configuration and Administration*

`start_address`

A pointer to the first instruction of the OS extension. A null value disables the OS extension previously associated with the call gate.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

If you use the same call gate for multiple OS extensions, you must use **rqe_set_os_extension** to terminate the association before establishing an association with another. If a task attempts to invoke an OS extension that has been disabled in this manner, a null operation occurs.

See also: OS extensions, *System Concepts*

To allow multiple calls to **rqe_set_os_extension** for the same gate during debug operations, be sure to first make the call with a null `start_address` parameter followed by a call with the correct `start_address` parameter. Otherwise an E_CONTEXT exception will occur.

⚠ **CAUTION**
When writing OS extensions, always reset the OS extension with a null value in the `start_address`. Then, issue the call again with the desired `start_address`. Otherwise, the system will not initialize on a warm reset. In this case, an E_CONTEXT (0005H) initialization error will occur.

See also: **rqe_set_os_extension** example, Nucleus examples

A flat model application can install itself as an OS extension. However, since these applications run in the ring three protection level, only other ring three applications will be able to access the extension. If you want to create a general-purpose OS extension, use a segmented memory model that runs in the ring zero protection level.

When writing a flat model OS extension, remember that the extension exit code must make a FAR return back through the call gate to the caller. This cannot be done from a high level language in flat model, it must be done in assembly code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The pointer to the start address is invalid. Either the selector doesn't refer to a valid segment, or the offset is outside the segment boundaries.
E_CONTEXT	0005H	The specified call gate is already associated with an OS extension.
E_EXIST	0006H	The `gate_number` parameter specifies an uninitialized GDT slot.
E_NOT_ALLOCATED	00F2H	The base part of the `start_address` parameter is a descriptor or virtual segment, and the offset part does not point to an area of the segment that contains physical memory.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	The `gate_number` parameter specifies an initialized GDT slot which is not a call gate.

set_pool_min

Sets the pool_min parameter of the calling task's job. The new value must not exceed that job's pool_max parameter.

Syntax, PL/M and C

```
CALL rq$set$pool$min (new_min, except_ptr);

rq_set_pool_min (new_min, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
new_min	WORD_32	NATIVE_WORD
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

new_min

Specifies the new pool_min parameter of the calling task's job.

Value	Meaning
0FFFFH	Set the pool_min parameter equal to the pool_max parameter.
Not 0FFFFH	The new value of the pool_min parameter.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

When the pool_min parameter is made larger than the current pool size, the pool is not enlarged until the additional memory is needed. An iRMX job can have a memory pool of up to 4 Gbytes in length.

See also: **rqe_create_job**, **set_pool_min** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_LIMIT	0004H	The new_min parameter is not 0FFFFH, but it is greater than the pool_max parameter of the calling task's job.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

set_priority

Dynamically changes the priority of a non-interrupt task. The new value must not exceed the containing job's maximum priority.

⚠ **CAUTION**
Tasks can be put to sleep for long periods of time, and real-time performance of the iRMX OS is degraded when a task uses this system call to lower its own priority.

See also: **create_task** example, Nucleus examples

Syntax, PL/M and C

```
CALL rq$set$priority (task, priority, except_ptr);
```

```
rq_set_priority (task, priority, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
task	SELECTOR	SELECTOR
priority	BYTE	UINT_8
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

`task`

A token for the task whose priority is to be changed. A null selector specifies the calling task.

`priority`

The task's new priority. The value 0 specifies the maximum priority of the specified task's containing job.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The specified task is an interrupt task. You cannot set the priority of an interrupt task dynamically.
E_EXIST	0006H	The task parameter is not a token for an existing object.
E_LIMIT	0004H	The priority parameter contains a priority value that is higher (numerically less) than the maximum priority of the specified task's containing job.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	The task parameter is not a token for a task.

set_time

Sets the date and time for the BIOS's local clock.

Syntax, PL/M and C

```
CALL rq$set$time (date_time, except_ptr);
```

```
rq_set_time (date_time, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
date_time	WORD_32	UINT_32
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

date_time

Contains a date/time value expressed as the number of seconds since a fixed, user-determined point in time.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Any time in the past can be used as the beginning of time; the iRMX OS uses 2:00 AM, January 1, 1978 as the default; PC Systems running DOS use 2:00 AM, January 1, 1980. The iRMX OS convention is used by the UDI and the HI, so it is recommended. When the date_time value reaches its maximum of 0FFFFFFFFH, it will stop incrementing and will not roll over to start again from 0.

See also: UDI call **dq_decode_time**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

signal_exception

Used with OS extensions to signal the occurrence of an exceptional condition.

Syntax, PL/M and C

```
CALL rq$signal$exception (exception_code, param_num, stack_ptr,
      reserved_1, reserved_2, except_ptr);
```

```
rq_signal_exception (exception_code, param_num, stack_ptr,
      reserved_1, reserved_2, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
exception_code	WORD_16	UINT_16
param_num	BYTE	UINT_8
stack_ptr	NATIVE_WORD	NATIVE_WORD
reserved_1	WORD_16	UINT_16
reserved_2	WORD_16	UINT_16
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

`exception_code`

The condition code for the exceptional condition detected.

See also: Condition codes, *Programming Techniques*

`param_num`

The number of the parameter that caused the exceptional condition. If no parameter is at fault, this parameter equals 0.

`stack_ptr`

If not 0, this parameter must contain the value of the stack pointer (ESP) saved on entry to the OS extension. The top 5 elements (for 16-bit or 32-bit tasks) in the stack (where EBP is at the top of the stack) must be:

16-bit	32-bit	Comments
FLAGS	EFLAGS	None
CS	CS	Saved by call gate to OS extension
IP	EIP	None
DS	CS	Saved by OS extension
BP	EBP	Saved on entry

Upon completion of **signal_exception**, control is returned to either of two instructions. If `the stack_ptr` parameter is null, control returns to the instruction following the call to **signal_exception**. Otherwise, control returns to the instruction identified in EIP.

See also: Entry procedure, *System Concepts*

`reserved_1, reserved_2`

Reserved, set to 0.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Depending on the exceptional condition and the calling task's exception mode, control may or may not pass directly to the task's exception handler. If the exception handler does not get control, the condition code is returned to the calling task.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.

signal_interrupt

Used by an interrupt handler to send an end-of-interrupt (EOI) signal to the hardware and then start up an interrupt task associated with the specified level by **set_interrupt**.

See also: *:rmx:demo/c/interrupt* directory for demo using **rq_signal_interrupt**

Syntax, PL/M and C

```
CALL rq$signal$interrupt (level, except_ptr);
```

```
rq_signal_interrupt (level, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`level`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`level` Specifies the interrupt level:

Bits	Value	Meaning
15-7	0	Reserved, set to 0
6-4	0-7	First digit of the interrupt level
3	0	Bits 2-0 specify the second digit (slave)
	1	Bits 6-4 specify the entire level number (master)
2-0	0-7	Second digit of the interrupt level

See also: Interrupts, *System Concepts*

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code. All condition codes must be processed in-line, as control does not pass to an exception handler.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	No interrupt task is assigned to the specified level.
E_INTERRUPT_OVERFLOW	000AH	The interrupt task has accumulated more than the maximum allowable number of **signal_interrupt** requests (as specified by the interrupt_task_flag parameter in **set_interrupt**). See also: Interrupts, *System Concepts*
E_INT_SATURATION	0009H	This is an informative message only and does not indicate an error. See also: Interrupts, *System Concepts*
E_LIMIT	0004H	An overflow has occurred because the interrupt task has received more than 255 **signal_interrupt** requests.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The level parameter is invalid.

sleep

Places the calling task in the asleep state for a specific amount of time.

Syntax, PL/M and C

```
CALL rq$sleep (time_limit, except_ptr);
```

```
rq_sleep (time_limit, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`time_limit`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`time_limit`

Specifies:

Value	Meaning
0	Calling task is placed on the ready list, immediately behind all tasks of equal priority. If there are no such tasks, the calling task continues to run with no effect.
1-65534	Calling task goes to sleep for this many clock intervals, after which it will awake.
65535	An error is returned.

See also: For ICU-configurable systems, CIN parameter, *ICU User's Guide and Quick Reference*

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The actual time expired from execution of this call to the end of the `time_limit` parameter varies depending upon how much time remains until the next system clock interval.

See also: **create_task** example, Nucleus examples

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration. This code is returned if you make this call as an RTE call from Windows instead of from DOS.
E_PARAM	8004H	The time_limit parameter contains the invalid value 65535.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.

suspend_task

Increases by one the suspension depth of a specified task.

Syntax, PL/M and C

```
CALL rq$suspend$task (task, except_ptr);
```

```
rq_suspend_task (task, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`task`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`task` Specifies:

Value	**Meaning**
Null selector	Suspend the calling task
Valid selector	Token for the task whose suspension depth is to be incremented

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

➠ **Note**

To synchronize tasks, use mailboxes or semaphores rather than using `suspend_task` to suspend another task.

If the specified task is already in either the suspended or asleep-suspended state, its state is not changed. If the task is in the ready or running state, it enters the suspended state. If the task is in the asleep state, it enters the asleep-suspended state.

Suspend_task cannot be used to suspend interrupt tasks. Also, a task should not attempt to suspend itself while accessing a region, because this will lock up the region and the memory the task is using, and the task will never run again.

See also: **create_task** example, Nucleus examples
regions and tasks, *System Concepts*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The specified task is an interrupt task.
E_EXIST	0006H	The task parameter is not a token for an existing object.
E_LIMIT	0004H	The suspension depth for the specified task is already at the maximum of 255.
E_TYPE	8002H	The task parameter is not a token for a task.

system_accounting

Enables or disables tracking of CPU use by the operating system.

See also: **rq_get_task_accounting** to receive the tracking information

Syntax, PL/M and C

```
CALL rq_system_accounting (mode, except_ptr);
```

```
rq_system_accounting (mode, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`Mode`	BYTE	UINT_8
`except_ptr`	POINTER to WORD_16	UINT_16 *

Parameters

`mode` One of the following. If you specify the same mode that is already in effect, the call returns an E_CONTEXT exception.

Value	Meaning
0	Disables tracking of CPU use
0FFH	Enables tracking of CPU use

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Since an error will be returned if the CPU utilization mode specified in the call is already in effect, first use the **rq_get_task_accounting** call with the `reset_opt` parameter set to 0 to determine the current mode.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The tracking mode specified in the call is already in effect.

rqe_timed_interrupt

Used by an interrupt task to signal its readiness to wait a specified period of time for an interrupt.

Syntax, PL/M and C

```
CALL rqe$timed$interrupt (level, time, except_ptr);
```

```
rqe_timed_interrupt (level, time, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`level`	WORD_16	UINT_16
`time`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`level` Specifies the interrupt level:

Bits	Value	Meaning
15-7	0	Reserved, set to 0
6-4	0-7	First digit of the interrupt level
3	0	Bits 2-0 specify the second digit (slave)
	1	Bits 6-4 specify the entire level number (master)
2-0	0-7	Second digit of the interrupt level

`time` Specifies the number of clock intervals the interrupt task is willing to wait for the interrupt to occur. 0FFFFH means to wait forever.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Rqe_timed_interrupt is similar to **wait_interrupt** except that **rqe_timed_interrupt** permits the interrupt task to limit the time that it waits. If the time limit expires before an interrupt occurs, the interrupt task is resumed without servicing an interrupt.

See also: Interrupts, *System Concepts*
:rmx:demo/c/interrupt directory for demo using **rqe_timed_interrupt**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The calling task is not the interrupt task for the given level.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The level parameter is invalid.
E_TIME	0001H	The time limit expired before an interrupt occurred.

uncatalog_object

Removes an entry from the object directory of the specified job.

See also: **create_task** example, Nucleus examples

Syntax, PL/M and C

```
CALL rq$uncatalog$object (job, name, except_ptr);
```

```
rq_uncatalog_object (job, name, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
job	SELECTOR	SELECTOR
name	POINTER	void far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

job Specifies:

Value	Meaning
Null selector	Delete entry from the object directory of the calling task's job.
Valid selector	Token identifying the job of the object directory from which an entry is to be deleted.

name A pointer to a STRING containing the name of the object whose entry is to be deleted.

except_ptr
A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The pointer to the STRING is invalid. Either the selector doesn't refer to a valid segment, or the offset is outside the segment boundaries. This code is not returned when using the DOS RTE.
E_CONTEXT	0005H	The specified object directory does not contain an entry with the designated name.
E_EXIST	0006H	The job parameter is neither null nor a token for an existing object.

E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The first byte of the STRING pointed to by the name parameter contains a value greater than 12 or equal to 0.
E_STATE	0007H	This request was made in the context of a hardware interrupt handler which could cause the DOS task state to be indeterminate. This is a DOS RTE error only.
E_TYPE	8002H	The job parameter is not a token for a job.

validate_buffer

Verifies that a buffer pointer is a valid pointer to physical memory and has access rights to the memory. You can call **validate_buffer** for both normal and virtual segments.

Syntax, PL/M and C

```
CALL rq$validate$buffer (seg, offset, length, flags,
      except_ptr);
```

```
rq_validate_buffer (seg, offset, length, flags, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
seg	SELECTOR	SELECTOR
offset	WORD_32	UINT_32
length	WORD_32	UINT_32
flags	WORD_32	UINT_32
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

`seg` A token for the segment containing the buffer. The segment can be a normal or virtual segment. If `seg` is null and the application is flat model, the parameter indicates the application's virtual segment. For segmented model applications, a null value is an error.

`offset`
The offset in `seg` where the buffer begins.

`length`
The size of the buffer in bytes.

`flags` Flags set by the calling task that have the following meaning:

Bit	Meaning
0	0 = read/write access 1 = read-only access
1-31	Reserved, set to 0

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The **validate_buffer** call can be used to quickly validate a buffer without knowing what type of segment (normal or virtual) it is in. This call verifies the entire buffer pointed to by `seg:offset` of length `length` with the access rights specified in `flags`. **Validate_buffer** fails if `seg:offset` does not point to a valid segment or if any part of the physical memory within the buffer does not have the access rights specified.

The buffer itself is split into the `seg` and `offset` parameters to provide maximum flexibility, especially for flat model applications that cannot build a far pointer.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The `offset` parameter is beyond the end of `seg`, or the call specified write access, but the segment itself or one or more page frames within the buffer are read-only.
E_EXIST	0006H	The `seg` parameter represents a segment that is being deleted, or `seg` is a null token and the caller is not a flat model application
E_NOT_ALLOCATED	00F2H	The virtual segment give by `seg` does not have physical memory allocated to it somewhere between the `offset` and the `offset` plus the `length`.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	The `seg` parameter is not a token for a segment.

wait_interrupt

Used by an interrupt task to signal its readiness to service an interrupt and willingness to wait forever.

```
CALL rq$wait$interrupt (level, except_ptr);
```

```
rq_wait_interrupt (level, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`level`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`level` Specifies the interrupt level:

Bits	Value	Meaning
15-7	0	Reserved, set to 0
6-4	0-7	First digit of the interrupt level
3	0	Bits 2-0 specify the second digit (slave)
	1	Bits 6-4 specify the entire level number (master)
2-0	0-7	Second digit of the interrupt level

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Wait_interrupt is used by interrupt tasks immediately after initializing and immediately after servicing interrupts. Such a call suspends an interrupt task until the interrupt handler for the same level resumes it by invoking **signal_interrupt**.

See also: Interrupts, *System Concepts*
:rmx:demo/c/interrupt directory for demo using **rq_reset_interrupt**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_CONTEXT	0005H	The calling task is not the interrupt task for the given level.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The level parameter is invalid.

□ □ □

UDI System Calls 7

dq_allocate

Requests additional memory from the free space pool which tasks may use for any purpose.

Syntax, PL/M and C

```
seg_t = dq$allocate (size, except_ptr);

seg_t = dq_allocate (size, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
seg_t	SELECTOR	SELECTOR
size	WORD_32	NATIVE_WORD
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

seg_t A token for a memory segment. If the request fails, an E_MEM condition code returns.

Parameters

size Defines the size of the segment:

Value	Meaning
0	64 Kbytes
Not 0	The size, in bytes, of the requested segment

See also: **create_segment**

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

Dq_allocate can also return condition codes generated by the **get_pool_attrib** and **create_segment** calls.

dq_attach

Obtains a connection to a file. This call does not affect existing connections.

Syntax, PL/M and C

```
connection_t = dq$attach (path_ptr, except_ptr);

connection_t = dq_attach (path_ptr, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`connection_t`	SELECTOR	SELECTOR
`path_ptr`	POINTER	STRING far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`connection_t`
A token for the connection to the file.

Parameters

`path_ptr`
A pointer to a STRING containing the pathname of the file to be attached.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Use the **dq_reserve_io_memory** call to reserve memory before making a **dq_attach** call. This reserves enough memory for UDI internal data structures and buffers. Insufficient memory can cause a **dq_attach** call to fail.

See also: **dq_reserve_io_memory**

When a task makes its first UDI call, a UDI environment is set up for the task. This includes the UDI default <Ctrl-C> handler, which is the same as the HI default, and overrides any previously set up <Ctrl-C> handler.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_FNEXIST	0021H	The specified file does not exist.
E_MEM	0002H	Insufficient memory exists for the requested operation.
E_SUPPORT	0023H	An unsupported operation was attempted.

Dq_attach can also generate condition codes returned by the EIOS call **s_attach_file**.

dq_change_access

Changes the owner or World user access rights to a file or directory. This call cannot change the read permissions for the Super user.

Syntax, PL/M and C

```
CALL dq$change$access (path_ptr, user, access, except_ptr);

dq_change_access (path_ptr, user, access, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`path_ptr`	POINTER	STRING far *
`user`	BYTE	UINT_8
`access`	BYTE	UINT_8
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`path_ptr`

A pointer to a STRING containing the file's pathname.

`user` Specifies the user whose access is to be changed. Use these numeric values. Specify DOS files as World.

Value	**User**
0	Owner of the file
1	World (all users on the system)
2	Group (ignored by the iRMX OS)
other	If used, an E_SUPPORT condition code returns.

`access`

Specifies the access to be granted the user. The user always has read access (bit 1) to DOS files and directories. Optionally select read/write access by setting bits 3, 2, or 0.

Bit	**Meaning**
7-5	Reserved. If used, an E_SUPPORT exception returns.
4	Execute the file. Set to the same value as bit 1 for compatibility with other OSs. Does not apply to iRMX OS files. iRMX OS users with write access may execute files.
3	Update (read and write) the file or change the directory access.
2	Append to the file or add an entry to the directory
1	Read the file or list the directory
0	Delete the file or directory

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Changing the access rights for a user ID does not effect a program's existing connections. But, **dq_change_access** changes the access granted when the program makes subsequent calls to **dq_attach.**

See also: User IDs, default users, access masks, World, access rights, owner IDs and connections, *System Concepts*

Condition Codes

E_OK	000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SUPPORT	0023H	The value specified for the user parameter is greater than 2. You tried to set bits 7-5 of the access parameter.
E_FACCESS	0026H	Access to the specified file is denied.

Dq_change_access can also return condition codes generated by the **s_change_access**.

dq_change_extension

Changes the filename extension as stored in memory, not on the mass storage volume. Filename extensions consist of the 3 characters that follow the last period of a filename.

Syntax, PL/M and C

```
CALL dq$change$extension (path_ptr, extension_ptr, except_ptr);
```

```
dq_change_extension (path_ptr, extension_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
path_ptr	POINTER	STRING far *
extension_ptr	POINTER	UINT_8 far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

path_ptr

A pointer to a STRING that contains the existing pathname for the file.

extension_ptr

A pointer to a series of three bytes containing the filename extension. This is not a STRING. Include 3 bytes, even if some bytes are blank.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Dq_change_extension changes or adds filename extensions of no more than 3 characters. For example, a compiler can use **dq_change_extension** to create the name of an object file (*:afd1:file.obj*) from a source file (*:afd1:file.src*).

The three-character filename extension may not contain delimiters recognized by **dq_get_argument** but may contain trailing blanks. If the first character pointed to by **extension_ptr** is a space, **dq_change_extension** deletes the existing extension.

See also: Delimiters, **dq_get_argument**

Condition Codes

E_OK	000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_STRING_BUFFER	0081H	The filename is more than 14 characters, including the period and extension.

dq_close

Closes a file connection that was opened by the **dq_open** system call.

Syntax, PL/M and C

```
CALL dq$close (connection_t, except_ptr);

dq_close (connection_t, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection_t`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection_t`

A token for an open file connection.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Dq_close functions are:

1. Waits until all currently running I/O operations for the connection are completed.
2. Ensures that any information in a partially-filled output buffer is written to the file.
3. Releases all buffers associated with the connection.
4. Closes the connection. The connection is still valid, and can be re-opened if necessary.

Condition Codes

E_OK	000H	This call returns E_OK if the connection is already closed. No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

Dq_close can also return condition codes generated by **s_close**.

dq_create

Creates a new file with the specified name and returns a connection for it. If a file exists with the same name, the existing file is truncated to 0 length and the data is destroyed.

⚠ **CAUTION**
To prevent accidentally destroying a file, call **dq_attach** before calling **dq_create**.

See also: **dq_create** example, UDI example

Syntax, PL/M and C

```
connection$t = dq_create (path_ptr, except_ptr);

connection_t = dq_create (path_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection_t	SELECTOR	SELECTOR
path_ptr	POINTER	STRING far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

connection_t
A token for the connection to the file.

Parameters

path_ptr
A pointer to a STRING containing a pathname for the file to be created.

except_ptr
A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_MEM	0002H	Insufficient memory remains to complete the call.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SPACE	0029H	Insufficient space exists on a direct-access device.
E_SUPPORT	0023H	An unsupported operation was attempted.

Dq_create can also return condition codes generated by **s_create_file** and **s_delete_file**.

dq_decode_exception

Returns the hexadecimal equivalent and mnemonic of the specified numeric condition code.

Syntax, PL/M and C

```
CALL dq$decode$exception (exception_code, buff_ptr,
      except_ptr);
```

```
dq_decode_exception (exception_code, buff_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
exception_code	WORD_16	UINT_16
buff_ptr	POINTER	STRING far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

exception_code

A location containing the numeric condition code that is to be translated.

buff_ptr

A pointer to a STRING where the hexadecimal value and mnemonic returns. This STRING should be at least 81 bytes long to accept the maximum size returned. For example, if you specify 2 in the exception_code parameter, the system returns:

```
0002H: E_MEM
```

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

Dq_decode_exception can also return condition codes generated by **c_format_exception**.

dq_decode_time

Returns the indicated date and time, each as a series of ASCII bytes. (Note that they are not strings.) You can also use **dq_decode_time** to decode the specified binary date/time value to ASCII characters.

Syntax, PL/M and C

```
CALL dq$decode$time (date_time_ptr, except_ptr);
```

```
dq_decode_time (date_time_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
date_time_ptr	POINTER	DATE_TIME_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

date_time_ptr

A pointer to this structure:

```
DECLARE date_time STRUCTURE(
   system_time          WORD_32,
   date(8)              BYTE,
   time(8)              BYTE);
```

or

```
typedef struct {
   UINT_32              system_time;
   UINT_8               date[8];
   UINT_8               time[8];
} DATE_TIME_STRUCT;
```

Where:

system_time

Specifies the date and time value to be decoded.

Value	Meaning
0	Requests the current date and time be returned
not 0	The number of seconds since midnight, January 1, 1978

date — The returned date in ASCII characters of the form MM/DD/YY for month, day, and year. The slashes (/) are in the third and sixth bytes.

time — The returned time in ASCII characters of the form HH:MM:SS for hours, minutes, and seconds, with separating (:) colons.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SUPPORT	0023H	An unsupported operation was attempted.

Dq_decode_time may also return condition codes generated by **get_time**.

dq_delete

Marks a file for deletion and disallows new connections. The file is finally deleted only when all open connections are removed.

See also: **dq_detach**

Syntax, PL/M and C

```
CALL dq$delete (path_ptr, except_ptr);
```

```
dq_delete (path_ptr, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`path_ptr`	POINTER	STRING far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`path_ptr`

A pointer to a STRING containing a pathname of the file to be deleted.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_FNEXIST	0021H	The specified file does not exist.
E_FACCESS	0026H	Access to the specified file is denied.

Dq_delete may also return condition codes generated by **s_delete_file**.

dq_detach

Deletes a file connection established by **dq_attach** or **dq_create**. If the connection is open, this call invokes **dq_close** first. If the file has been marked for deletion, **dq_detach** also deletes the file.

Syntax, PL/M and C

```
CALL dq$detach (connection_t, except_ptr);
```

```
dq_detach (connection_t, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection_t`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection_t`

A token for a file connection.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

Dq_detach may also return condition codes generated by **dq_close** and **s_delete_connection**.

dq_exit

Terminates a program, closing and detaching all open connections, and returning all allocated memory to the memory pool. No condition codes return to this call.

Syntax, PL/M and C

```
CALL dq$exit (completion_code);
```

```
dq_exit (completion_code);
```

Parameter	PL/M Data Type	C Data Type
`completion_code`	WORD_16	UINT_16

Parameter

`completion_code`

An encoded reason for termination. **Dq_exit** converts the `completion_code` value into a condition code.

	Condition		
Value	**Code**	**Mnemonic**	**Meaning**
0	0000H	E_OK	Termination was normal.
1	0C1H	E_WARNING_EXIT	Warning messages were issued.
2	0C2H	E_ERROR_EXIT	Errors were detected.
3	0C3H	E_FATAL_EXIT	Fatal errors were detected.
4	0C4H	E_ABORT_EXIT	The job was aborted.
5-65535	0C0H	E_UNKNOWN_EXIT	Cause for exit not known.

Additional Information

Typically the calling task is running in an I/O job. The job's response mailbox receives one of the condition codes described above. **Dq_exit** invokes **exit_io_job**, placing the condition code in the `user_fault_code` parameter. **Exit_io_job** places the code in a structure that is sent to the response mailbox. The calling task may then invoke **dq_decode_exception** to convert the condition code into its associated mnemonic.

See also: **create_io_job** and **exit_io_job**

dq_file_info

Returns information about the specified directory or data file.

Syntax, PL/M and C

```
CALL dq$file$info (connection_t, mode, file_info_ptr,
      except_ptr);
```

```
dq_file_info (connection_t, mode, file_info_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection_t	SELECTOR	SELECTOR
mode	BYTE	UINT_8
file_info_ptr	POINTER	U_FILE_INFO_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection_t
A token for a file connection.

mode Specifies this:

Value	Meaning
0	Do not return the file owner's user ID.
1	Return the owner's user ID.
2-255	Return E_SUPPORT condition code.

file_info_ptr
A pointer to this structure:

```
DECLARE u_file_info STRUCTURE(
   owner(15)              BYTE,
   length                 WORD_32,
   type                   BYTE,
   owner_access           BYTE,
   world_access           BYTE,
   create_time            WORD_32,
   last_mod_time          WORD_32,
   group_access           BYTE,
   reserved(19)           BYTE);
```

or

```
typedef struct {
   UINT_8                 owner[15];
   UINT_32                length;
   UINT_8                 type;
   UINT_8                 owner_access;
   UINT_8                 world_access;
   UINT_32                create_time;
   UINT_32                last_mod_time;
   UINT_8                 group_access;
   UINT_8                 reserved[19];
} U_FILE_INFO_STRUCT;
```

Where:

`owner` The user ID of the file's owner if it is requested.

`length` The size of the file in bytes.

`type` Indicates the file type.

Value	**File Type**
0	Data file
1	Directory file
2	System-specific file
3-255	Reserved

`owner_access`
Indicates this:

Bits	**Meaning**
7-5	Reserved
4	Execute the file. This bit is always set to 0.
3	Update a file or change access to the directory.
2	Append to a data file or add entry to the directory.
1	Read a data file or display a directory.
0	Delete.

`world_access`

Indicates the access granted to the World user.

Bit	Associated Access Type
7-5	Reserved.
4	Execute the file. Set to the same value as bit 1 for compatibility with other OSs.
3	Update a file or change access to the directory.
2	Append to a data file or add entry to the directory.
1	Read a data file or display a directory.
0	Delete.

Note

DOS does not make distinctions between types of access. For DOS files, `owner_access` and `world_access` are the same.

`create_time`

The date and time that the file or directory was created, expressed as the number of seconds since midnight, January 1, 1978. Convert this date/time to ASCII characters by calling **dq_decode_time**.

`last_mod_time`

The date and time that the file or directory was last modified. For data files, modified means written to or truncated; for directories, modified means an entry was changed or an entry was added. Convert this date/time to ASCII characters by calling **dq_decode_time**.

`group_access`

Always set to the value of the `world_access` field.

`reserved` Reserved.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SUPPORT	0023H	The mode parameter has a value greater than 1.

Dq_file_info can also return condition codes generated by **create_mailbox** and **receive_message** and **a_get_file_status**.

dq_free

Frees a segment by returning its allocated memory to the memory pool from which it was allocated using **dq_allocate**. Subsequent attempts to use a deleted segment may cause errors or unexpected results, because the memory may be reallocated.

See also: **dq_allocate**

Syntax, PL/M and C

```
CALL dq$free (seg_t, except_ptr);
```

```
dq_free (seg_t, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
seg_t	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

seg_t A token for segment. When the segment is freed, this token is no longer valid.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.

Dq_free can also return condition codes generated by **delete_segment**.

dq_get_argument

Gets arguments, one at a time, from a command line entered at the system console. This command line is either the one that invoked the program containing the **dq_get_argument** call or a command line entered while the program is running.

Syntax, PL/M and C

```
delimit_char = dq$get$argument (argument_ptr, except_ptr);
```

```
delimit_char = dq_get_argument (argument_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`delimit_char`	BYTE	UINT_8
`argument_ptr`	POINTER	STRING far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`delimit_char`

Receives the delimiter character. A delimiter returns only if the condition code is E_OK. The OS recognizes these delimiters:

```
,  )  (  =  #  !  %  +  -  &  ;  <  >  [  ]  \  '  |  ~
```

The OS also recognizes the ASCII character values ranging from 1 through 20H and between 7FH and 0FFH as delimiters; this includes the space and carriage return <CR>.

See also: Delimiters, *System Concepts*

Parameters

`argument_ptr`

A pointer to a STRING that receives the argument. This STRING should be at least 81 bytes long to accept the maximum size returned.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

There are two buffering methods for command line arguments:

- For invocation command lines that invoke a program from the console, there is a default command line buffer.
- For arguments entered in response to requests within a program, a program must create a buffer and read the command line into a buffer using **dq_read**.

Before returning argument STRINGs, **dq_get_argument** edits the STRINGs in the argument buffer.

- Ampersands (&) and semicolons (;) are deleted.
- Multiple spaces between arguments are replaced with a single space. Tabs are treated as spaces.
- Lowercase characters are converted to uppercase, unless they are contained in quotes.
- The command line and the argument buffer, after a **dq_switch_buffer** system call, are preceded by a null delimiter.

Dq_get_argument returns characters enclosed in matching pairs of single or double quotes as literals. Enclosing quotes are not returned as part of the argument.

Example

This example shows the arguments and delimiters returned by successive calls to **dq_get_argument**. The example buffer contains this command line:

```
PLM386 LINKER.PLM PRINT(:LP:) NOLIST
```

Calling **dq_get_argument** five times returns this output:

Call	**Delimiter**	**Argument Returned**
1	space	(06H)PLM386
2	space	(0AH)LINKER.PLM
3	space	((05H)PRINT
4	)	(04H):LP:
5	CR	(06H)NOLIST

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_STRING_BUFFER	0081H	The argument exceeds 80 characters. Issue another call to **dq_get_argument** to obtain the rest of the argument.

dq_get_connection_status

Returns information about a file connection. Use this system call to determine the file pointer location after a program performs several read or write operations.

Syntax, PL/M and C

```
CALL dq$get$connection$status (connection_t, info_ptr,
      except_ptr);
```

```
dq_get_connection_status (connection_t, info_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection_t`	SELECTOR	SELECTOR
`info_ptr`	POINTER	U_CONN_STATUS_STRUCT far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection_t`

A token for a connection.

`info_ptr`

A pointer to this structure:

```
DECLARE u_conn_status STRUCTURE(
   open                   BYTE,
   access                 BYTE,
   seek                   BYTE,
   file_ptr               WORD_32);
```

or

```
typedef struct {
   UINT_8                 open;
   UINT_8                 access;
   UINT_8                 seek;
   UINT_32                file_ptr;
} U_CONN_STATUS_STRUCT;
```

Where:

open
: Indicates this:

Value	Meaning
0	The connection is not open.
0FFH	The connection is open.

access
: Indicates user access to the connection. The user always has read access to DOS files and directories. Read/write access is optional.

Bit	Meaning
7-5	Reserved.
4	Execute the file. Set to the value of bit 1 for compatibility with other OSs. This bit does not apply to iRMX III files. iRMX III OS users with write access may execute files.
3	Update the file or change access to the directory.
2	Append to the file or add entry to the directory.
1	Read the file or list the directory.
0	Delete the file or directory.

seek
: Indicates the types of seek supported.

Value	Meaning
0	No seek allowed
3	Seek forward and backward

file_ptr
: The current position of the file pointer, expressed as the number of bytes from the beginning of the file. Byte 0 is the first byte. This field is undefined if the file is not open or if seek is not supported by the device. For example, seek operations are not valid for a line printer.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SUPPORT	0023H	An unsupported operation was attempted.

Dq_get_connection_status can also return condition codes generated by **s_get_connection_status**.

dq_get_exception_handler

Returns the address of the current exception handler.

Syntax, PL/M and C

```
CALL dq$get$exception$handler (current_handler_ptr,
      except_ptr);
```

```
dq_get_exception_handler (current_handler_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
current_handler_ptr	POINTER	HANDLER_PTR_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

current_handler_ptr

A pointer to the entry point of the current exception handler. **Dq_trap_exception** specifies this entry point if it is called.

```
typedef struct {
   NATIVE_WORD          offset;
   SELECTOR             base;
} HANDLER_PTR_STRUCT;
```

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This call returns the address specified in the most recent call to **dq_trap_exception**, if any. Otherwise, the value returned is the address of the system default exception handler.

See also: **dq_trap_exception**, **dq_decode_exception**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

Dq_get_exception_handler can also return condition codes generated by **get_exception_handler**.

dq_get_msize

Returns the size of a segment allocated by the **dq_mallocate** system call.

Syntax, PL/M and C

```
size = dq$get$msize (seg_ptr, exception_ptr);

size = dq_get_msize (seg_ptr, exception_ptr);
```

Parameter	PL/M Data Type	C Data Type
`size`	WORD_32	UINT_32
`seg_ptr`	POINTER	UINT_8 far *
`exception_ptr`	POINTER	UINT_16 far *

Return Value

`size` The size in bytes of the memory block previously allocated by **dq_mallocate**. Since, for flat model applications **dq_mallocate** rounds up the request to the next 4 Kbyte boundary, the size returned in this call is the rounded-up size, not necessarily the original requested size.

Parameters

`seg_ptr`

A pointer to the memory block.

`exception_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SUPPORT	0023H	An unsupported operation was attempted.

Dq_get_msize can also return condition codes generated by **get_size**.

dq_get_size

Returns the size of a previously allocated memory segment.

Syntax, PL/M and C

```
size = dq$get$size (seg_t, except_ptr);
```

```
size = dq_get_size (seg_t, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
size	WORD_32	NATIVE_WORD
seg_t	SELECTOR	SELECTOR
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

size Indicates this:

Value	Meaning
0	For 16-bit applications only, this value indicates that the segment size is 64 Kbytes.
not 0	The size in bytes of the segment.

Parameters

seg_t A token for a segment of memory allocated by the **dq_allocate** call.

except_ptr
A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

Dq_get_msize can also return condition codes generated by **get_size**.

dq_get_system_id

Returns the version number of the iRMX OS.

Syntax, PL/M and C

```
CALL dq$get$system$id (id_ptr, except_ptr);
```

```
dq_get_system_id (id_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
id_ptr	POINTER	UINT_8 far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

id_ptr

A pointer to a 21-byte buffer where the identity of the OS returns.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

dq_get_time

Obsolete. Use the more general **dq_decode_time** system call for this function.

dq_mallocate

Requests that a specific amount of logically contiguous memory be added to the memory pool of the calling program. If successful, the call returns a pointer to the first byte of the acquired memory. Multiple calls to **dq_mallocate** result in multiple segments being allocated. For a flat model application, this call allocates physical memory into the flat address space of the application, instead of creating an iRMX segment.

Syntax, PL/M and C

```
seg_ptr = dq$mallocate (size, except_ptr);

seg_ptr = dq_mallocate (size, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`seg_ptr`	POINTER	UINT_8 far *
`size`	WORD_32	UINT_32
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`seg_ptr`
A pointer to the first byte of the acquired block of memory.

Parameters

`size` Requests the number of bytes of memory. If you specify a size of 0, the call allocates 64 Kbytes (10000H). For flat model applications only, the minimum amount of memory returned is 4 Kbytes, and all requests are rounded up to the next 4 Kbyte boundary.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

For flat model applications only, this call invokes the paging subsystem call **rqv_allocate**.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.

E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SUPPORT	0023H	An unsupported operation was attempted.

Dq_mallocate can also return condition codes generated by **get_pool_attrib**, **create_segment**, or **rqv_allocate**.

dq_mfree

Releases an entire block of block of memory, previously acquired using **dq_mallocate**, to the memory pool. The freed memory is no longer available to the calling program and may be reallocated to another process. You cannot release just a portion of the memory.

Syntax, PL/M and C

```
CALL dq$mfree (seg_ptr, exception_ptr);
```

```
dq_mfree (seg_ptr, exception_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
seg_ptr	POINTER	UINT_8 far *
exception_ptr	POINTER	UINT_16 far *

Parameters

seg_ptr

A pointer to a block of memory.

exception_ptr

A pointer to a location where the condition code returns.

Additional Information

For flat model applications only, this call invokes the paging subsystem call **rqv_free**.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	For flat model applications only, the seg_ptr parameter does not point to valid allocated physical memory within the caller's virtual segment.

Dq_mfree may also return condition codes generated by **delete_segment** or **rqv_free**.

dq_open

Opens a file for I/O operations. **Dq_open** prepares a connection for use with **dq_read**, **dq_write**, **dq_seek**, and **dq_truncate**.

Syntax, PL/M and C

```
CALL dq$open (connection_t, mode, num_buf, except_ptr);
```

```
dq_open (connection_t, mode, num_buf, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
connection_t	SELECTOR	SELECTOR
mode	BYTE	UINT_8
num_buf	BYTE	UINT_8
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

connection_t

A token for the file connection to be opened.

mode Specifies the file access mode.

Value	Meaning
1	Read only
2	Write only
3	Update
4	Reserved
5-7	Available for UNIX systems; ignored for the iRMX OS
8-255	Reserved

num_buf

Specifies the kind of buffering needed for this connection.

Value	Meaning.
0	No buffering required.
1-2	Requests double buffering which automatically performs read-ahead and/or write-behind buffering
3-255	The E_SUPPORT condition code returns.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Dq_open does this:

1. Creates the requested buffers.
2. Sets the connection's file pointer to the beginning of the file.
3. Starts reading ahead if `num_buf` is greater than 0 and the `access` parameter is read only or update.

Use the **dq_reserve_io_memory** call to reserve memory before making a **dq_open** call. This reserves enough memory for UDI internal data structures and buffers. Insufficient memory can cause a **dq_open** call to fail.

See also: **dq_reserve_io_memory**

The amount of buffers that you choose affects system performance. These performance guidelines are true of every system:

- Request at least two buffers to overlap I/O with computation.
- Request no buffers if memory is more important than performance.
- Request no buffers for interactive programs when opening *:ci:* and *:co:*.
- Request one buffer if your program normally calls **dq_seek** before calling **dq_read** or **dq_write**.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SUPPORT	0023H	At least one of these is true: • The `mode` parameter is set to a reserved value. • The num_buffs parameter is greater than 2.
E_FACCESS	0026H	Access to the specified file is denied.
E_SHARE	0028H	The specified file may not be shared.
E_MEM	0002H	Insufficient memory remains to complete the call.

Dq_open can also return condition codes generated by **s_open**.

dq_overlay

Loads an overlay module. The root module calls this system call. Overlay code is 16-bit code that runs in PVAM.

See also: Overlays, *System Concepts*

Syntax, PL/M and C

```
CALL dq$overlay (name_ptr, except_ptr);
```

```
dq_overlay (name_ptr, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`name_ptr`	POINTER	STRING far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`name_ptr`

A pointer to a STRING containing the name of an overlay module. The name must be in uppercase.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

To maintain portability to other OSs that support the UDI, call no more than one level of overlay from the root module.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SUPPORT	0023H	An supported operation was attempted.

Dq_overlay can also return condition codes generated by **s_overlay**.

dq_read

Reads contiguous bytes from a file and places them in the specified buffer. The connection must be open for reading and updating.

Syntax, PL/M and C

```
bytes_read = dq$read (connection_t, buff_ptr, count,
      except_ptr);
```

```
bytes_read = dq_read (connection_t, buff_ptr, count,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`bytes_read`	WORD_32	NATIVE_WORD
`connection_t`	SELECTOR	SELECTOR
`buff_ptr`	POINTER	UINT_8 far *
`count`	WORD_32	NATIVE_WORD
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`bytes_read`

The number of bytes actually read. This number is always equal to or less than the `count` parameter.

Parameters

`connection_t`

A token for the connection to the file. The file pointer must point to the first byte to be read.

See also: **dq_seek**

`buff_ptr`

A pointer to a STRING that receives the data from the file. The STRING must be at least as large as the `count` parameter.

`count`

The number of bytes to be read from the file.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SUPPORT	0023H	An unsupported operation was attempted.

Dq_read can also return condition codes generated by **s_read_move**, with the exception of E_FLUSHING.

dq_rename

Changes a directory or data file's pathname. Renaming a directory changes the pathnames of all files contained in the directory. Existing connections to a renamed file remain established.

Syntax, PL/M and C

```
CALL dq$rename (path_ptr, new_path_ptr, except_ptr);
```

```
dq_rename (path_ptr, new_path_ptr, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`path_ptr`	POINTER	STRING far *
`new_path_ptr`	POINTER	STRING far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`path_ptr`

A pointer to a STRING for the file's existing pathname.

`new_path_ptr`

A pointer to a STRING for the file's new pathname. This pathname must not be an existing pathname.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Successfully renaming a file without appropriate access permission depends on the OS.

DOS users cannot rename a file or a directory to a different subdirectory. Otherwise, a file's pathname can be changed in any way, if the file or directory remains on the same volume. If an OS does not allow renaming a file to another volume or storage device, an E_SUPPORT exception returns.

Condition Codes

E_OK	000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_FEXIST	0020H	The file represented by new_path_ptr already exists.
E_SUPPORT	0023H	The file represented by new_path_ptr exists on another volume.
E_FNEXIST	0021H	The file represented by path_ptr does not exist.

Dq_rename can also return condition codes generated by **s_rename_file**.

dq_reserve_io_memory

Reserves enough memory to ensure that your program can open and attach the files. Use this call only if your program exclusively uses UDI system calls to communicate with the OS.

Syntax, PL/M and C

```
CALL dq$reserve$io$memory (number_files, number_buffers,
      except_ptr);
```

```
dq_reserve_io_memory (number_files, number_buffers,
      except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`number_files`	WORD_16	UINT_16
`number_buffers`	WORD_16	UINT_16
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`number_files`

Specifies the maximum number of files, up to 12, the program will attach. No more than 6 files may be open simultaneously.

`number_buffers`

Specifies the total number of buffers, up to 12, that will be needed at one time. For example, if your program has 2 files open at the same time and each of them has 2 buffers, `number_files` should be 2 and `number_buffers` should be 4.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Memory reserved with this call cannot be allocated by **dq_allocate** or **dq_mallocate**.

If you specify a 0 for both `number_files` and `number_buffers`, the memory reserved returns to the memory pool.

Use this call to reserve memory before using **dq_attach** and **dq_open**. Otherwise, the memory used by those calls counts against the file and buffer counts specified in this call. This can exhaust the memory supply.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_MEM	0002H	Insufficient memory remains to complete the call.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SUPPORT	0023H	At least one of these is true: • The value specified for `number_files` is greater than 12. • The value specified for number_buffers is greater than 12.

dq_seek

Positions the file pointer to a location where a non-sequential I/O operation using the **dq_read**, **dq_truncate**, or **dq_write** calls begins. Do not use this call for applications performing stream I/O operations.

Syntax, PL/M and C

```
CALL dq$seek (connection_t, mode, off_set, except_ptr)
```

```
dq_seek (connection_t, mode, off_set, except_ptr)
```

Parameter	PL/M Data Type	C Data Type
`connection_t`	SELECTOR	SELECTOR
`mode`	BYTE	UINT_8
`off_set`	WORD_32	UINT_32
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection_t`

A token for an open file connection.

`mode` Specifies the file pointer movement.

Value	File Pointer Movement
1	Back by off_set bytes; if the pointer moves past the beginning of the file, it is set to 0 (first byte).
2	Set to the position specified by the off_set parameter. Moving the beyond the EOF is permitted.
3	Forward by off_set bytes. Moving beyond the EOF is permitted.
4	Move to the EOF and then back by off_set bytes; if the pointer moves beyond the beginning of the file, it is set to 0 (first byte). This option is not supported for EDOS directories; E_SUPPORT returns.

`off_set`

Specifies how far, in bytes, to move the file pointer.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

When the file pointer is positioned beyond the EOF, this happens:

- **Dq_read** behaves as though the read operation began at the EOF. A subsequent read returns an indication of an EOF.
- **Dq_write** extends the file and the data is written as requested. Attempting a seek past the end of a file without performing an explicit **dq_write** call produces undetermined results.

See also: **a_seek**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	0023H	The mode parameter was set outside the range 1-4.

Dq_seek can also return condition codes generated by the EIOS call **s_seek**.

dq_special

Changes the operating mode for a terminal input device.

Syntax, PL/M and C

```
CALL dq$special (mode, parameter_ptr, except_ptr);

dq_special (mode, parameter_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
mode	BYTE	UINT_8
parameter_ptr	POINTER	void far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

mode Specifies the options to be set or the actions to be performed .

Value	Mode	Description.
1	Transparent	Enables interactive applications to obtain characters from the console exactly as typed. Two exceptions to this are (1) signal characters (e.g., the HI <Ctrl-C>) set by specifying "set signal" in the spec_func parameter of **a_special** or **s_special**, and (2) any enabled output control characters or OSC sequences
2	Line Editing	Use this option to correct typing errors by using special keys before the program receives the characters that are typed. Characters used for editing are OS-dependent. The carriage return <CR> character is always converted to carriage return-line feed <CRLF>. This is the default mode when the system starts to run.
3	Polling	This mode is almost the same as Transparent mode except that characters typed between successive calls to read the terminal are held in the type-ahead buffer.
4-5	Reserved	E_SUPPORT returns.
6	Baud Rate	Specifies baud rate selection through the structure pointed to by parameter_ptr.

`parameter_ptr`

A pointer to this structure:

```
DECLARE line STRUCTURE (
   conn                    SELECTOR,
   in_baud_rate            BYTE,
   out_baud_rate           BYTE);
```

or

```
typedef struct {
   SELECTOR                conn;
   UINT_8                  in_baud_rate;
   UINT_8                  out_baud_rate
} LINE_STRUCT;
```

Where:

`conn` A token for a connection previously established using **dq_attach**.

`in_baud_rate`

The input baud rate encoded.

Value	**Baud Rate**
0	Unspecified
1	300
2	600
3	1200
4	2400
5	4800
6	9600
7	19200
8-255	Reserved

`out_baud_rate`

The output baud rate encoded as above.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

In transparent mode, normal input characters are placed in the buffer specified by the call to **dq_read**. **Dq_read** returns control to the calling program when the number of characters entered equals the number of characters specified in the read request.

In polling mode, **dq_read** returns control to your program immediately after it is called, regardless of whether any characters were typed since the last call to **dq_read**. If no characters have been typed, this is indicated by the `bytes_read` parameter of the **dq_read** call.

See also: **dq_read**

Condition Codes

E_OK	000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SUPPORT	0023H	The mode parameter represents an unsupported mode.

Dq_special can also return the codes generated by the EIOS call **s_special**.

dq_switch_buffer

Substitutes the specified command line buffer for the existing buffer.

Syntax, PL/M and C

```
char_offset = dq$switch$buffer (buff_ptr, except_ptr);

char_offset = dq_switch_buffer (buff_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
char_offset	WORD_32	NATIVE_WORD
buff_ptr	POINTER	UINT_8 far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Return Value

char_offset

The offset location in bytes from the beginning of the command line to the last character of the last argument retrieved by **dq_get_argument**. Use this offset to determine the current argument pointer location in the command line.

Parameters

buff_ptr

A pointer to a STRING containing a new command line buffer. The buffer must not exceed 32 Kbytes in length.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

Dq_switch_buffer can also return condition codes generated by **c_set_parse_buffer**.

dq_trap_cc

Substitutes an alternate interrupt procedure that will receive control when you enter an interrupt character such as <Ctrl-C> on the console.

Syntax, PL/M and C

```
CALL dq$trap$cc (cc_routine_ptr, except_ptr);
```

```
dq_trap_cc (cc_routine_ptr, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`cc_routine_ptr`	POINTER	HANDLER_PTR_STRUCT far *
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`cc_routine_ptr`

A pointer to the entry point of the alternate interrupt procedure.

```
typedef struct {
   NATIVE_WORD          offset
   SELECTOR             base;
} HANDLER_PTR_STRUCT;
```

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The OS saves a program's execution context when **dq_trap_cc** is invoked. Due to the context switch when the interrupt procedure receives control, the contents of the CPU registers at that time may not be those associated with your program. For example, the CPU registers may contain values for an internal task that was executing when the interrupt character was entered.

See also: Interrupt routines and characters, *System Concepts*

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

dq_trap_exception

Designates an alternate exception handler.

Syntax, PL/M and C

```
CALL dq$trap$exception (handler_ptr, except_ptr);
```

```
dq_trap_exception (handler_ptr, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
handler_ptr	POINTER	HANDLER_PTR_STRUCT far *
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

handler_ptr

A pointer to this structure containing the entry point of the alternate exception handler.

```
typedef struct {
   NATIVE_WORD          offset
   SELECTOR             base;
} HANDLER_PTR_STRUCT;
```

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The **dq_trap_exception** routine should restore the default exception handler before it terminates. Therefore, a program should call **dq_get_exception_handler** before calling **dq_trap_exception** to get the default exception handler address.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.

Dq_trap_exception can also return condition codes generated by **set_exception_handler**.

dq_truncate

Truncates a file at the current position of the file pointer and releases file space beyond the pointer to other files. If the pointer is at or beyond the EOF, no truncation is performed. Use the **dq_seek** system call to position the pointer before calling **dq_truncate**.

Syntax, PL/M and C

```
CALL dq$truncate (connection_t, except_ptr);
```

```
dq_truncate (connection_t, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`connection_t`	SELECTOR	SELECTOR
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection_t`

A token for a connection to a named or DOS data file. The byte indicated by the file pointer is the first byte to be dropped from the file. The connection should have write, or read/write access rights.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SUPPORT	0023H	An unsupported operation was attempted.

Dq_truncate can also return condition codes generated by **s_truncate_file**.

dq_write

Writes a number of bytes from a buffer to a file. Use **dq_seek** to position the file pointer.

Syntax, PL/M and C

```
CALL dq$write (connection_t, buff_ptr, count, except_ptr);
```

```
dq_write (connection_t, buff_ptr, count, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`connection_t`	SELECTOR	SELECTOR
`buff_ptr`	POINTER	UINT_8 far *
`count`	WORD_32	NATIVE_WORD
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`connection_t`
A token for a connection to the file being written to.

`buff_ptr`
A pointer to a buffer containing the data to be written to the specified file.

`count` The number of bytes to be written from the buffer to the file.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Dq_write may write fewer bytes than requested by the calling program. This happens under these circumstances:

- When **dq_write** encounters an I/O error
- When the volume to which a program is writing becomes full

After the writing operation is completed, the file pointer points to the byte immediately following the last byte written.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SUPPORT	0023H	An unsupported operation was attempted.
E_SPACE	0029H	Inadequate memory space remains to complete the write.

Dq_write can also generate condition codes from **s_write_move**.

❑ ❑ ❑

Windows- and DOS-Specific System Calls 8

rqe_read_segment iRMX for Windows only

Enables a DOS application program to transfer data from a PVAM segment to a Real Mode segment. The maximum size of the transfer is limited to 65535 bytes, the length of a Real Mode segment.

⇒ **Note**
This system call is used by DOS applications only; it is not supported in the iRMX OS.

Syntax, PL/M and C

```
CALL rqe$read$segment (pvam_seg, pvam_offset, realmode_ptr,
      size, status_ptr);
```

```
rqe_read_segment (pvam_seg, pvam_offset, realmode_ptr, size,
      status_ptr);
```

Parameter	PL/M Data Type	C Data Type
pvam_seg	SELECTOR	SELECTOR
pvam_offset	WORD_32	UINT_32
realmode_ptr	POINTER	void far *
size	WORD_16	UINT_16
status_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

pvam_seg

Specifies the PVAM source segment. This must be a valid selector, such as the token for a segment that was received from a mailbox.

pvam_offset

The offset into the source segment where the transfer starts.

realmode_ptr

A pointer to the destination segment.

`size` Specifies the amount of data being transferred. If the size of the transfer is greater than the limit of the source segment, an exceptional condition code returns and no transfer takes place.

> **CAUTION**
> If the size of the destination segment is less than the requested size, the transfer takes place anyway and corrupts the DOS application program's memory.

`status_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The DOS application program calling sequence is shown here. See your compiler documentation for guidance in constructing an assembly language routine to do this:

1. Push the `pvam_seg` parameter onto the stack.
2. Push the `pvam_offset` parameter onto the stack.
3. Push the `realmode_ptr` parameter onto the stack.
4. Push the `size` parameter onto the stack.
5. Push the `status_ptr` parameter onto the stack.
6. Put the function code 30 into the AX CPU register.
7. Put the offset of the `status_ptr` parameter into the SI CPU register.
8. Cause a software interrupt number B8H.
9. Clear the stack.

Condition Codes

E_OK	0000H	Data transfer successful.
E_NUC_BAD_BUF	80E2H	Indicates one of these: • `Pvam_seg` does not refer to a valid segment. • `Pvam_offset` is outside the segment boundaries. • The specified size would cause the data transfer to exceed the pvam_seg limit.

rqe_write_segment

Enables a DOS application program to transfer data from a Real Mode segment to a PVAM segment. The maximum size of the transfer is limited to 65535 bytes, the length of a Real Mode segment.

➡ **Note**
This system call is used by DOS applications only; it is not supported in the iRMX OS.

Syntax, PL/M and C

```
CALL rqe$write$segment (realmode_ptr, pvam_seg, pvam_offset,
      size, status);
```

```
rqe_write_segment (realmode_ptr, pvam_seg, pvam_offset, size,
      status_ptr);
```

Parameter	PL/M Data Type	C Data Type
realmode_ptr	POINTER	void far *
pvam_seg	SELECTOR	SELECTOR
pvam_offset	WORD_32	UINT-32
size	WORD_16	UINT_16
status_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

realmode_ptr
A pointer to the Real Mode source segment.

pvam_seg
Specifies the PVAM destination segment. This must be a valid selector, such as the token for a data mailbox in an **rq_receive** Nucleus system call.

pvam_offset
The offset into the destination segment where the transfer starts.

size
Specifies the amount of data being transferred. If the size of the transfer is greater than the length of the destination segment, an exceptional condition code returns and no transfer takes place.

status_ptr
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The DOS application program calling sequence is shown here. See your compiler documentation for guidance in constructing an assembly language routine to do this:

1. Push the `realmode_ptr` parameter onto the stack.
2. Push the `pvam_seg` parameter onto the stack.
3. Push the `pvam_offset` parameter onto the stack.
4. Push the `size` parameter onto the stack.
5. Push the `status_ptr` parameter onto the stack.
6. Put the function code 31 into the AX CPU register.
7. Put the offset of the `status_ptr` parameter into the SI CPU register.
8. Cause a software interrupt number B8H.
9. Clear the stack.

Condition Codes

E_OK	0000H	Data transfer successful.
E_NUC_BAD_BUF	80E2H	Indicates one of these: • The `pvam_seg` does not refer to a valid segment. • The `pvam_offset` is outside the segment boundaries. • The specified size would cause the data transfer to exceed the pvam_seg limit.

rqe_set_vm86_extension

Installs and removes a Virtual 8086 Mode (VM86) extension at the specified interrupt level.

Syntax, PL/M and C

```
CALL rqe$set$vm86$extension (int_level, entry_ptr,
      deletion_handler_ptr, status_ptr);

rqe_set_vm86_extension (int_level, entry_ptr,
      deletion_handler_ptr, status_ptr);
```

Parameter	PL/M Data Type	C Data Type
`int_level`	BYTE	UINT_8
`entry_ptr`	POINTER	void (far *)(void)
`deletion_handler_ptr`	POINTER	void (far *)(void)
`status_ptr`	POINTER to WORD_16	UINT_16 far *

Parameters

`int_level`

Specifies the interrupt level at which the extension is installed. These interrupt levels are used; beware of conflicts from interrupt levels already in use in your system:

Level	Use
00H-10H	CPU traps and DOS hardware vectors
11H-20H	ROM BIOS services
21H-2FH	DOS services
38H-3FH	iRMX hardware vectors for master PIC
50H-57H	iRMX hardware vectors for slave PIC
5BH	Network redirector
80H	Used by the VM86 dispatcher
82H	Used then released by the TSR
85H	iRMX For Windows interface TSR
B8H	DOS Real-time Extensions (DOS RTE)
C3H	UDI
Others	Available for user extensions

`entry_ptr`

A pointer to the start of the VM86 Extension code. This pointer must reference a valid USE32 PVAM executable segment. The VM86 dispatcher calls the extension whenever a software interrupt at `int_level` occurs in VM86. A null pointer removes a previously installed extension at the specified level.

`deletion_handler_ptr`

A pointer to a deletion handler procedure that is called whenever a DOS program terminates or when DOS alone is restarted. This procedure cleans up the iRMX environment by removing any iRMX objects created by the VM86 Extension. This pointer must reference a valid USE32 PVAM executable segment. If a deletion handler is not necessary, use a null pointer.

`status_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

VM86 Dispatcher and VM86 Extension

The calling syntax for the VM86 Extension and Deletion Handler is described here. The VM86 Dispatcher calls the VM86 Extension as follows:

```
done = entry_procedure (state_ptr,flags);
```

Where:

`done` A byte returned to the VM86 Dispatcher by the VM86 Extension that indicates this:

Value	**Meaning**
0	The request needs further processing. The VM86 Dispatcher will reflect the interrupt into the real-mode interrupt handler. The real-mode interrupt handler will eventually return to the DOS application program.
0FFH	The request has been processed completely. The VM86 Dispatcher will restore all CPU registers from the dos_state structure, and then return to the DOS application program.

`state_ptr` A pointer to this structure. This structure must contain the current contents of the CPU registers indicated.

```
DECLARE dos_state STRUCTURE(
   edi                    WORD_32,
   esi                    WORD_32,
   ebp                    WORD_32,
   res1                   WORD_32,
   ebx                    WORD_32,
   edx                    WORD_32,
   ecx                    WORD_32,
   eax                    WORD_32,
   res2                   WORD_32,
   eip                    WORD_32,
   cs                     WORD_32,
   eflags                 WORD_32,
   esp                    WORD_32,
   ss                     WORD_32,
   es                     WORD_32,
   ds                     WORD_32,
   fs                     WORD_32,
   gs                     WORD_32);
```

or

```
typedef struct {
   UINT_32                edi;
   UINT_32                esi;
   UINT_32                ebp;
   UINT_32                res1;
   UINT_32                ebx;
   UINT_32                edx;
   UINT_32                ecx;
   UINT_32                eax;
   UINT_32                res2;
   UINT_32                eip;
   UINT_32                cs;
   UINT_32                eflags;
   UINT_32                esp;
   UINT_32                ss;
   UINT_32                es;
   UINT_32                ds;
   UINT_32                fs;
   UINT_32                gs;
} DOS_STATE_STRUCT;
```

`flags` Indicates this:

Bits	Value	Meaning
31-1		Reserved
0	0	The interrupt did not occur within the context of a DOS hardware interrupt handler.
	1	The interrupt occurred in the context of a DOS hardware interrupt handler.

The VM86 Dispatcher calls the Deletion Handler as follows:

```
deletion_handler (flags);
```

Where:

`flags` Indicates this:

Bits	Value	Meaning
31-1		Reserved
0	0	The current DOS program is being deleted.
	1	All DOS programs are being deleted. DOS is being restarted.

Additional Information

This system call can be made from any iRMX task running in Protected Mode.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_PARAM	8004H	At least one of these is true: • The `deletion_handler_ptr` parameter does not point to a valid USE32 PVAM executable segment. • The `entry_ptr` parameter does not point to a valid USE32 PVAM executable segment. • The int_level parameter does not specify a valid interrupt level.

rqe_dos_request

Makes DOS/ROM BIOS requests and other software interrupts handled by DOS applications.

Syntax, PL/M and C

```
CALL rqe$dos$request (register_ptr, wait_time, status_ptr)
```

```
rqe_dos_request (register_ptr, wait_time, status_ptr)
```

Parameter	PL/M Data Type	C Data Type
register_ptr	POINTER	DOS_DATA_STRUCT far *
wait_time	WORD_16	UINT_16
status_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

register_ptr

A pointer to this structure. This structure holds all the information DOS needs to execute the requested system call. This includes the values passed in the CPU registers, and additional information to allow data transfer to and from the iRMX OS. The iRMX for Windows TSR (**rmxtsr**) moves this data into the CPU registers and initiates the required DOS/ROM BIOS software interrupt. This system call can also return **rmxtsr** error messages.

```
DECLARE dos_data STRUCTURE(
   status                 WORD_16,
   flags                  WORD_16,
   int_num                BYTE,
   tsr_flags              BYTE,
   reg_al                 BYTE,
   reg_ah                 BYTE,
   reg_bl                 BYTE,
   reg_bh                 BYTE,
   reg_cl                 BYTE,
   reg_ch                 BYTE,
   reg_dl                 BYTE,
   reg_dh                 BYTE,
   reg_di                 WORD_16,
   reg_si                 WORD_16,
   reg_ds                 WORD_16,
   reg_es                 WORD_16,
   reg_bp                 WORD_16,
   xfer_data              BYTE,
   src1_xfer_pair         BYTE,
   src2_xfer_pair         BYTE,
   dest1_xfer_pair        BYTE,
   dest2_xfer_pair        BYTE,
   src_ptr_1              POINTER,
   src_count_1            WORD_16,
   src_ptr_2              POINTER,
   src_count_2            WORD_16,
   dest_ptr_1             POINTER,
   dest_count_1           WORD_16,
   dest_ptr_2             POINTER,
   dest_count_2           WORD_16)
```

or

```
struct DOSBYTEREGS {
   UINT_16               status;
   UINT_16               flags;
   UINT_8                int_num;
   UINT_8                tsr_flags;
   UINT_8                reg_al;
   UINT_8                reg_ah;
   UINT_8                reg_bl;
   UINT_8                reg_bh;
   UINT_8                reg_cl;
   UINT_8                reg_ch;
   UINT_8                reg_dl;
   UINT_8                reg_dh;
   UINT_16               reg_di;
   UINT_16               reg_si;
   UINT_16               reg_ds;
   UINT_16               reg_es;
   UINT_16               reg_bp;
   UINT_8                xfer_data;
   UINT_8                src1_xfer_pair;
   UINT_8                src2_xfer_pair;
   UINT_8                dest1_xfer_pair;
   UINT_8                dest2_xfer_pair;
   void far *            src_ptr_1;
   UINT_16               src_count_1;
   void far *            src_ptr_2;
   UINT_16               src_count_2;
   void far *            dest_ptr_1;
   UINT_16               dest_count_1;
   void far *            dest_ptr_2;
   UINT_16               dest_count_2;
} ;
```

```
struct DOSWORDREGS {
   UINT_16                status;
   UINT_16                flags;
   UINT_8                 int_num;
   UINT_8                 tsr_flags;
   UINT_16                reg_ax;
   UINT_16                reg_bx;
   UINT_16                reg_cx;
   UINT_16                reg_dx;
   UINT_16                reg_di;
   UINT_16                reg_si;
   UINT_16                reg_ds;
   UINT_16                reg_es;
   UINT_16                reg_bp;
   UINT_8                 xfer_data;
   UINT_8                 src1_xfer_pair;
   UINT_8                 src2_xfer_pair;
   UINT_8                 dest1_xfer_pair;
   UINT_8                 dest2_xfer_pair;
   void far *             src_ptr_1;
   UINT_16                src_count_1;
   void far *             src_ptr_2;
   UINT_16                src_count_2;
   void far *             dest_ptr_1;
   UINT_16                dest_count_1;
   void far *             dest_ptr_2;
   UINT_16                dest_count_2;
} ;

typedef union {
   struct                 DOSWORDREGS x;
   struct                 DOSBYTEREGS h;
} DOS_DATA_STRUCT;
```

Where:

`status` Indicates this:

Value	**Meaning**
0	The TSR was able to perform the request
not 0	The TSR was not able to perform the request.

`flags` The contents of the 16-bit CPU FLAGS register during DOS/ROM BIOS calls.

`int_num` The DOS/ROM BIOS interrupt number. These functions are not supported.

DOS Functions not Supported

Int	**Function/ Subfunction**		**Description**
21h	18		Reserved for DOS
	1D		Reserved for DOS
	1E		Reserved for DOS
	1F		Get default disk parameter block
	20		Reserved for DOS
	31		Terminate and stay resident
	32		Get drive parameter block
	34		Get address of IN-DOS flag
	37		Get/set switch character
	48		Allocate memory block
	4B	00	Execute program
		03	Load overlay
	4C		Terminate with return code
	4D		Get return code
	50		Set PSP
	52		Get disk list
	53		Translate PBP
	54		Get verify flag
	55		Create PSP
	5D	06	Get critical error flag address
	60		Reserved for DOS
	61		Reserved for DOS
	64		Reserved for DOS
27h			Terminate and stay resident
28h			Keyboard busy loop
29h			Fast put char
2Eh			Execute command

ROM BIOS Functions not Supported

Int	Function	Description
15h	00	Cassette
	01	Cassette
	02	Cassette
	03	Cassette
	0F	Format unit
	21	Error log
	4F	Keyboard intercept
	80	Device open
	81	Device close
	82	Program termination
	83	Event wait
	85	System-Request key pressed
	86	Wait
	87	Move data to/from protected mode memory
	89	Switch processor to protected mode
	90	device busy
	91	Interrupt complete
	C3	Enable/Disable watchdog timeout
	C4	Programmable Option (PS/2)
16h	00	Read keyboard character
	01	Read keyboard status
18h		ROM Basic
19h		System warm boot
1Ch		timer tick interrupt
1Dh		Video initialization data
1Eh		disk controller initialization data
1Fh		Graphics Bit-map table
70h		Real-Time clock

`tsr_flags` Indicates this:

Bits	Value	Meaning
7-4		Reserved
3	1	If int_num is a graphics function.
	0	Otherwise (if not a graphics function).
2-1		Reserved
0	1	Execute the requested function in the current DOS program, not switching to the TSR context.
	0	Execute the requested DOS/ROM BIOS call, switching to the context of the TSR. This is the typical value.

`reg_al` through `reg_bp`

CPU registers (corresponding to AL through BP) used to pass parameters for DOS/ROM BIOS requests. You must set `reg_ah`; you set any other registers as required by the DOS/ROM BIOS call being accessed, and the `xfer_data` field.

If there are pointer values required by the DOS/ROM BIOS, you do not need to transfer these values. The OS automatically sets the appropriate registers using transfer buffers in a reserved area of low memory.

`xfer_data` Indicates whether or not input or output data is associated with the request. It is possible to specify two complete data transfers, each with its own source and destination buffers. The combined maximum amount of data is 32 Kbytes.

Value	**Meaning**
0	The remaining fields and their contents are ignored.
0FFH	The remaining fields are valid. These values are set as required by the data transfer.

`src1_xfer_pair through dest2_xfer_pair`

Each of these specify which CPU register pair holds its associated data pointer. Initialize these pairs to 0, even if you are not using them.

Value	**Meaning**
0	No data is passed
1	DS:BX
2	DS:DX
3	DS:DI
4	DS:SI
5	DS:BP
6	ES:BX
7	ES:DX
8	ES:DI
9	ES:SI
10	ES:BP

The `_xfer_pair` parameters relate to the remaining fields as follows:

Register Pair	**Data Pointer**	**Byte Count**
src1_xfer_pair	src_ptr_1	src_count_1
src2_xfer_pair	src_ptr_2	src_count_2
dest1_xfer_pair	dest_ptr_1	dest_count_1;
dest2_xfer_pair	dest_ptr_2	dest_count_2.

`src_ptr_1` through `dest_ptr_2`

Pointers to source and destination buffers.

⇒ **Note**

For flat model applications only, treat the pointer parameters `src_ptr_1` through `dest_ptr_2` as two separate fields each in the structure. The first field has the name listed above and is a near pointer. The second field has the same name with `_seg` appended at the end. It is a segment selector for the pointer.

`src_count_1` through `dest_count_2`

Specifies the number of bytes transferred.

`wait_time`

Specifies the time the caller is willing to wait for the requested service to start.

Value	**Meaning**
0	Do not wait.
1-65534	Wait this number of clock intervals.
65535	Wait forever.

`status_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Because the register values in the `dos_data` structure are changed by the DOS/ROM BIOS call, the application must supply the values each time an **rqe_dos_request** is made.

See also: Making DOS and ROM BIOS Calls from iRMX Tasks, *Programming Concepts for DOS and Windows*

Condition Codes

E_OK 0000H No exceptional conditions occurred.

E_TIME 0001H The specified timeout occurred before the request could be started.

E_PARAM 8004H One of these:

- deletion_handler_ptr or entry_ptr do not point to a valid executable segment.
- int_level is not valid.
- One or more of the src*n*_xfer_pair, dest*n*_xfer_pair, src_ptr_*n*, dest_ptr_*n*, src_count_*n*, or dest_count_*n* parameters contains an invalid value.
- xfer_data < > 0 and all the src*n*_xfer_pair and dest*n*_xfer_pair parameters are set to 0.

RQEGetRmxStatus

Obtains the current status of the iRMX environment.

➠ **Note**
Use the syntax exactly as shown. Do not use underscores or dollar signs ($) in this system call.

Syntax, PL/M and C

```
Status = RQEGetRmxStatus;

Status = RQEGetRmxStatus();
```

Parameter	**PL/M Data Type**	**C Data Type**
`status`	WORD_32	UINT_32

Return Value

`status` Indicates the operational state of iRMX.

Additional Information

Issue this call from DOS applications before calling any other RTE primitive to ensure that RTE services are available. Unpredictable results will occur if RTE primitives are called when iRMX is not present.

Condition Codes

E_OK	0000H	iRMX OS is loaded and running.
E_EXIST	0006H	iRMX OS is not present (or unavailable).

❑ ❑ ❑

Kernel System Calls and Handlers 9

KN_create_alarm

Creates and starts a virtual alarm clock. You cannot make this call in a flat model application.

Syntax, PL/M and C

```
alarm = KN_create_alarm (area_ptr, handler_ptr, time_limit,
      flags);
```

```
alarm = KN_create_alarm (area_ptr, handler_ptr, time_limit,
      flags);
```

Parameter	**Kernel Data Type**
`alarm`	KN_TOKEN
`area_ptr`	UINT_32 far *
`handler_ptr`	void far *
`time_limit`	UINT_32
`flags`	KN_FLAGS

Return Value

`alarm` A token for the newly created alarm.

Parameters

`area_ptr`

A pointer to an area that holds the alarm's state. The area must be at least KN_ALARM_SIZE bytes long.

handler_ptr

A pointer to a procedure to be executed when the time period elapses. The mapping of the handler_ptr parameter to physical memory must remain constant until either the alarm is deleted or until a single-shot alarm handler is invoked. Write the entry point for an alarm interrupt handler:

```
alarm_handler (alarm_ptr);
```

Where:

alarm_ptr A pointer to the area holding the alarm state. If additional information is associated with the alarm, use this pointer to access it.

time_limit

Specifies the number of Kernel clock ticks that must elapse before invoking the handler:

Value	**Meaning**
0 or 1	The alarm handler is called on the next clock tick, and for repetitive alarms, on every clock tick. Only the remainder of the current clock tick elapses, not necessarily one full clock tick. (The value 0 is treated the same as 1.)
>1	The handler is called after (t - 1) + (remainder of current clock tick) ticks. If you set the value 5 and only half a tick currently remains, the alarm is called after 4-1/2 clock ticks.

flags Specifies the attributes of the alarm:

KN_ALARM_REPETITION_MASK

Specifies whether the alarm generates a single interrupt or repeated interrupts. Choose one of these literals:

Literal	**Meaning**
KN_SINGLE_SHOT	The alarm object generates a single interrupt. This alarm becomes inactive after its initial time interval elapses, and its memory becomes available for reuse.
KN_REPEATER	The alarm object generates repeated interrupts. This alarm resets after each invocation of the handler so that the handler is called again after the next interval elapses. Repetitive alarms generate periodic interrupts until you explicitly delete them.

KN_HANDLER_CONVENTION_MASK

Use this literal:

Literal	**Meaning**
KN_CALL_FAR	The alarm handler is in a different subsystem than the Kernel code and, therefore, must make a far call to it. This flag must be set.

Additional Information

Always specify a time limit and a handler. When the time limit elapses, the Kernel invokes the handler, thereby simulating a timer interrupt. When the alarm handler is invoked, interrupts are disabled and scheduling is locked. Since this call is non-scheduling, it is safe for use by interrupt handlers.

See also: Kernel time management, *System Concepts*

KN_create_area

Allocates an area of memory of the specified size from the specified memory pool.

Syntax, PL/M and C

```
area = KN_create_area (pool, size);
```

```
area = KN_create_area (pool, size);
```

Parameter	**Kernel Data Type**
`area`	void *
`pool`	KN_TOKEN
`size`	UINT_32

Return Value

`area` A pointer to an area of the desired size. If no area can be allocated, the Kernel returns a null pointer.

Parameters

`pool` A token for the memory pool from which the area is allocated. This is the token returned from a **KN_create_pool** system call.

`size` Specifies the size of the requested area in bytes. This value can range from KN_MINIMUM_AREA_SIZE to the `pool_largest` value returned by the **KN_get_pool_attributes** system call. If you specify a value smaller than KN_MINIMUM_AREA_SIZE, the Kernel rounds the request upward to the minimum size.

Additional Information

If the memory pool was created from memory aligned on a 4-byte boundary, the area assigned with this system call will also be aligned on a 4-byte boundary. If there is insufficient contiguous memory in the pool to satisfy the request, a null pointer is returned.

To allocate an area of size X, an available area of size

```
    X + KN_AREA_OVERHEAD
```

must exist within the pool. KN_AREA_OVERHEAD is the number of bytes of overhead associated with each area allocated from the pool.

Note
This call is blocking; use it with caution in interrupt handlers.

See also: Kernel memory management, pool and area overhead, *System Concepts*

KN_create_mailbox

Creates a mailbox in a specified area of memory.

Syntax, PL/M and C

```
mailbox = KN_create_mailbox (area_ptr, message_size,
      queue_size, flags);
```

```
mailbox = KN_create_mailbox (area_ptr, message_size,
      queue_size, flags);
```

Parameter	Kernel Data Type
`mailbox`	KN_TOKEN
`area_ptr`	UINT_32 far *
`message_size`	UINT_32
`queue_size`	UINT_32
`flags`	KN_FLAGS

Return Value

`mailbox`

A token for the newly created mailbox.

Parameters

`area_ptr`

A pointer to the area where the mailbox is created. For better performance, align this area on a 4-byte boundary. The size of this area must be:

KN_MAILBOX_SIZE + (message_size + KN_MAILBOX_MSG_OVERHEAD) * queue_size

Literal	Meaning
KN_MAILBOX_SIZE	The number of bytes required for a mailbox object, excluding the message queue.
KN_MAILBOX_MSG _OVERHEAD	The number of bytes of overhead for each message in the message queue of a mailbox.

`message_size`

Specifies the maximum size in bytes of the messages to be exchanged through this mailbox. Never send messages larger than the maximum message size specified for the mailbox; if you do, the results are unpredictable. Keep messages as small as possible. Transferring large messages can degrade the interrupt latency of the system.

queue_size

Specifies the maximum number of messages that can be stored in the mailbox. Add 1 to queue_size to specify that 1 of the slots in the mailbox queue is reserved for a high-priority message. The reserved slot ensures that at least 1 high-priority message is accepted even when the mailbox queue is full. If the message queue is full when a high-priority message arrives, the Kernel puts the high-priority message into the reserved slot instead. If that reserved slot is also taken, an E_LIMIT_EXCEEDED exception is returned. This is the same exception code that is returned when a non-priority message cannot be sent because the mailbox queue is full.

If you set KN_RESERVE_PRIORITY_DATA, then 1 is automatically taken away from queue_size. When the Kernel assigns messages to the mailbox, it assigns them in a circular fashion, assuming that the number of message slots is equal to queue_size and the size of each message is equal to message_size.

Even if the number of messages queued at the mailbox never reaches queue_size, the circular queuing means that all the memory allocated for messages will be accessed at one time or another. The amount of memory you assign to the mailbox must match the values you specify for message_size and queue_size.

flags Specifies the type of mailbox to be created.

KN_EXCH_TYPE_MASK

Specifies whether the mailbox uses FIFO or Priority queueing. Choose one of these literals:

Literal	**Meaning**
KN_FIFO_QUEUEING	Tasks are queued in the order that they arrive at the mailbox.
KN_PRIORITY_QUEUEING	Tasks are queued based on their task priority.

KN_RESERVE_PRIORITY_DATA_MASK

Specifies whether the mailbox queue has a slot reserved for a high-priority message. Choose one of these literals:

Literal	**Meaning**
KN_DONT_RESERVE_PRIORITY_DATA	Do not reserve a slot for a high-priority message
KN_RESERVE_PRIORITY_DATA	Reserve a slot for a high-priority message

Additional Information

The Kernel attempts to place high-priority messages ahead of all other messages in the regular queue. If the message queue is full, the Kernel puts the high-priority message into the reserved slot (if you specified 1).

The purpose of the reserved slot is to ensure at least 1 high-priority message is accepted even when the mailbox queue is full.

This call is non-scheduling and is safe for use by interrupt handlers.

KN_create_pool

Creates a memory pool in a specified range of memory.

Syntax, PL/M and C

```
pool = KN_create_pool (pool_ptr, size);
```

```
pool = KN_create_pool (pool_ptr, size);
```

Parameter	**Kernel Data Type**
`pool`	KN_TOKEN
`pool_ptr`	void far *
`size`	UINT_32

Return Value

`pool` A token for the newly created memory pool.

Parameters

`pool_ptr`

A pointer to the first location in memory to be included in the new memory pool.

`size` Specifies the number of bytes to include in the new memory pool.

To determine the total number of bytes, consider the number and size of each area that could conceivably be allocated at the same time. For many applications, all areas allocated from a memory pool are of the same size. Therefore, to create a pool that can exactly allocate N areas all of size M, an area of this size is required for the pool. M must be greater than or equal to KN_MINIMUM_AREA_SIZE:

N * (M + KN_AREA_OVERHEAD) + KN_POOL_OVERHEAD

Literal	Meaning
KN_AREA_OVERHEAD	The number of bytes of overhead associated with each area allocated from the pool.
KN_POOL_OVERHEAD	The number of bytes of overhead in a new pool. For a pool of X bytes, request a pool of X + KN_POOL_OVERHEAD using the **create_pool** system call. The smallest pool size is: KN_MINIMUM_POOL_SIZE + KN_POOL_OVERHEAD
KN_MINIMUM_POOL_SIZE	The minimum number of bytes necessary for a pool object
KN_MINIMUM_AREA_SIZE	The smallest area which can be allocated from a memory pool.

Additional Information

Only access the memory pool with the **KN_create_area** system call. If the memory used to contain the pool is aligned on a 4-byte boundary, all areas allocated from the pool are also aligned on 4-byte boundaries.

Provide the memory area for the pool by either declaring it as a program variable or by allocating it using **rq_create_segment**. Using this system call will also ensure that memory is 4-byte aligned. This call is non-scheduling and is safe for use by interrupt handlers.

See also: Kernel memory management, pool and area overhead, *System Concepts*

KN_create_semaphore

Creates 1 of 3 kinds of semaphores with 0 or 1 initial units.

Syntax, PL/M and C

```
semaphore = KN_create_semaphore (area_ptr, flags);
```

```
semaphore = KN_create_semaphore (area_ptr, flags);
```

Parameter	**Kernel Data Type**
`semaphore`	KN_TOKEN
`area_ptr`	UINT_32 far *
`flags`	KN_FLAGS

Return Value

`semaphore`

A token for the newly created semaphore.

Parameters

`area_ptr`

A pointer to the area where the semaphore is to be created. This area must be at least KN_SEMAPHORE_SIZE bytes long. For better performance, align the area on a 4-byte boundary.

`flags` Specifies the attributes of the semaphore:

KN_EXCH_TYPE_MASK

Specifies the type of semaphore. Choose one of these literals:

Literal	**Meaning**
KN_FIFO_QUEUEING	The semaphore uses FIFO queueing.
KN_PRIORITY_QUEUEING	The semaphore uses priority queueing.
KN_REGION	The exchange is a 1 (or single) unit region.

KN_INITIAL_SEM_STATE_MASK

Specifies the number of initial units the semaphore receives. Choose one of these literals:

Literal	**Meaning**
KN_ZERO_UNITS	The semaphore is created with no units.
KN_ONE_UNIT	The semaphore is created with 1 unit.

Additional Information

FIFO and priority semaphores can contain as many as 65,535 units, which are placed in the semaphore by using multiple **KN_send_unit** calls, 1 for each unit.

If a region is created with 0 units, the creating task holds the region's unit and is therefore the owning task. If a region is created with 1 unit, no task owns the region until it invokes **KN_receive_unit** for that region. This call is non-scheduling and is safe for use by interrupt handlers.

create_task_handler

Creates a task. You cannot write this handler in a flat model application.

Syntax, C

```
void create_task_handler (task_ptr);
```

Parameter	**Kernel Data Type**
`task_ptr`	KN_TASK_STATE far *

Parameter

`task_ptr`

A pointer to the area containing the state of the new task. This area can be dereferenced using the structure KN_TASK_STATE. Do not change this structure.

See also: **create_task** in this manual,
KN_TASK_STATE structure in Chapter 1

Additional Information

The **create_task_handler** is a user-supplied procedure that the Kernel invokes whenever it creates a task. During task creation, the Kernel invokes **create_task_handler** after it initializes the new task but before the task is allowed to execute. The handler will typically perform additional initialization to any additional task state maintained by the application.

Set up the **create_task_handler** using the **KN_set_handler** system call.

Task creation handlers are invoked with interrupts disabled and scheduling locked.

See also: **KN_set_handler**

KN_delete_alarm

Deletes a previously created alarm. You cannot make this call in a flat model application.

Syntax, PL/M and C

```
CALL KN_delete_alarm (alarm);
```

```
void KN_delete_alarm (alarm);
```

Parameter	**Kernel Data Type**
`alarm`	KN_TOKEN

Parameters

`alarm` A token for the alarm to be deleted.

Additional Information

As a result of this call, the handler associated with the alarm will not be invoked. The area occupied by the alarm is available for reuse.

Single-shot alarms are detected when they are invoked; it is acceptable to delete these alarms even if they have already been deleted when they executed. This prevents race conditions in which task execution speed is responsible for error conditions.

⟹ **Note**
Since the Kernel does not perform parameter validation, do not delete an alarm that has not yet been created.

See also: Kernel time management, *System Concepts*

KN_delete_area

Returns an area of memory to the memory pool from which it was allocated.

Syntax, PL/M and C

```
CALL KN_delete_area (area, pool);
```

```
void KN_delete_area (area, pool);
```

Parameter	Kernel Data Type
`area`	void far *
`pool`	KN_TOKEN

Parameters

`area` A pointer to the area to be deleted.

`pool` A token for the memory pool from which the area was allocated.

Additional Information

After this call, the memory assigned to the mailbox is available for reuse, and should no longer be accessed directly by the application.

Note
This call is blocking and is unsafe for use by interrupt handlers.

KN_delete_mailbox

Deletes the specified mailbox.

Syntax, PL/M and C

```
CALL KN_delete_mailbox (mailbox);
```

```
void KN_delete_mailbox (mailbox);
```

Parameter	**Kernel Data Type**
`mailbox`	KN_TOKEN

Parameters

`mailbox`

A token for the mailbox to be deleted.

Additional Information

All tasks waiting at the mailbox are awakened and given an E_NONEXIST status, and all messages queued at the mailbox are lost. After this call, the memory assigned to the mailbox is available for reuse.

➾ **Note**
This is a signaling call. Use the **KN_stop_scheduling** system call in interrupt handlers.

See also: **KN_stop_scheduling**

KN_delete_pool

Deletes a memory pool.

Syntax, PL/M and C

```
CALL KN_delete_pool (pool);
```

```
void KN_delete_pool (pool);
```

Parameter	**Kernel Data Type**
`pool`	KN_TOKEN

Parameters

`pool`

A KN_TOKEN for the memory pool to be deleted.

Additional Information

This system call makes the entire address range of the memory pool available for reuse. Do not invoke any system call that uses the pool (such as **KN_create_area** and **KN_delete area**) after the pool has been deleted.

Memory pools can be deleted even if the tasks currently have access to areas of memory allocated from those pools. The tasks accessing the areas will still have access to them. However, the Kernel does not prevent other tasks from accessing these in-use areas after the pool is deleted.

This call is non-scheduling and is safe for use by interrupt handlers.

KN_delete_semaphore

Deletes the specified semaphore.

Syntax, PL/M and C

```
CALL KN_delete_semaphore (semaphore);
```

```
void KN_delete_semaphore (semaphore);
```

Parameter	**Kernel Data Type**
`semaphore`	KN_TOKEN

Parameters

`semaphore`

A token for the semaphore to be deleted.

Additional Information

All tasks waiting at the semaphore are awakened with the E_NONEXIST status code.

Do not delete a region semaphore while a task has access to the region or the task is no longer guarded by a region. Any dynamic adjustments that were made to that task's priority as a result of accessing the region are nullified, the task resumes its static priority, and may be preempted. Because the region no longer exists, the task must not send the region's unit back to the region.

⇒ **Note**
This is a signaling call. Use **KN_stop_scheduling** in interrupt handlers.

See also: Kernel semaphores, *System Concepts*

delete_task_handler

The Kernel invokes this procedure when it deletes a task. You cannot write this handler in a flat model application.

Syntax, C

```
void delete_task_handler (task_ptr);
```

Parameter	**Data Type**
`task_ptr`	KN_TASK_STATE far *

Parameter

`task_ptr`

A pointer to the area containing the state of the task to be deleted. This area can be dereferenced using the structure KN_TASK_STATE. Do not change this structure.

See also: **create_task_handler, create_task**, **delete_task**
KN_TASK_STATE structure in Chapter 1

Additional Information

The **delete_task_handler** is a user-supplied procedure that the Kernel invokes whenever it deletes a task.

Se up the **delete_task_handler** using the **KN_set_handler** system call. The Kernel invokes the task deletion handler after the task is removed from any scheduling queues (to prevent it from executing), but before the task state is destroyed. The deletion handler should perform additional task cleanup to any additional task state maintained by the application.

Task deletion handlers are invoked with interrupts disabled and with scheduling locked.

See also: **KN_set_handler**

KN_get_pool_attributes

Provides information about the specified memory pool.

Syntax, PL/M and C

```
CALL KN_get_pool_attributes (pool, attributes_ptr);
```

```
void KN_get_pool_attributes (pool, attributes_ptr);
```

Parameter	Kernel Data Type
pool	KN_TOKEN
attributes_ptr	KN_POOL_ATTRIBUTES_STRUC far *

Parameters

pool A token for the memory pool whose attributes are requested.

attributes_ptr

A pointer to KN_POOL_ATTRIBUTES_STRUC where the Kernel returns the attributes of the specified memory pool. This is the format of this structure:

```
typedef struct {
   UINT_32                pool_size;
   UINT_32                pool_available;
   UINT_32                pool_largest;
} KN_POOL_ATTRIBUTES_STRUC;
```

Where:

pool_size The total number of bytes in the memory pool; the size of the memory supplied when the memory pool was created.

pool_available

The total number of bytes of available space in the memory pool.

pool_largest

The number of bytes in the largest contiguous available space in the memory pool.

Additional Information

The memory pool must previously be established with the **KN_create_pool** system call. This call is non-scheduling and is safe for use by interrupt handlers.

KN_get_time

Returns the current value of the counter that the Kernel uses to keep track of the number of clock ticks that have occurred.

Syntax, PL/M and C

```
time = KN_get_time ();
```

```
time = KN_get_time ();
```

Parameter	**Kernel Data Type**
`time`	UINT_64

Parameters

`time` Contains the current value of the system clock.

Additional Information

The Kernel defines the UINT 64 type as a long integer type for use in some system calls. Write modules that use these system calls in PL/M or Assembly language. Keep 64-bit operations isolated in a separate module where the `long64` switch is enabled. For C applications where the compiler does not support 64-bit data types, use the **KNE_get_time** call.

When the Kernel is initialized, the count is set to 0. You can set the count to any value with the **KN_set_time** system call.

This call is non-scheduling and is safe for use by interrupt handlers.

See also: Kernel time management, *System Concepts*

KNE_get_time

Returns the current value of the counter that the Kernel uses to keep track of the number of clock ticks that have occurred. Unlike **KN_get_time**, the value is returned in a structure that allows use of 32-bit data types.

Syntax, PL/M and C

```
CALL KNE_get_time (time_struct);

KNE_get_time (time_struct);
```

Parameter	**Kernel Data Type**
`time_struct`	KN_TIME_STRUCT far *

Parameters

`time_struct`

A pointer to the following structure that contains the current value of the system clock.

```
typedef struct {
   UINT_32               lo;
   UINT_32               hi;
   } KN_TIME_STRUCT;
```

Where:

`lo`	Specifies the lower 32-bits of the 64-bit time value kept by the kernel.
`hi`	Specifies the upper 32-bits of the 64-bit time value kept by the kernel.

Additional Information

When the Kernel is initialized, the count is set to 0. You can set the count to any value with the **KNE_set_time** system call.

This call is non-scheduling and is safe for use by interrupt handlers.

See also: Kernel time management, *System Concepts*

KN_receive_data

Requests a message from the specified mailbox.

Syntax, PL/M and C

```
status = KN_receive_data (mailbox, data_ptr, length_ptr,
      time_limit);
```

```
status = KN_receive_data (mailbox, data_ptr, length_ptr,
      time_limit);
```

Parameter	Kernel Data Type
status	KN_STATUS
mailbox	KN_TOKEN
data_ptr	void far *
length_ptr	UINT_32 far *
time_limit	UINT_32

Return Value

status Indicates the result of the call. Values are:

Literal	Meaning
E_OK	The task received a message.
E_TIME_OUT	The time limit expired.
E_NONEXIST	The mailbox was deleted while the task was waiting.

➨ **Note**

If the mailbox is deleted before the task begins waiting, the call will not return the E_NONEXIST message. Do not delete the mailbox before a task begins waiting.

Parameters

`mailbox`

A token for the mailbox from which the message is requested.

`data_ptr`

A pointer to an area where the message is placed. The area size must be equal to the `message_size` parameter specified when the mailbox was created.

`length_ptr`

A pointer to where the Kernel specifies the length (in bytes) of the message it returns.

`time_limit`

Specifies the number of clock ticks the caller is willing to wait for a message. Values are:

Literal	**Meaning**
KN_DONT_WAIT	The task will not wait at all.
KN_WAIT_FOREVER	The task is willing to wait indefinitely.
UINT_32 value	The task will wait for the specified number of clock ticks.

Additional Information

If the mailbox currently contains at least 1 message, the oldest message or the latest high-priority message is removed from the message queue and returned to the calling task. If there are no messages queued at the mailbox and the task is willing to wait, it is put to sleep and queued at the mailbox for the amount of time it is willing to wait. The task is queued at the mailbox in either FIFO or priority-based order, depending on the type of mailbox. The task will be awakened by 1 of 3 events:

- The task is at the head of the mailbox queue and another task invokes **KN_send_data** on the mailbox.
- The number of clock ticks specified by the task expires.
- The mailbox is deleted.

Note

When receiving (using the **KN_receive_data** system call) and sending (using the **KN_send_data** system call) mailbox messages, interrupts are disabled for the time it takes to copy the message. A large data transfer using mailboxes may affect interrupt latency.

This is a blocking call; use it with caution in interrupt handlers.

KN_receive_unit

Requests a unit from the specified semaphore.

Syntax, PL/M and C

```
status = KN_receive_unit (semaphore, time_limit);

status = KN_receive_unit (semaphore, time_limit);
```

Parameter	Kernel Data Type
`status`	KN_STATUS
`semaphore`	KN_TOKEN
`time_limit`	UINT_32

Return Value

`status` Indicates the result of the call. Values are:

Literal	Meaning
E_OK	The task received the requested unit.
E_TIME_OUT	The time limit expired.
E_NONEXIST	The semaphore was deleted while the task was waiting.

Note
If the semaphore is deleted before the task begins waiting, the call will not return the E_NONEXIST message. Do not delete the semaphore before a task begins waiting.

Parameters

`semaphore`

A token for the semaphore from which a unit is requested.

`time_limit`

Specifies the number of clock ticks the calling task is willing to wait for the unit. Choose one of these literals (or enter a value):

Literal	Meaning
KN_DONT_WAIT	The task will not wait at all.
KN_WAIT_FOREVER	The task is willing to wait indefinitely.
UINT_32 value	The task will wait for the specified number of clock ticks.

Additional Information

If the semaphore currently contains units, the number of units is reduced by 1 and the task proceeds. If the semaphore has no units and the task is willing to wait, the task is put to sleep and placed into the semaphore's task queue. The task will be awakened by 1 of 3 events:

- The task is at the head of the semaphore queue and another task invokes **KN_send_unit** on this semaphore.
- The number of clock ticks specified by the task expires.
- The semaphore is deleted.

Note
This is a blocking call; use it with caution in interrupt handlers.

KN_reset_alarm

Returns a previously created alarm to its creation state. You cannot make this call in a flat model application.

Syntax, PL/M and C

```
CALL KN_reset_alarm (alarm);
```

```
void KN_reset_alarm (alarm);
```

Parameter	**Kernel Data Type**
`alarm`	KN_TOKEN

Parameters

`alarm` A token for the alarm to be reset.

Additional Information

This operation is equivalent to invoking the **KN_delete_alarm** system call, then invoking the **KN_create_alarm** system call.

Because **KN_reset_alarm** may be invoked on single-shot alarms even if the alarm has gone off, it is not necessary to synchronize between an alarm reset and the expiration of the alarm time interval. This call is non-scheduling and is safe for use by interrupt handlers.

See also: Kernel time management, *System Concepts*

KN_reset_handler

Dynamically removes an application-supplied task handler. You cannot make this call in a flat model application.

Syntax, PL/M and C

```
CALL KN_reset_handler (hdlr_area);
```

```
void KN_reset_handler (hdlr_area);
```

Parameter	**Kernel Data Type**
`hdlr_area`	KN_HDLR_STRUC far *

Parameters

`hdlr_area`

A pointer to a KN_HDLR_STRUC that sets and resets the task creation, task deletion, task switch, and task change priority handlers dynamically.

See also: **KN_set_handler** system call for the format of this structure

Additional Information

This call resets the handler previously set by the **KN_set_handler** system call. This call is non-scheduling and is safe for use by interrupt handlers.

KN_send_data

Sends a message to the specified mailbox.

Syntax, PL/M and C

```
status = KN_send_data (mailbox, data_ptr, length):
```

```
status = KN_send_data (mailbox, data_ptr, length):
```

Parameter	Kernel Data Type
`status`	KN_STATUS
`mailbox`	KN_TOKEN
`data_ptr`	void far *
`length`	UINT_32

Return Value

`status` Indicates the result of the call. Values are:

Literal	Meaning
E_OK	The mailbox accepted the message.
E_LIMIT_EXCEEDED	The message was rejected because the mailbox was full.

Parameters

`mailbox`

A token for the mailbox where the message is sent.

`data_ptr`

A pointer to an area containing the message to be sent.

`length`

Specifies the number of bytes in the message to be sent. Its maximum allowable value is the maximum message size specified when the mailbox was created.

Additional Information

If a task is waiting at the mailbox, it receives the message; otherwise, the message is queued. If the mailbox is full, an exception returns. When receiving (using the **KN_receive_data** system call) and sending (using the **KN_send_data** system call) mailbox messages, interrupts are disabled for the time it takes to copy the message. A large data transfer using mailboxes may affect interrupt latency.

➠ **Note**
Since this is a signaling call, call **KN_stop_scheduling**.

See also: **KN_create_mailbox**

KN_send_priority_data

Sends a high-priority message to the specified mailbox and places it at the head of the queue.

Syntax, PL/M and C

```
status = KN_send_priority_data (mailbox, data_ptr, length);
```

```
status = KN_send_priority_data (mailbox, data_ptr, length);
```

Parameter	**Kernel Data Type**
`status`	KN_STATUS
`mailbox`	KN_TOKEN
`data_ptr`	void far *
`length`	UINT_32

Return Value

`status`

Indicates the result of the call. Values are:

Literal	**Meaning**
E_OK	The mailbox accepted the message.
E_LIMIT_EXCEEDED	The message was rejected because the mailbox was full.

Parameters

`mailbox`

A token for the mailbox where the message is sent.

`data_ptr`

A pointer to an area containing the message to be sent.

`length`

The number of bytes in the message to be sent. This value can be no greater than the maximum message size specified when the mailbox was created.

Additional Information

If a task is waiting at the mailbox, it receives the message; otherwise, the message is queued. If the mailbox is full, an exception returns.

Mailboxes normally store messages in a FIFO queue. A series of **KN_send_priority_data** calls results in messages being queued in LIFO order.

When you create a mailbox with **KN_create_mailbox**, you can specify 1 of the slots in its queue as reserved for a high-priority message. **KN_send_priority_data** can then use that slot.

See also: **KN_create_mailbox**

Note

This is a signaling call; use **KN_stop_scheduling**.

KN_send_unit

Adds a unit to a specified semaphore.

Syntax, PL/M and C

```
CALL KN_send_unit (semaphore);
```

```
void KN_send_unit (semaphore);
```

Parameter	Kernel Data Type
`semaphore`	KN_TOKEN

Parameters

`semaphore`

A token for the semaphore where the unit is sent.

Additional Information

If tasks are waiting at the semaphore, the task at the head of the queue is awakened and given the unit.

If **KN_send_unit** is invoked on a semaphore that contains the maximum of 65,535 units, the number of units in the semaphore is not incremented, and the results will be unpredictable.

Note

This is a signaling call; call **KN_stop_scheduling**.

KN_set_handler

Dynamically installs a user-supplied task handler. You cannot make this call in a flat model application.

Syntax, PL/M and C

```
CALL KN_set_handler (hdlr_area);
```

```
void KN_set_handler (hdlr_area);
```

Parameter	Kernel Data Type
hdlr_area	KN_HDLR_STRUC far *

Parameters

hdlr_area

A pointer to a KN_HDLR_STRUC that sets and resets the task creation, task deletion, and task switch handlers dynamically. Its format is:

```
typedef struct {
   UINT_32                reserved[2];
   KN_FLAGS               hdlr_flags;
   void *                 hdlr_ptr;
   KN_HDLR_TYPE           hdlr_type;
   UINT_8                 hdlr_res[3];
} KN_HDLR_STRUC;
```

Where:

reserved Do not use.

hdlr_flags

Use this literal:

Literal	Meaning
KN_CALL_FAR	Should be set.

hdlr_ptr A pointer to the task handler.

hdlr_type A KN_HDLR_TYPE. Choose one:

KN_TASK_CREATION_HANDLER
KN_TASK_DELETION_HANDLER
KN_TASK_SWITCH_HANDLER

hdlr_res Do not use.

Note

Preserve this structure until the associated handler is reset using the **KN_reset_handler** system call. Include the structure passed to the handlers, but do not reuse the handler structure.

Additional Information

You can install multiple task handlers for creation, deletion, and task switching by invoking **KN_set_handler** multiple times. This call is non-scheduling and is safe for use by interrupt handlers.

See also: Kernel task handlers, *System Concepts*

KN_set_time

Sets the value of the counter that the Kernel uses to keep track of the number of clock ticks that have occurred.

Syntax, PL/M and C

```
CALL KN_set_time (time);

void KN_set_time (time);
```

Parameter	**Kernel Data Type**
`time`	UINT_64

Parameters

`time` Specifies the new value of the system clock.

Additional Information

The Kernel defines the UINT_64 type as a long integer type for use in some system calls. Write modules that use these system calls in PL/M or Assembly language. Keep 64-bit operations isolated in a separate module where the `long64` switch is enabled. For C applications where the compiler does not support 64-bit data types, use the **KNE_set_time** call.

When the Kernel is initialized, the count is set to 0. You can determine the current value of the clock by calling the **KN_get_time** system call.

This call is non-scheduling and is safe for use by interrupt handlers.

KNE_set_time

Sets the value of the counter that the Kernel uses to keep track of the number of clock ticks that have occurred. Unlike **KN_set_time**, you set the value in a structure that allows use of 32-bit data types.

Syntax, PL/M and C

```
CALL KNE_set_time (time_struct);
```

```
void KNE_set_time (time_struct);
```

Parameter	**Kernel Data Type**
`time_struct`	KN_TIME_STRUCT far *

Parameters

`time_struct`

A pointer to the following structure that contains a value for the system clock.

```
typedef struct {
   UINT_32              lo;
   UINT_32              hi;
} KN_TIME_STRUCT;
```

Where:

`lo` Specifies the lower 32-bits of the 64-bit time value kept by the Kernel.

`hi` Specifies the upper 32-bits of the 64-bit time value kept by the Kernel.

Additional Information

When the Kernel is initialized, the count is set to 0. You can determine the current value of the clock with the **KNE_get_time** system call.

This call is non-scheduling and is safe for use by interrupt handlers.

See also: Kernel time management, *System Concepts*

KN_sleep

Puts the calling task to sleep for the specified number of clock ticks.

Syntax, PL/M and C

```
CALL KN_sleep (time_limit);
```

```
void KN_sleep (time_limit);
```

Parameter	**Kernel Data Type**
`time_limit`	UINT_32

Parameter

`time_limit`

Specifies the number of clock ticks for which the task is to sleep, or one of these literals:

Literal	**Meaning**
KN_DONT_WAIT	The task will not wait at all.
KN_WAIT_FOREVER	The task is willing to wait indefinitely.

KN_DONT_WAIT does not cause the running task to go to sleep. It has an effect only if there are other ready tasks of equal priority. In that case, the running task is made ready and put in the ready queue after all other ready tasks of equal priority. If there are no other ready tasks of equal priority, the current task remains running.

KN_WAIT_FOREVER causes the task to sleep forever. This effectively deletes the task but the task's memory is not released.

Additional Information

Note

This is a rescheduling call and is unsafe for use by interrupt handlers.

KN_start_scheduling

Cancels one scheduling lock imposed by **KN_stop_scheduling**.

Syntax, PL/M and C

```
CALL KN_start_scheduling ();
```

```
void KN_start_scheduling ();
```

Additional Information

If the lock that is canceled is the last outstanding scheduling lock, all task state transitions that were temporarily delayed are carried out, and the highest priority ready task begins executing.

➠ **Note**
This call is in the signaling scheduling category. Call **KN_stop_scheduling** in the interrupt handlers.

The Kernel sometimes stops scheduling internally, so that scheduling might not restart immediately even though the application has canceled all the scheduling locks that it established.

If **KN_start_scheduling** is invoked when scheduling is not stopped, the results are undefined.

KN_stop_scheduling

Temporarily locks the scheduling mechanism or places an additional lock on the mechanism for the running task.

Syntax, PL/M and C

```
CALL KN_stop_scheduling ();
```

```
void KN_stop_scheduling ();
```

Additional Information

Any task state transitions that would move the task from the running state to the ready state are delayed until scheduling is resumed. For example, with scheduling stopped, if the running task sends a message to a mailbox at which a higher-priority task is waiting, that waiting task becomes ready, but it would not become the running task until scheduling is resumed.

The **KN_stop_scheduling** system call does not necessarily halt task switching. If the running task invokes a blocking system call (such as **KN_receive_data** or **KN_sleep**) while scheduling is stopped, the task enters the asleep or suspended state immediately and the highest priority ready task becomes the running task. The new task is restored with all its scheduling locks in place. When the first task is again restored to the running state, its scheduling locks are also restored to the level they were at the time of the block.

You can invoke **stop_scheduling** repeatedly when scheduling is locked. Scheduling is resumed only when all scheduling locks are canceled. This call is non-scheduling and is safe for use by interrupt handlers.

task_switch_handler

This procedure executes whenever a task switch occurs. You cannot write this handler in a flat model application.

Syntax, PL/M and C

```
void task_switch_handler (new_task_ptr);
```

Parameter	**Data Type**
`*new_task_ptr`	void

Parameter

`new_task_ptr`

A pointer to the area containing the state of the task that will be the next running task. Part of this area can be dereferenced using the structure KN_TASK_STATE. Do not change this structure.

See also: KN_TASK_STATE structure, in Chapter 1

Additional Information

The **task_switch_handler** is a user-supplied procedure that the Kernel invokes whenever a task switch occurs. You can set it up using the **KN_set_handler** system call.

Whenever the Kernel switches the running task, it invokes the task switch handler. The handler is invoked in the context of the old task (the task giving up the processor). A pointer to the new running task is supplied as a parameter to the handler.

Task switch handlers are invoked with interrupts disabled and with scheduling locked.

See also: **KN_set_handler**

□ □ □

Virtual Memory System Calls 10

rqv_allocate

Allocates physical memory to a virtual segment.

Syntax, PL/M and C

```
offset = rqv$allocate (vseg, size, except_ptr);

offset = rqv_allocate (vseg, size, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`offset`	POINTER	void near *
`vseg`	TOKEN	SELECTOR
`size`	DWORD	UINT_32
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`offset` A near pointer to the allocated physical memory within the virtual segment.

Parameters

`vseg` A token for the virtual segment. If this parameter is null and the application is flat model, the parameter indicates the application's virtual segment. For segmented model applications, a null value is an error.

`size` The amount, in bytes, of contiguous physical memory to be allocated.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This call is primarily used in flat model application programs. The calling task must belong to the same job that created the virtual segment. The call automatically rounds up `size` (in bytes) to a multiple of 4K. The allocated pages are contiguous; they start and end on 4K boundaries.

The virtual segment manager finds an available space within the virtual segment and returns a near pointer to the allocated physical memory. The call fails if `size` bytes of contiguous physical memory are not available, if `size` exceeds the segment size, or if there is not enough virtual address space available in the virtual segment. The memory required for page tables is charged to the calling job's memory pool. The first allocation to a virtual segment incurs a 4K (minimum) overhead for at least one page table.

If `vseg` is a null selector (0) and the application is flat model, the application's virtual segment is assumed; otherwise, a null selector is an error.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_EXIST	0006H	The `vseg` parameter represents a segment that is being deleted, or `vseg` is a null token and the caller is not a flat model application
E_MEM	0002H	There is insufficient physical memory available to satisfy this request.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The `size` parameter is larger than the virtual segment or is zero.
E_SLOT	000CH	There is no room in the GDT for another descriptor.
E_TYPE	8002H	The `vseg` parameter is not a token for a virtual segment.
E_VMEM	00F0H	There is insufficient virtual memory available in the virtual segment to satisfy this request.
E_VSEG	80F0H	The calling task does not belong to the same job that created the virtual segment.

rqv_allocate_at

Allocates physical memory to a virtual segment at a specific offset.

Syntax, PL/M and C

```
CALL rqv$allocate$at (vseg, offset, size, except_ptr);
```

```
rqv_allocate_at (vseg, offset, size, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
vseg	TOKEN	SELECTOR
offset	POINTER	void near *
size	DWORD	UINT_32
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

vseg A token for the virtual segment. If this parameter is null and the application is flat model, the parameter indicates the application's virtual segment. For segmented model applications, a null value is an error.

offset The location within the virtual segment where the allocated physical memory is to begin. The offset must be on a 4 Kbyte boundary.

size The amount, in bytes, of contiguous physical memory to be allocated. If not a multiple of 4 Kbytes, the size will be rounded up to the next 4 Kbyte boundary.

except_ptr
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

This call is typically used by system utilities such as the Application Loader, not by an application. The calling task must belong to the same job that created the virtual segment. Allocation starts within the virtual segment at `offset` for `size` bytes (the size is rounded up to 4K pages by the call). The allocated pages are contiguous; they start and end on 4K boundaries.

The call fails if `size` of contiguous physical memory are not available, if `size` bytes exceeds the segment size, or if there is a collision with previously allocated space. The memory required for page tables is charged to the calling job's memory pool. The first allocation to a virtual segment incurs a 4K (minimum) overhead for at least one page table.

If vseg is a null selector (0) and the application is flat model, the application's virtual segment is assumed; otherwise, a null selector is an error.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_ALIGNMENT	80F1H	The offset parameter is not on a 4K boundary.
E_ALLOCATED	00F1H	The requested area of the virtual segment already has physical memory allocated to it.
E_BAD_ADDR	800FH	The offset parameter is beyond the end of the virtual segment.
E_EXIST	0006H	The vseg parameter represents a segment that is being deleted, or vseg is a null token and the caller is not a flat model application
E_MEM	0002H	There is insufficient physical memory available to satisfy this request.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The size parameter is zero or is larger than the virtual segment, or the size + offset is beyond the end of the virtual segment.
E_SLOT	000CH	There is no room in the GDT for another descriptor.
E_TYPE	8002H	The vseg parameter is not a token for a virtual segment.
E_VMEM	00F0H	There is insufficient virtual memory available in the virtual segment to satisfy this request.
E_VSEG	80F0H	The calling task does not belong to the same job that created the virtual segment.

rqv_change_access

Changes the access rights for physical memory within a virtual segment.

Syntax, PL/M and C

```
CALL rqv$change$access (vseg, offset, size, access,
                        except_ptr,);
```

```
rqv_change_access (vseg, offset, size, access, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
vseg	TOKEN	SELECTOR
offset	POINTER	void near *
size	DWORD	UINT_32
access	DWORD	UINT_32
except_ptr	POINTER to WORD_16	UINT_16 far *

Parameters

vseg A token for the virtual segment. If this parameter is null and the application is flat model, the parameter indicates the application's virtual segment. For segmented model applications, a null value is an error.

offset A pointer to the location within the virtual segment where the physical memory begins for which access rights will be changed.

size The amount, in bytes, of contiguous physical memory for which access rights will be changed.

access

The new access rights of the memory, encoded as follows:

Bit	Description
0	0 = Pages will be read/write 1 = Pages will be read-only
1-31	Reserved, set to zero.

except_ptr

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Rqv_change_access changes the access rights associated with the physical memory pages within `vseg` starting at `offset`, for `size` number of bytes (the call rounds both `offset` and `size` up to a 4K boundary). The call sets the attributes of every page within this area to `access`. **Rqv_change_access** fails if `offset` + `size` is beyond the end of the virtual segment or if there are no allocated pages at `offset`.

The calling task must belong to the same job that created the virtual segment. If `vseg` is a null selector (0) and the application is flat model, the application's virtual segment is assumed; otherwise, a null selector is an error.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The `offset` parameter is beyond the end of the virtual segment.
E_EXIST	0006H	The `vseg` parameter represents a segment that is being deleted, or `vseg` is a null token and the caller is not a flat model application
E_NOT_ALLOCATED	00F2H	There is no physical memory allocated at the requested area of the virtual segment.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_PARAM	8004H	The `size` parameter is larger than the virtual segment, or `size` is 0.
E_TYPE	8002H	The `vseg` parameter is not a token for a virtual segment.
E_VSEG	80F0H	The calling task does not belong to the same job that created the virtual segment.

rqv_create_segment

Creates a virtual segment with no physical memory allocated to it.

Syntax, PL/M and C

```
vseg_t = rqv$create$segment (vseg_size, except_ptr);

vseg_t = rqv_create_segment (vseg_size, except_ptr);
```

Parameter	**PL/M Data Type**	**C Data Type**
`vseg_t`	TOKEN	SELECTOR
`vseg_size`	DWORD	UINT_32
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`vseg_t`

A token for the newly created virtual segment.

Parameters

`vseg_size`

Specifies the size, in bytes, of the virtual segment.

`except_ptr`

A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

Rqv_create_segment creates a segment where `vseg_size` virtual address space is allocated but physical memory is not. Virtual memory is allocated on a 4 Mbyte boundary, in 4 Mbyte units. Therefore, the call rounds up the `vseg_size` parameter to the nearest 4 Mbyte boundary. This allows a system-wide total of up to 1K virtual segments (minus the physical memory in the system). Use the **rq_delete_segment** call to delete the virtual segment.

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_LIMIT	0004H	The calling task's job has already reached its object limit.

E_MEM	0002H	There is insufficient physical memory available to create a virtual segment object.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SLOT	000CH	There is no room in the GDT for another descriptor.
E_VMEM	00F0H	There is insufficient virtual memory available in the system to create a virtual segment of the specified size.

rqv_free

Frees physical memory associated with a virtual segment.

Syntax, PL/M and C

```
actual = rqv$free (vseg, offset, size, except_ptr,);

actual = rqv_free (vseg, offset, size, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`actual`	DWORD	UINT_32
`vseg`	TOKEN	SELECTOR
`offset`	POINTER	void near *
`size`	DWORD	UINT_32
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`actual` The number of bytes of physical memory that were freed.

Parameters

`vseg` A token for the virtual segment. If this parameter is null and the application is flat model, the parameter indicates the application's virtual segment. For segmented model applications, a null value is an error.

`offset` The location within the virtual segment where the physical memory is to be freed.

`size` Must be set to 0FFFFFFFFH (or -1), meaning delete all contiguous physical memory found at `offset`.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The deallocation deletes all contiguous physical memory that is found at `offset`, even if the contiguous block begins before `offset`. In other words, **rqv_free** deletes all memory previously allocated from a single call to **rqv_allocate** or **rqv_allocate_at**. All memory allocation and deallocation is performed in contiguous memory blocks, maintaining the physically contiguous memory model required by iRMX OS device drivers.

If the physical memory pointed to by offset was mapped by a previous call to **rqv_map_physical**, the mapping is deleted and the associated virtual memory is freed.

The relationship between **rqv_allocate** and **rqv_free** corresponds to the relationship between **rq_create_segment** and **rq_delete_segment**. However, you can use **rq_delete_segment** instead to automatically free all physical memory within a virtual segment. A page table is automatically freed to the calling job's memory pool when all pages within the page are freed.

The calling task must belong to the same job that created the virtual segment. If vseg is a null selector (0) and the application is flat model, the application's virtual segment is assumed; otherwise, a null selector is an error.

See also: **rq_create_segment** and **rq_delete_segment** Nucleus calls

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_BAD_ADDR	800FH	The offset parameter is beyond the end of the virtual segment.
E_EXIST	0006H	The vseg parameter represents a segment that is being deleted, or vseg is a null token and the caller is not a flat model application
E_NOT_ALLOCATED	00F2H	There is no physical memory allocated at the requested area of the virtual segment.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_SUPPORT	0023H	The size parameter was not set to 0FFFFFFFFH (-1).
E_TYPE	8002H	The vseg parameter is not a token for a virtual segment.
E_VSEG	80F0H	The calling task does not belong to the same job that created the virtual segment.

rqv_map_physical

Maps physical memory into the address space within a virtual segment. This call is the flat model equivalent of the **rqe_create_descriptor** call.

⚠ **CAUTION**
This system call can set up an address space to refer to any area of physical memory, even if other descriptors already refer to that memory. Although this may be useful for aliasing purposes, do not overlap memory accidentally.

See also: **rqv_free** and Nucleus call **rqe_create_descriptor**

Syntax, PL/M and C

```
offset = rqv$map$physical (vseg, abs_addr, size, except_ptr);

offset = rqv_map_physical (vseg, abs_addr, size, except_ptr);
```

Parameter	PL/M Data Type	C Data Type
`offset`	POINTER	void near *
`vseg`	TOKEN	SELECTOR
`abs_addr`	DWORD	UINT_32
`size`	DWORD	UINT_32
`except_ptr`	POINTER to WORD_16	UINT_16 far *

Return Value

`offset` A near pointer to the mapped physical memory within the virtual segment.

Parameters

`vseg` A token for the virtual segment. If this parameter is null and the application is flat model, the parameter indicates the application’s virtual segment. For segmented model applications, a null value is an error.

`abs_addr`
Specifies a full, 32-bit physical address. This is the address where the mapping will start. The address must be aligned on a 4K boundary.

`size` The amount, in bytes, of contiguous physical memory to be mapped. The amount of memory must be a multiple of 4K.

`except_ptr`
A pointer to a variable declared by the application where the call returns a condition code.

Additional Information

The **map_physical** call maps physical memory starting at `abs_addr` for `size` bytes into the virtual segment specified by `vseg`. Because of hardware alignment restrictions, `abs_addr` must be on a 4K boundary and `size` must be a multiple of 4K. Due to the critical nature of this call, the `abs_addr` and `size` parameters are not rounded up by the call.

If `vseg` is a null selector (0) and the application is flat model, the application's virtual segment is assumed; otherwise, a null selector is an error.

Use **rqv_free** to delete the mapping created by this call and to free the virtual memory associated with it. This is similar to the use of **delete_segment** to delete a descriptor created with **rqe_create_descriptor**.

See also: **rqv_free,** and Nucleus calls **delete_segment** and **rqe_create_descriptor**

Condition Codes

E_OK	0000H	No exceptional conditions occurred.
E_ALIGNMENT	80F1H	The `abs_addr` parameter is not on a 4K boundary or `size` is not a multiple of 4K.
E_BAD_ADDR	800FH	The `offset` parameter is beyond the end of the virtual segment.
E_EXIST	0006H	The `vseg` parameter represents a segment that is being deleted, or `vseg` is a null token and the caller is not a flat model application
E_MEM	0002H	There is insufficient physical memory available to create page table(s) for this request.
E_NOT_CONFIGURED	0008H	This system call is not part of the present configuration.
E_TYPE	8002H	The `vseg` parameter is not a token for a virtual segment.
E_VMEM	00F0H	There is insufficient virtual memory available in the virtual segment to satisfy this request.

□ □ □

Application Loader Examples A

rqe_a_load_io_job and rqe_s_load_io_job example

```
/*
 *     "C" examples for
 *     rqe_a_load_io_job
 *     rqe_s_load_io_job
 */



    /*
     * prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>
#include <rmx_def.h>


#define UNLIMITED      0xFFFFF
#define NO_DELAY       0
#define DELAY_REQ      2
#define TERMINATION_OK 0x100


    /*
     * This module is an example using two Application Loader system
     * calls: rqe_a_load_io_job and rqe_s_load_io_job.  The calling
     * task's priority is the maximum allowed for its job.
     */
```

```
main ()
{

    SELECTOR        conn;
    SELECTOR        aload_mbox;
    SELECTOR        sload_mbox;
    SELECTOR        aload_job;
    SELECTOR        sload_job;
    SELECTOR        aload_res_t;
    SELECTOR        exit_seg_t;
    UINT_32         pool_min;
    UINT_32         pool_max;
    UINT_8          priority;
    UINT_16         status;
    UINT_16         job_flags;
    UINT_16         task_flags;
    char            my_name [] = {7,"my_prog"};

    A_LOAD_LRS_STRUCT far *     aload_res_seg;
    EXCEPTION_STRUCT            except_handler;


        /*
         * Initialize exception handler structure and create
         * mailboxes for two Application Loader calls.
         */

    except_handler.exception_handler_ptr = NULL;
    except_handler.exception_mode = 0;

    sload_mbox= rq_create_mailbox ((UINT_16) FIFO_QUEUING, &status);
    if (status != E_OK) goto exit;


        /*
         * Rqe_a_load_io_job.  Obtain a connection to the file, then
         * prepare the input parameters. Let the Application Loader
         * decide the memory pool size for the job. Do not allow the
         * new job to borrow memory from its parent; set max = to
         * min. The loaded code starts execution as soon as it is in
         * memory and has the maximum priority of its parent.
         */
```

```
conn = rq_s_attach_file (my_name, &status);
if (status != E_OK) goto exit;

pool_min    =   0;
pool_max    =   0;
priority    =   0;
job_flags   =   0;
task_flags  =   NO_DELAY;

aload_job = rqe_a_load_io_job (conn,
                        pool_min,
                        pool_max,
                        (EXCEPTIONSTRUCT far *) &except_handler,
                        job_flags,
                        priority,
                        task_flags,
                        aload_mbox,
                        &status);
if (status != E_OK) goto exit;

    /*
     * Rqe_a_load_io_job is asynchronous, so only its sequential
     * part is is executed and loading is still in progress.
     * Prepare the parameters for rqe_s_load_io_job and call it.
     * Let the Application Loader decide memory pool size, but
     * let the job borrow memory from its parent. Specify delay
     * to control execution of the code. The Application Loader
     * calls will probably load the same file concurrently.
     */

pool_max    =   UNLIMITED;
task_flags  =   DELAY_REQ;

aload_job = rqe_s_load_io_job (my_name,
                        pool_min,
                        pool_max,
                        (EXCEPTIONSTRUCT far *) &except_handler,
                        job_flags,
                        priority,
                        task_flags,
                        sload_mbox,
                        &status);
if (status != E_OK) goto exit;
```

```
    /*
     * Rqe_s_load_io_job has completed. Wait at the specified
     * mailbox for results about rqe_a_load_io_job.
     */

aload_res_t = rq_receive_message (aload_mbox,
                                  (UINT_16) WAIT_FOREVER,
                                  NULL,
                                  &status );
if (status != E_OK) goto exit;


    /*
     * Inspect the the Loader Result Segment to determine the
     * allocated memory pool size, or if an error occurred, see
     * its nature.
     */

aload_res_seg = buildptr(aload_res_t,(void near*) 0);

if (aload_res_seg->except_code != TERMINATION_OK) goto exit;


    /*
     * Rqe_a_load_io_job completed successfully, and the loaded
     * program is waiting for the CPU since no delay was
     * requested.  The second copy of the program is waiting in
     * memory for permission to start.  Let it start.
     */

rq_start_io_job (sload_job, &status);


    /*
     * The two loaded programs are running. Wait for them to
     * terminate using rq_exit_io_job, then kill them.
     */

exit_seg_t = rq_receive_message (aload_mbox,
                                 (UINT_16) WAIT_FOREVER,
                                 NULL,
                                 &status);
if (status != E_OK) goto exit;
```

```
        /*
         * Examine the exit message.
         */

    rq_delete_job (aload_job, &status);
    if (status != E_OK) goto exit;
    exit_seg_t = rq_receive_message (sload_mbox,
                                     (UINT_16) WAIT_FOREVER,
                                     NULL,
                                     &status);
    if (status != E_OK) goto exit;

    rq_delete_job (sload_job, &status);
    if (status != E_OK) goto exit;


        /*
         * Exit of program.
         */

exit:
    rq_exit_io_job ((UINT_16) 0, NULL, &status);
    if (status != E_OK) {}


        /*
         * The end. If an error was detected in this module,
         * recovery can be attempted, a message can be printed to
         * the terminal, or the program can just terminate.
         */

}
```

❑ ❑ ❑

Nucleus Examples B

Examples using these calls are included here:

rqe_create_descriptor
rq_create_extension
rqe_create_job
rq_create_mailbox
rq_create_region
rq_create_segment
rq_create_semaphore
rq_create_task
rq_delete_job
rq_force_delete
rqe_get_address
rq_get_exception_handler
rqe_get_pool_attrib
rq_get_pool_attrib
rq_get_task_tokens
rq_get_type
rqe_offspring
rq_offspring
rq_receive_data
rq_receive_message
rq_receive_units
rqe_set_os_extension
rq_set_pool_min

See also: *:rmx:demo/c/interrupt* directory for demos using **rq_signal_interrupt**, **rq_reset_interrupt**, **rqe_timed_interrupt**, and **rq_set_interrupt**

rqe_create_descriptor example

```
/*
 *      "C" example for
 *      rqe_create_descriptor
 *      rqe_change_descriptor
 *      rqe_delete_descriptor
 *
 */

    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>


main ()
{

    SELECTOR        desc_token;
    UINT_32         abs_addr;
    UINT_32         seg_size;
    UINT_16         status;


        /*
         *  The absolute address of the memory area being given an
         *  address is 2M bytes.
         */

    abs_addr = 0x200000;


        /*
         *  The size of the block is 256 bytes.
         */

    seg_size = 256;
 /*
         *  The token desc_token is returned when the calling task
         *  invokes create_descriptor.
         */

    desc_token = rqe_create_descriptor (abs_addr,
                                        seg_size,
                                        &status);



        /*
         *  The absolute address of the memory area is changed
         *  to 10M bytes.
         */

    abs_addr = 0xA00000;


        /*
         *  The size of the requested descriptor is 512 bytes.
         */

    seg_size = 512;


        /*
         *  Change the position of the descriptor.
         */

    rqe_change_descriptor  (desc_token,
                            abs_addr,
                            seg_size,
                            &status);



        /*
         *  When the descriptor is no longer needed, it may be
         *  deleted by a task that knows the descriptor token.
         */

    rqe_delete_descriptor  (desc_token, &status);

}
```

rq_create_extension example

```
/*
 *      "C" example for
 *      rq_create_extension
 *      rq_delete_extension
 *
 */

    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>


main ()
{

    SELECTOR        extension;
    UINT_16         type_code;
    SELECTOR        deletion_mailbox;
    UINT_16         status;


        /*
         *  Supply a valid value for a new type.
         */

    type_code = 0x8000;


        /*
         *  No deletion mailbox is desired for this new type.
         */

    deletion_mailbox = (SELECTOR)NULL;
```

```
        /*
         *  To delete an extension, a task must have the token
         *  for that extension.  In this example, the needed token
         *  is known because the calling task creates the extension.
         */

    extension = rq_create_extension  (type_code,
                                      deletion_mailbox,
                                      &status);


        /*
         * When the extension is no longer needed, it may be deleted
         * by any task that knows the token for the extension.
         */

    rq_delete_extension  (extension, &status);

}
```

rqe_create_job example

```
/*
 *       "C" example for rqe_create_job
 */

    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>
#include <rmx_def.h>

    /*
     *  Dummy task for job creation.
     */

void initial_task (void)
{}

    /*
     *  Main task to create the job.
     */

main ()
{

    SELECTOR        job;
    UINT_16         directory_size;
    SELECTOR        param_obj;
    UINT_32         pool_min;
    UINT_32         pool_max;
    UINT_16         max_objects;
    UINT_16         max_tasks;
    UINT_8          max_priority;
    UINT_16         job_flags;
    UINT_8          task_priority;
    SELECTOR        data_seg;
    UINT_16 far *   stack_ptr;
    UINT_32         stack_size;
    UINT_16         task_flags;
    UINT_16         status;
```

```
EXCEPTIONSTRUCT far * except_handler;
void (far *start_address);

    /*
     * Set up the create job parameters using the following
     * characteristics: 10 entries in object directory, new job
     * has no parameter object, min 0x1ff, max 0xffff, 16-byte
     * paragraphs in job pool, no limit to number of objects, 10
     * tasks can exist simultaneously, inherit max priority of
     * parent, use system default except handler, parameter
     * validation is on, set init task to max priority, points
     * to first instruction of initial task, init task sets up
     * own data segment, Nucleus allocates stack, 1024 bytes in
     * stack of initial task, no floating-point instructions.
     */

directory_size  =   10;
param_obj       =   NULL_TOKEN;
pool_min        =   0x1FF;
pool_max        =   0xFFFFF;
max_objects     =   0xFFFF;
max_tasks       =   10;
max_priority    =   0;
except_handler  =   NULL;
job_flags       =   0;
task_priority   =   0;
start_address   =   &initial_task;
data_seg        =   NULL_TOKEN;
stack_ptr       =   NULL;
stack_size      =   1024;
task_flags      =   0;


    /*
     *  Create the job.
     */
```

```
   job = rqe_create_job (directory_size,
                         param_obj,
                         pool_min,
                         pool_max,
                         max_objects,
                         max_tasks,
                         max_priority,
                         except_handler,
                         job_flags,
                         task_priority,
                         start_address,
                         data_seg,
                         stack_ptr,
                         stack_size,
                         task_flags,
                         &status);

}
```

rq_create_mailbox example

```
/*
 *      "C" examples for
 *      rq_create_mailbox
 *      rq_delete_mailbox
 */

    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>
#include <rmx_def.h>


    /*
     *  Main task to create, delete the mailbox.
     */

main ()
{

    SELECTOR        mailbox;
    UINT_16         mailbox_flags;
    UINT_16         status;


        /*
         *  Designates a high performance object queue
         *  of eight objects, first-in/first-out task queue.
         */

    mailbox_flags = FIFO_QUEUING;


        /*
         *  The token is returned when the calling task invokes
         *  create_mailbox.
         */

    mailbox = rq_create_mailbox (mailbox_flags, &status);
```

```
        /*
         * To delete a mailbox, a task must have the token for that
         * mailbox. In this example, the needed token is known
         * because the calling task creates the mailbox. When the
         * mailbox is not needed, it may be deleted.
         */

    rq_delete_mailbox (mailbox, &status);

}
```

rq_create_region example

```
/*
 *      "C" examples for
 *      rq_create_region
 *      rq_accept_control
 *      rq_send_control
 *      rq_receive_control
 *      rq_delete_region
 */

    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>
#include <rmx_def.h>

    /*
     *  Main task to create the region.
     */

main ()
{

    SELECTOR        region;
    UINT_16         region_flags;
    UINT_16         status;


        /*
         * To access the data within a region, a task must have the
         * token for that region. In this example, the needed token
         * is known because the calling task creates the region.
         * This is created to use the priority based queuing scheme.
         */

    region_flags = PRIOR_QUEUING;

    region = rq_create_region (region_flags, &status);
```

```
        /*
         *  At some point in the task, access is needed to the data
         *  protected by the region.  The calling task invokes
         *  accept_control and obtains access to the data.
         */

    rq_accept_control (region, &status);


        /*
         *  When the task is ready to relinquish access to the data
         *  protected by the region, it invokes send_control.
         */

    rq_send_control (&status);


        /*
         * When access to the data protected by a region is needed
         * and the calling task is willing to wait, it may invoke
         * receive_control.
         */

    rq_receive_control (region, &status);


        /*
         *  When the task is ready, it invokes send_control.
         */

    rq_send_control (&status);


        /*
         * When the region is no longer needed, it may be deleted by
         * any task that knows the token for the region.
         */

    rq_delete_region (region, &status);

}
```

rq_create_segment example

```
/*
 *      "C" examples for
 *      rq_create_segment
 *      rqe_change_object_access
 *      rq_delete_segment
 */

    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>

    /*
     *  Main task to create, change access, and delete segment.
     */

main ()
{

    SELECTOR        segment;
    UINT_32         seg_size;
    UINT_8          access;
    UINT_8          limit_mode;
    UINT_16         status;


        /*
         *  The size of the requested segment is 256 bytes.
         */

    seg_size = 256;


        /*
         *  The token is returned when the calling task invokes
         *  create segment.
         */

    segment = rq_create_segment (seg_size, &status);
```

```
        /*
         * The access rights are changed to make a writable data
         * segment present in memory, and not accessed. Single byte
         * granularity.
         */

    access      =   0x92;
    limit_mode  =   0;

    rqe_change_object_access (segment,
                              access,
                              limit_mode,
                              &status);


        /*
         * When the segment is no longer needed, it may be deleted
         * by any task that knows the token for the segment.
         */

    rq_delete_segment (segment, &status);

}
```

rq_create_semaphore example

```
/*
 *      "C" examples for
 *      rq_create_semaphore
 *      rq_delete_semaphore
 */


    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>


    /*
     *  Main task to create and delete semaphore.
     */

main ()
{

    SELECTOR        semaphore;
    UINT_16         semaphore_flags;
    UINT_16         initial_value;
    UINT_16         max_value;
    UINT_16         status;


        /*
         *  The new semaphore has one initial unit,
         *  and can have a maximum of 16 units,
         *  and is designated as a first-in/first-out task queue.
         */

    initial_value   =   1;
    max_value       =   0x10;
    semaphore_flags =   0;
```

```
        /*
         *  The token is returned when the calling task
         *  invokes create_semaphore.
         */

    semaphore = rq_create_semaphore  (initial_value,
                                      max_value,
                                      semaphore_flags,
                                      &status);


        /*
         * To delete a semaphore, a task must have the token for
         * that semaphore. In this example, the needed token is
         * known because the calling task creates the semaphore.
         */

    rq_delete_semaphore (semaphore, &status);

}
```

rq_create_task example

```
/*
 *      "C" examples for
 *      rq_create_task
 *      rq_suspend_task
 *      rq_resume_task
 *      rq_catalog_object
 *      rq_uncatalog_object
 *      rq_set_priority
 *      rqe_set_max_priority
 *      rq_get_priority
 *      rq_sleep
 *      rq_delete_task
 */


    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>
#include <rmx_def.h>


    /*
     *  Bind your taskcode to this demo.
     */

extern void taskcode (void);


    /*
     *  Main task to create and delete task.
     */
```

```
main ()
{

    SELECTOR        task;
    SELECTOR        job;
    SELECTOR        calling_task_job;
    UINT_8          priority;
    UINT_8          selection;
    void far *      start_address;
    SELECTOR        data_seg;
    UINT_16 far *   stack_ptr;
    UINT_32         stack_size;
    UINT_16         task_flags;
    UINT_16         status;
    char            taskname [] = {9,"TASKCODE"};


        /*
         *  Parameters for the create task.
         *  Task sets up own data seg, automatic stack allocation,
         *  no floating point instructions.
         */

    start_address   =   taskcode;
    data_seg        =   NULL_TOKEN;
    stack_ptr       =   NULL;
    task_flags      =   0;
    priority        =   200;
    stack_size      =   512;


        /*
         *  Create a non-interrupt task whose code is
         *  labeled TASKCODE.
         */

    task = rq_create_task (priority,
                           start_address,
                           data_seg,
                           stack_ptr,
                           stack_size,
                           task_flags,
                           &status);
        /*
         * To use suspend_task, a task must know the token for that
         * task. In this example, the needed token is known because
         * the calling task creates the new task.  Suspend_task
         * increases by one the suspension depth of the new task.
         */

    rq_suspend_task (task, &status);


        /*
         * Using the token for the suspended task (whose code is
         * labeled TASKCODE), the calling task invokes resume_task
         * to decrease by one the suspension depth of the suspended
         * task.
         */

    rq_resume_task (task, &status);


        /*
         * The calling task in this example does not need to invoke
         * catalog_object to ensure the successful use of
         * set_priority. To allow other tasks access to the new
         * task, however, requires that the task's object token  be
         * cataloged.
         */

    job = NULL_TOKEN;
    rq_catalog_object (job, task, taskname, &status);


        /*
         * The new task (whose code is labeled TASKCODE) is not an
         * interrupt task, so its priority may be changed
         * dynamically by invoking set_priority.
         */

    priority = 166;
    rq_set_priority (task, priority, &status);
```

```
        /*
         * If the need for the higher priority is no longer present,
         * invoke set_priority a second time to change the priority
         * back to its original priority.
         */

    priority = 200;
    rq_set_priority (task, priority, &status);


        /*
         *  Try to set the task priority to more than the job's max
         *  priority.  This will cause an E_LIMIT exception.
         */

    priority = 128 - 10;
    rq_set_priority (task, priority, &status);


        /*
         * If the rq_set_priority call causes an E_LIMIT condition,
         * use rqe_set_max_priority to raise the job's maximum
         * priority.
         */

    if (status == E_LIMIT)
    {
        priority            =   128 - 20;
        calling_task_job    =   NULL_TOKEN;
        rqe_set_max_priority (calling_task_job, priority, &status);
        priority            =   128 - 10;
        rq_set_priority (task, priority, &status);
    }


        /*
         *  Get_priority returns the priority of the calling task.
         */

    calling_task_job = NULL_TOKEN;
    priority = rq_get_priority (calling_task_job, &status);
```

```
        /*
         *  Invoke sleep to put the calling task in the asleep
         *  state for 100 (1 second) clock ticks.
         */

    rq_sleep ((UINT_16) 100, &status);


        /*
         *  Remove the task token from the object directory.
         */

    rq_uncatalog_object (job, taskname, &status);


        /*
         *  To use delete_task, a task must know the token for the
         *  task to be deleted.  In this example, the needed token
         *  is known because the calling task creates the new task.
         *  Any task that knows this task's token may delete the
         *  task.
         */

     rq_delete_task (task, &status);

}
```

rq_delete_job example

```
/*
 *      "C" examples for rq_delete_job
 */


    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>
#include <rmx_def.h>


    /*
     *  Main task to delete job.
     */

main ()
{

    SELECTOR        job;
    UINT_16         status;


        /*
         *  Set job to calling task's job.
         */

    job = NULL_TOKEN;


        /*
         *  If you set the job parameter to (SELECTOR)NULL,
         *  delete_job will delete the calling task's job.
         */

    rq_delete_job (job, &status);

}
```

rq_force_delete example

```
/*
 *      "C" examples for
 *      rq_force_delete
 *      rq_create_semaphore
 *      rq_disable_deletion
 */


    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>
#include <rmx_def.h>


    /*
     *  Main task to force deletion.
     */

main ()
{

    SELECTOR        semaphore;
    SELECTOR        extension;
    UINT_16         semaphore_flags;
    UINT_16         initial_value;
    UINT_16         max_value;
    UINT_16         status;


        /*
         *  The new semaphore has one initial unit,
         *  and can have a maximum of 16 units,
         *  and is designated as a first-in/first-out task queue.
         */

    initial_value   =   1;
    max_value       =   0x10;
    semaphore_flags =   0;
```

```
        /*
         *  In this example, the calling task creates the object
         *  to become immune to deletion.  Create_semaphore is
         *  invoked by the calling task to create a semaphore.
         */

    semaphore = rq_create_semaphore  (initial_value,
                                      max_value,
                                      semaphore_flags,
                                      &status);


        /*
         *  Using the semaphore token, the calling task invokes
         *  disable_deletion to increase the disabling depth by
         *  one.  This makes the semaphore immune to ordinary
         *  deletion.
         */

    rq_disable_deletion (semaphore, &status);


        /*
         *  To delete the semaphore, the calling task invokes
         *  force_delete.  This call deletes the semaphore even
         *  though the disabling depth of the semaphore is one.
         *  There is no extension object, so set the extension
         *  parameter to NULL.
         */

    extension = NULL_TOKEN;
    rq_force_delete (extension, semaphore, &status);

}
```

rqe_get_address example

```
/*
 *      "C" examples for
 *      rqe_get_address
 *      rq_create_segment
 */



    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>



    /*
     *  Main task using rqe_get_address to create a segment,
     *  convert the segment's SELECTOR to a pointer, and
     *  return the physical address of the segment.
     */

main ()
{

    SELECTOR        segment;
    UINT_32         seg_size;
    void far *      log_addr;
    UINT_32         phys_addr;
    UINT_16         status;


        /*
         *  The size of the requested segment is 256 bytes.
         */

    seg_size = 0256;

        /*
         *  The token is returned when the calling task invokes
         *  create segment.
         */
segment = rq_create_segment (seg_size, &status);


        /*
         *  The segment SELECTOR is converted to a pointer.
         */

    log_addr = buildptr (segment, (void near*) 0);


        /*
         *  The pointer with the logical address is used to
         *  get the physical address.
         */

    phys_addr = rqe_get_address (log_addr, &status);

}
```

rq_get_exception_handler example

```
/*
 *      "C" examples for
 *      rq_get_exception_handler
 *      rq_set_except_handler
 */

    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>


    /*
     *  Bind your exception_handler to this demo.
     */

extern void exception_handler ();


    /*
     *  Main task to use get and set exception handler.
     */

main ()
{

    EXCEPTIONSTRUCT new_x_handler;
    EXCEPTIONSTRUCT x_handler;
    UINT_16         status;


        /*
         * The address of the calling task's exception handler and
         * the value of the task's exception mode (when to pass
         * control to the exception handler) are both returned when
         * the calling task invokes get_exception_handler.
         */

    rq_get_exception_handler (&x_handler, &status);
/*
         *  Set up the parameters for new exception handler,
         *  all exceptions.
         */

    new_x_handler.exceptionhandlerptr = exception_handler;
    new_x_handler.exceptionmode = 3;


        /*
         * The  calling task may invoke set_exception_handler to
         * first set a new exception handler and then to later reset
         * the old exception handler.
         */

    rq_set_exception_handler (&new_x_handler, &status);


        /*
         * No longer needing the new exception handler, the calling
         * task uses the address and mode of the old exception
         * handler to return exception handling to its original
         * exception handler.
         */

    rq_set_exception_handler (&x_handler, &status);

}
```

rqe_get_pool_attrib example

```
/*
 *      "C" examples for rqe_get_pool_attrib
 */

    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>
#include <rmx_def.h>

    /*
     *  Main task to use rqe_get_pool_attrib.
     */

main ()
{

    UINT_16             status;
    EPOOLATTRIBSTRUCT   mem_pool;


        /*
         *  Set the calling task's job as the calling job.
         */

    mem_pool.targetjob  =   NULL_TOKEN;


        /*
         *  The parent job's token, the maximum and minimum size
         *  of the memory pool, the original value of mem_pool_min,
         *  and the amount of allocated, available, and borrowed
         *  memory in the memory pool of the calling task's job are
         *  all returned when the task invokes rqe_get_pool_attrib.
         */

    rqe_get_pool_attrib (&mem_pool, &status);

}
```

rq_get_pool_attrib example

```
/*
 *      "C" examples for rq_get_pool_attrib
 */

    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>


    /*
     *  Main task to use rq_get_pool_attrib.
     */

main ()
{

    UINT_16             status;
    POOLATTRIBSTRUCT    mem_pool;


        /*
         *  The maximum and minimum size of the memory pool,
         *  the original value of the minimum pool size, and
         *  the allocated and available number of 16-byte
         *  paragraphs in the memory pool of the calling
         *  task's job are all returned when the calling task
         *  invokes get_pool_attrib.
         */

    rq_get_pool_attrib (&mem_pool, &status);

}
```

rq_get_task_tokens example

```
/*
 *      "C" examples for
 *      rq_get_task_tokens
 *      rq_disable_deletion
 *      rq_enable_deletion
 */

    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>

    /*
     *  Main task to disable and enable deletion.
     */

main ()
{

    SELECTOR        task;
    UINT_16         status;
    UINT_8          selection;

        /*
         *  In this example, the calling task will be the object
         *  to become immune to deletion.  Get_task_token is invoked
         *  by the calling task to obtain its own token.
         */

    selection = 0;
    task = rq_get_task_tokens (selection, &status);

        /*
         * Using its own token, the calling task invokes
         * disable_deletion to increase its own disabling depth by
         * one. This makes the calling task immune to ordinary
         * deletion.
         */
    rq_disable_deletion (task, &status);


        /*
         *  In order to allow itself to be deleted, the calling task
         *  invokes enable_deletion.  This call decreases by one the
         *  disabling depth of an object. In this example, the
         *  object is the calling task.
         */

    rq_enable_deletion (task, &status);

}
```

rq_get_type example

```
/*
 *      "C" examples for
 *      rq_get_type
 *      rq_lookup_object
 *      rq_receive_message
 */



    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>
#include <rmx_def.h>


    /*
     *  Main task to use rq_get_type.
     */

main ()
{

    UINT_16         type_code;
    SELECTOR        object;
    SELECTOR        job;
    SELECTOR        mailbox;
    char            name[] = {3,"MBX"};
    UINT_16         time_limit;
    SELECTOR        response;
    UINT_16         status;


        /*
         *  To invoke get_type, the calling task must have the
         *  token for an object.  In this example, the calling
         *  task invokes lookup_object and then receive_message
         *  to receive the token for an object of unknown type
         *  (object_token).
         */
    job         =   NULL_TOKEN;
    time_limit  =   WAIT_FOREVER;
    mailbox = rq_lookup_object (job, name, time_limit, &status);


        /*
         *  Receive_message returns object_token to the calling
         *  task after the calling task invoked lookup_object to
         *  receive the token for the mailbox named 'MBX'.  'MBX'
         *  had been designated as the mailbox another task would
         *  use to send an object.
         */

    object = rq_receive_message (mailbox,
                                 time_limit,
                                 (SELECTOR far *)&response,
                                 &status);


        /*
         *  Using the type code returned by get_type, the calling
         *  task can find out if the object is a job, task, mailbox,
         *  region, segment, semaphore, extension, or composite.
         */

    type_code = rq_get_type (object, &status);

}
```

rqe_offspring example

```
/*
 *      "C" examples for rqe_offspring
 */


    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>
#include <rmx_def.h>


    /*
     *  Main task to use rqe_offspring.
     */

main ()
{

    SELECTOR        job;
    OFFSPRINGSTRUCT list;
    UINT_16         status;


        /*
         *  In this example, the calling task invokes
         *  rqe_offspring to obtain a list of up to 20
         *  tokens for the jobs that are the immediate
         *  children of the calling task's job.
         */

    job         =   NULL_TOKEN;
    list.maxnum =   20;
    rqe_offspring (job, &list, &status);

}
```

rq_offspring example

```
/*
 *      "C" examples for rq_offspring
 */


    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>
#include <rmx_def.h>


    /*
     *  Main task to use rq_offspring.
     */

main ()
{

    SELECTOR        job;
    SELECTOR        token_list;
    UINT_16         status;


        /*
         * In this example, the calling task invokes offspring to
         * obtain a token for a segment. This segment contains the
         * tokens for jobs that are immediate children of the
         * calling task's job.
         */

    job = NULL_TOKEN;
    token_list = rq_offspring (job, &status);

}
```

rq_receive_data example

```
/*
 *      "C" examples for
 *      rq_receive_data
 *      rq_send_data
 *      rq_lookup_object
 *      rq_catolog_object
 *      rq_create_mailbox
 */

    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>
#include <rmx_def.h>

#define BUF_SIZE  128

    /*
     *  Procedure to create, catolog, and send data to mailbox.
     */

start_mail ()
{

    UINT_16         actual;
    UINT_16         mailbox_flags;
    SELECTOR        job;
    SELECTOR        mailbox;
    char            name[] = {3,"MBX"};
    UINT_16         time_limit;
    char            send_message[BUF_SIZE];
    UINT_16         status;

        /*
         *  Create and catalog a data mailbox.
         */

    mailbox_flags   =   0x20;
    job             =   NULL_TOKEN;
```

```
        /*
         *  The calling task creates a mailbox and catalogs the
         *  mailbox token.  The calling task then sends message
         *  data to the mailbox.
         */

    mailbox = rq_create_mailbox (mailbox_flags, &status);

        /*
         *  It is not mandatory for the calling task to catalog
         *  the mailbox token to send a message.  It is necessary,
         *  however, to catalog (or communicate) the mailbox
         *  token if another task is to receive the message.
         */

    rq_catalog_object (job, mailbox, name, &status);


        /*
         *  The calling task invokes send_data to send a message
         *  to the specified mailbox.
         */

    rq_send_data (mailbox, &send_message, (UINT_16) 18, &status);

}



    /*
     *  In this example, the calling task looks up the token
     *  for the mailbox prior to invoking receive_data.
     */

receive_mail ()
{

    UINT_16         actual;
    SELECTOR        job;
    SELECTOR        mailbox;
    char            name[] = {3,"MBX"};
    UINT_16         time_limit;
    char            receive_message[BUF_SIZE];
    UINT_16         status;
```

```
        /*
         *  Set up the parameters for look up.
         */

    job         =   NULL_TOKEN;
    time_limit  =   WAIT_FOREVER;
    mailbox = rq_lookup_object (job, name, time_limit, &status);


        /*
         *  Knowing the token for the mailbox, the calling task
         *  can wait for a message from this mailbox by invoking
         *  receive_data.
         */

    actual = rq_receive_data (mailbox,
                              &receive_message,
                              time_limit,
                              &status);

}




main ()
{

    start_mail ();

}
```

rq_receive_message example

```
/*
 *      "C" examples for
 *      rq_receive_message
 *      rq_send_message
 *      rq_lookup_object
 *      rq_catolog_object
 *      rq_create_segment
 *      rq_create_mailbox
 */


    /*
     *  nucleus prototype defines
     */


#include <rmx_c.h>
#include <rmx_err.h>
#include <rmx_def.h>


    /*
     *  Task to use rq_send_message.
     */


send ()
{

    UINT_16         mailbox_flags;
    UINT_32         seg_size;
    SELECTOR        job;
    SELECTOR        mailbox;
    SELECTOR        segment;
    SELECTOR        no_response;
    char            name[] = {3,"MBX"};
    UINT_16         time_limit;
    UINT_16         status;
```

```
        /*
         *  The calling task creates a segment and a mailbox
         *  and catalogs the mailbox token.  The calling task then
         *  uses the tokens for both objects to send a message.
         */


    seg_size        =   64;
    mailbox_flags   =   FIFO_QUEUING;
    no_response     =   NULL_TOKEN;
    job             =   NULL_TOKEN;

    segment = rq_create_segment (seg_size, &status);
    mailbox = rq_create_mailbox (mailbox_flags, &status);


        /*
         *  It is not mandatory for the calling task to catalog
         *  the mailbox token to send a message.  It is necessary,
         *  however, to catalog (or communicate) the mailbox
         *  token if another task is to receive the message.
         */


    rq_catalog_object (job, mailbox, name, &status);


        /*
         *  The calling task invokes send_message to send the
         *  token for the segment to the specified mailbox.
         */

    rq_send_message (mailbox, segment, no_response, &status);
}
```

```
    /*
     *  In this example the calling task looks up the token
     *  for the mailbox prior to invoking receive_message.
     */

receive ()
{

    SELECTOR        job;
    SELECTOR        mailbox;
    SELECTOR        object;
    SELECTOR        response;
    char            name[] = {3,"MBX"};
    UINT_16         time_limit;
    UINT_16         status;

        /*
         *  The calling task creates a segment and a mailbox
         *  and catalogs the mailbox token.  The calling task then
         *  uses the tokens for both objects to send a message.
         */

    job = NULL_TOKEN;
    mailbox = rq_lookup_object (job, name, time_limit, &status);

        /*
         *  Knowing the token for the mailbox, the calling task
         *  can wait for a message from this mailbox by invoking
         *  receive_message.
         */

    object = rq_receive_message (mailbox,
                                 time_limit,
                                 (SELECTOR far *)&response,
                                 &status);

}

main ()
{

    send ();
}
```

rq_receive_units example

```
/*
 *      "C" examples for
 *      rq_receive_units
 *      rq_send_units
 *      rq_lookup_object
 *      rq_catolog_object
 *      rq_create_semaphore
 */

    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>
#include <rmx_def.h>


    /*
     *  Send semaphore units.
     */

send_sema ()
{

    UINT_16         units;
    UINT_16         semaphore_flags;
    UINT_16         initial_value;
    UINT_16         max_value;
    SELECTOR        job;
    SELECTOR        semaphore;
    char            name[] = {5,"SEMA4"};
    UINT_16         time_limit;
    UINT_16         status;


        /*
         *  The calling task creates a semaphore and catalogs the
         *  semaphore token.  The calling task then uses the token
         *  to send a unit.
         */
    initial_value   =   1;
    max_value       =   0x10;
    semaphore_flags =   0;
    semaphore = rq_create_semaphore  (initial_value,
                                      max_value,
                                      semaphore_flags,
                                      &status);


        /*
         *  It is not mandatory to catalog the semaphore token in
         *  order to send units.  It is necessary, however, to
         *  catalog (or communicate) the semaphore token
         *  if another task is to receive the units.
         */

    job = NULL_TOKEN;
    rq_catalog_object (job, semaphore, name, &status);


        /*
         *  The calling task invokes send_units to send the units
         *  to the semaphore just created (sem_token.)
         */

    units = 3;
    rq_send_units (semaphore, units, &status);

}




    /*
     *  In this example, the calling task looks up the token
     *  for the semaphore prior to invoking receive_units.
     */
```

```
receive_sema ()
{

    UINT_16         units;
    UINT_16         value;
    SELECTOR        job;
    SELECTOR        semaphore;
    char            name[] = {5,"SEMA4"};
    UINT_16         time_limit;
    UINT_16         status;

        /*
         *  The calling task creates a semaphore and catalogs the
         *  semaphore token.  The calling task then uses the token
         *  to send a unit.
         */

    job         =   NULL_TOKEN;
    time_limit  =   WAIT_FOREVER;
    semaphore = rq_lookup_object (job, name, time_limit,
                               &status);

        /*
         *  Knowing the token for the semaphore, the calling task
         *  can wait for units at this semaphore by invoking
         *  receive_units.
         */

    units = 4;
    value = rq_receive_units (semaphore, units, time_limit,
                           &status);

}



main ()
{

    send_sema ();
    receive_sema ();

}
```

rqe_set_os_extension example

```
/*
 *      "C" examples for rq_set_os_extension
 */

    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>

    /*
     *  Bind your entry_440 to this demo.
     */

extern void entry_440 ();

    /*
     *  Main task to use set os extension.
     */

main ()
{

    UINT_16         gate_number;
    UINT_16         status;

        /*
         *  Rqe_set_os_extension sets the call gate used by
         *  an OS extension.  The example assumes the gate number
         *  was reserved.
         *  The calling task invokes rqe_set_os_extension to set
         *  the call gate at entry 440 in the GDT.  The entry point
         *  address is also specified.
         */

    gate_number = 440;
    rqe_set_os_extension (gate_number,
                          (void (far*)(void)) entry_440,
                          &status);
}
```

rq_set_pool_min example

```
/*
 *      "C" examples for rq_set_pool_min
 */

    /*
     *  nucleus prototype defines
     */

#include <rmx_c.h>
#include <rmx_err.h>

    /*
     *  Main task to use set pool minimum.
     */

main ()
{

    UINT_16         new_min;
    UINT_16         status;


        /*
         *  Sets pool_min attribute of calling task's job equal
         *  to job's pool_max attribute.
         */

    new_min = 0xffff;

        /*
         *  In this example the pool_min attribute of the
         *  calling task's job is to be set equal to that job's
         *  pool_max attribute.
         */

    rq_set_pool_min ((UINT_32) new_min, &status);

}
```

□ □ □

UDI Examples C

dq_create example

```
/*
 *     "C" examples for
 *     dq_create
 *     dq_open
 *     dq_read
 *     dq_switch_buffer
 *     dq_get_argument
 *     dq_exit
 *
 */


    /*
     *  prototype defines
     */

#include <udi_c.h>
#include <rmx_err.h>
#include <rmx_def.h>

#define  SINGLE_BUFFER 0


    /*
     * Main routine to use dq_switch_buffer to check the
     * buffer location of the current argument.
     */
```

```
main ()
{

    char            buffer[1024];
    char *          buff_ptr;
    char            arg[1024];
    SELECTOR        co_conn;
    SELECTOR        ci_conn;
    char            delimit_char;
    UINT_16         bytes_read;
    UINT_16         next_arg_pos;
    UINT_16         char_offset;
    char            ci_name [] = {4,":CI:"};
    UINT_16         status;


        /*
         * Intialize variables and buffer pointer.
         */

    next_arg_pos = 0;


        /*
         * Open input and read the command line from the console.
         */

    ci_conn = dq_create (ci_name, &status);
    dq_open (ci_conn,
             (UINT_8) READ_ONLY,
             (UINT_8) SINGLE_BUFFER,
             &status);
    bytes_read = dq_read (ci_conn,
                          (UINT_8 far *) buffer,
                          (NATIVE_WORD) 80,
                          &status);


           /*
            * Switch  command line buffers and  get arguments from a
            * buffer.
            */
    char_offset = dq_switch_buffer ((UINT_8 far*) buffer,
                                    &status );
    if (status != E_OK)
        dq_exit ((UINT_16) E_FATAL_EXIT);

    delimit_char = dq_get_argument ((STRING far *) &arg, &status);
    if (status != E_OK)
        dq_exit ((UINT_16) E_FATAL_EXIT);


        /*
         *  Determine where next argument starts.
         */

    char_offset = dq_switch_buffer ((UINT_8 far *) buffer,
                                    &status);
    if (status != E_OK)
        dq_exit ((UINT_16) E_FATAL_EXIT);

    next_arg_pos = char_offset + next_arg_pos;


        /*
         * Return to desired point in buffer.
         */

    buff_ptr    =   buffer;
    buff_ptr    +=  char_offset;
    char_offset = dq_switch_buffer ((UINT_8 far *) buff_ptr,
                                    &status);

        /*
         * Continue processing arguments.
         */

    delimit_char = dq_get_argument ((STRING far *) &arg,
                                    &status);
    if (status != E_OK)
        dq_exit ((UINT_16) E_FATAL_EXIT);

    dq_exit((UINT_16) E_OK);

}
```

```
/*
 *  UPPER - UDI2.C
 *
 *  This program demonstrates the use of UDI file-handling and
 *  command-line-parsing system calls.  The program reads an input
 *  file of characters and converts all lowercase alphabetic
 *  characters to uppercase.  The converted data are written to a
 *  second file. UPPER expects the command line that invokes it to
 *  be of the form: UPPER infile [TO outfile] (If "TO outfile" is
 *  not specified, :CO: is assumed.)
 */

    /*
     *  prototype defines
    */

#include <udi.h>
#include <rmx_err.h>

#define BUF_SIZE     1024

SELECTOR co_conn;


    /*
     * Procedure to check an exception code.  If the exception
     * code is not E_OK, print a message and exit.
     */

check_exception (UINT_16 exception, char * info_ptr)
{

    UINT_16         check_status;
    char            exc_buf[90];
    char            colon [] = {2,": "};
    char            crlf [] = {2, CR, LF};
    UINT_32         count;
    char *          buf_ptr;

    if (exception != E_OK)
    {
        dq_decode_exception (exception, exc_buf, &check_status);
        count   =   (UINT_32) exc_buf[0];
        buf_ptr =   &exc_buf[1];
        dq_write (co_conn,
                  (UINT_8 far*) buf_ptr,
                  count,
                  &check_status);

        count   =   2;
        buf_ptr =   colon;
        dq_write (co_conn,
                  (UINT_8 far*) buf_ptr,
                  count,
                  &check_status);

        count   =   (UINT_32) (info_ptr[0]);
        buf_ptr =   &info_ptr[1];
        dq_write (co_conn,
                  (UINT_8 far*) buf_ptr,
                  count,
                  &check_status);

        count   =   2;
        buf_ptr =   crlf;
        dq_write (co_conn,
                  (UINT_8 far*) buf_ptr,
                  count,
                  &check_status);

        dq_exit ((UINT_16) E_FATAL_EXIT);
    }
}



#define SINGLE_BUFFER  0
#define DOUBLE_BUFFER  1
```

```
   /*
     * The main routine to demo file handling
     * and command line parsing.
     */

main ()
{

    UINT_16         status;
    UINT_16         delim;
    UINT_16         in_count;
    UINT_16         i;
    UINT_16         not_done;
    char            in_name[50];
    char            out_name[50];
    char            buffer[BUF_SIZE];
    UINT_32         count;
    SELECTOR        in_conn;
    SELECTOR        out_conn;
    char            co_name [] = {4,":CO:"};
    char            invalid_msg [] = {"Invalid output file",CR, LF};


        /*
         * Create a connection to :CO:  (console output).
         */

    co_conn = dq_create (co_name, &status);
    dq_open (co_conn,
             (UINT_8) WRITE_ONLY,
             (UINT_8) SINGLE_BUFFER,
             &status);


        /*
         * Ignore the name of the program (the first argument).
         */

    delim = dq_get_argument (buffer, &status);
    check_exception (status, NULL);
    if (delim == CR) dq_exit ((UINT_16) E_OK);
```

```
        /*
         * Attach the input file, and open it.  Get the name of the
         * input file from the next comand line.
         */

    delim = dq_get_argument (in_name, &status);
    check_exception (status, NULL);

    in_conn = dq_attach (in_name, &status);
    check_exception (status, in_name);

    dq_open (in_conn,
             (UINT_8) READ_ONLY,
             (UINT_8) DOUBLE_BUFFER,
             &status);
    check_exception (status, in_name);

        /*
         * Find out if there is an output file specified.  If so,
         * attach and open it.  If not, use :CO: for output.
         */

    if (delim != CR)
    {
        delim = dq_get_argument (buffer, &status);
        check_exception (status, NULL);

        if ((delim == CR)       || (buffer[0] != 2) ||
            (buffer[1] != 'T')  || (buffer[2] != 'O'))
        {
            count = 21;
            dq_write (co_conn,
                      (UINT_8 far *) invalid_msg,
                      count,
                      &status);
            dq_exit ((UINT_16) E_FATAL_EXIT);
        }

        delim = dq_get_argument (out_name, &status);
        check_exception (status, NULL);

        out_conn = dq_create (out_name, &status);
        check_exception (status, out_name);
        dq_open (out_conn,
                 (UINT_8) WRITE_ONLY,
                 (UINT_8) DOUBLE_BUFFER,
                 &status);
        check_exception (status, out_name);
    }


        /*
         * Write to :CO: if no file specified.
         */

    else out_conn = co_conn;


        /*
         * Read from input, convert, and write to output.
         */

    not_done = TRUE;
    while (not_done)
    {
        in_count = dq_read (in_conn,
                            (UINT_8 far *) buffer,
                            BUF_SIZE,
                            &status);
        check_exception (status, in_name);



            /*
             *  If no characters are in the file,
             *  then fail next test.
             */

        if (in_count == 0) not_done = FALSE;


            /*
             * If characters are in the file, then process them.
             */
```

```
       if (not_done == TRUE)

        {
            for (i=0; i < in_count; i++)
            {
                if ((buffer[i] >= 'a') && (buffer[i] <= 'z'))
                {
                    buffer[i] = buffer[i] - 0x20;
                }
            }
        }

        dq_write (out_conn,
                  (UINT_8 far *) buffer,
                  (NATIVE_WORD) in_count,
                  &status);
        check_exception (status, out_name);
    }


        /*
         * Close input and output files, and exit.
         */

    dq_close (in_conn, &status);
    check_exception (status, in_name);

    dq_close (out_conn, &status);
    check_exception (status, out_name);

    dq_exit ((UINT_16) E_OK);
}
```

□ □ □

Condition Codes D

This appendix provides a list of the iRMX condition codes that can be returned from iRMX system calls. The condition codes are divided into two categories:

- Programmer errors
- Environmental conditions

A programmer error is a condition, such as a syntax error, that can be changed in the application code. An environmental condition is an operating system problem over which you do not have direct control.

This appendix lists the condition codes by operating system layer and by ascending numeric values. Each entry includes the condition code mnemonic, the numeric value, and a brief description.

See also: Individual call descriptions in this manual
Network User's Guide and Reference for condition codes returned by iNA 960 **cq_** calls

Environmental Conditions

E_OK	0H	The last system call that returned a status was successful.

Nucleus Environmental Conditions

E_TIME	01H	A time limit (possibly 0) expired without a task's request being satisfied.
E_MEM	02H	There is not sufficient memory available to satisfy a task's request.
E_BUSY	03H	Another task currently has access to the data protected by a region.
E_LIMIT	04H	A task attempted an operation which, if successful, would have violated a Nucleus-enforced limit.

E_CONTEXT	05H	A system call was issued out of context or the operating system was asked to perform an impossible operation.
E_EXIST	06H	A token parameter has a value which is not a valid token.
E_STATE	07H	A task attempted an operation which would have caused an impossible transition of a task's state.
E_NOT_CONFIGURED	08H	This system call is not part of the present configuration.
E_INTERRUPT_SATURATION	09H	An interrupt task has accumulated the maximum allowable number of **signal_interrupt** requests.
E_INTERRUPT_OVERFLOW	0AH	An interrupt task has accumulated more than the maximum allowable amount of **signal_interrupt** requests.
E_TRANSMISSION	0BH	A NACK, timeout, or bus error occurred.
E_SLOT	0CH	There are no available GDT slots.
E_DATA_CHAIN	0DH	A data chain has been returned. The token points to the beginning of the data chain block.

I/O System Environmental Conditions

E_FEXIST	20H	The specified file already exists.
E_FNEXIST	21H	The specified file does not exist.
E_DEVFD	22H	The device driver and file driver are incompatible.
E_SUPPORT	23H	The combination of parameters entered is not supported.
E_EMPTY_ENTRY	24H	The specified entry in a directory file is empty.
E_DIR_END	25H	The specified directory entry index is beyond the end of the directory file.

E_FACCESS	26H	The connection does not have the correct access to the file.
E_FTYPE	27H	The requested operation is not valid for this file type.
E_SHARE	28H	The requested operation attempted an improper kind of file sharing, or the file does not allow sharing.
E_SPACE	29H	There is no space left on the volume.
E_IDDR	2AH	An invalid device driver request occurred.
E_IO	2BH	An I/O error occurred.
E_FLUSHING	2CH	The connection specified in the call was deleted before the operation completed.
E_ILLVOL	2DH	The device contains an invalid or improperly formatted volume.
E_DEV_OFFLINE	2EH	The device being accessed is now offline.
E_IFDR	2FH	An invalid file driver request occurred.
E_FRAGMENTATION	30H	The volume is too fragmented for a file to be extended.
E_DIR_NOT_EMPTY	31H	The call is attempting to delete a directory that is not empty.
E_NOT_FILE_CONN	32H	The specified connection is not a file connection.
E_NOT_DEVICE_CONN	33H	The specified connection is not a device connection.
E_CONN_NOT_OPEN	34H	The connection is not open for reading, writing, or updating.
E_CONN_OPEN	35H	The task attempted to open a connection that is already open.
E_BUFFERED_CONN	36H	The specified connection was opened by the EIOS and used by the BIOS, which is not allowed.
E_OUTSTANDING_CONNS	37H	A soft detach was specified, but connections to the device still exist.
E_ALREADY_ATTACHED	38H	The specified device is already attached.

E_DEV_DETACHING	39H	The file specified is on a device that the operating system is in the process of detaching.
E_NOT_SAME_DEVICE	3AH	The existing pathname and the new pathname refer to different devices. You cannot simultaneously rename a file and move it to another device.
E_ILLOGICAL_RENAME	3BH	The call is attempting to rename a directory to a new path containing itself.
E_STREAM_SPECIAL	3CH	A stream file request is out of context. Either it is a query request and another query request is already queued, or it is a satisfy request and the request queue is empty or a query request is queued.
E_INVALID_FNODE	3DH	The connection refers to a file with an invalid fnode. Delete this file.
E_PATHNAME_SYNTAX	3EH	The specified pathname contains invalid characters.
E_FNODE_LIMIT	3FH	One of these: The volume already contains the maximum number of files and no more fnodes are available for new files. or The file cannot be created or extended to this size because it has reached the maximum number of volume blocks available for a file.
E_LOG_NAME_SYNTAX	40H	The specified pathname starts with a colon (:), but it does not contain a second, matching colon; or the specified logical has more than 12 characters or contains invalid characters.
E_CANNOT_CLOSE	41H	The buffers cannot be written to the device to complete the I/O request.
E_IOMEM	42H	The BIOS has insufficient memory to process a request.
E_MEDIA	44H	The device containing a specified file is not on line.

E_LOG_NAME_NEXIST	45H	The specified path contains an explicit logical name, but the EIOS was unable to find the name in the object directories of the local job, the global job, or the root job.
E_NOT_OWNER	46H	The user who attempted to detach the device is not the owner of the device.
E_IO_JOB	47H	The EIOS could not create an I/O job because the default directory size (DDS) configuration parameter is too small.
E_UDF_FORMAT	48H	The user definition file (UDF) is not in the right format.
E_NAME_NEXIST	49H	The user name specified in the call is not listed in the UDF.
E_UID_NEXIST	4AH	The user ID in the specified user object does not match the ID listed in the UDF for the corresponding user name.
E_PASSWORD_MISMATCH	4BH	The password specified in the call does not match the one listed in the UDF for the corresponding user name.
E_UDF_IO	4CH	The UDF specified cannot be found. An error code came from a remote UDF and not another remote file.
E_IO_UNCLASS	50H	An unknown type of I/O error occurred.
E_IO_SOFT	51H	A soft I/O error occurred. A retry might be successful.
E_IO_HARD	52H	A hard I/O error occurred. A retry is probably useless.
E_IO_OPRINT	53H	The device was off-line. Operator intervention is required.
E_IO_WRPROT	54H	The volume is write-protected.
E_IO_NO_DATA	55H	A tape drive attempted to read the next record, but it found no data.
E_IO_MODE	56H	A tape drive attempted a read (write) operation before the previous write (read) completed.

E_IO_NO_SPARES	57H	No spare tracks/sectors.
E_IO_ALT_ASSIGNED	58H	Alternate track/sector was assigned.

Application Loader Environmental Conditions

E_BAD_HEADER	62H	The object file contains an invalid header record.
E_EOF	65H	The Application Loader encountered an unexpected EOF file while reading a record.
E_NO_LOADER_MEM	67H	There is insufficient memory to satisfy the memory requirements of the AL.
E_NO_START	6CH	The AL could not find the start address.
E_JOB_SIZE	6DH	The maximum memory-pool size of the job being loaded is smaller than the amount of memory required to load its object file.
E_OVERLAY	6EH	The overlay name does not match any of the overlay module names.
E_LOADER_SUPPORT	6FH	The file requires features not supported by the AL as configured.

Human Interface Environmental Conditions

E_LITERAL	80H	The parsing buffer contains a literal with no closing quote.
E_STRING_BUFFER	81H	The string to be returned exceeds the size of the buffer the user provided in the call.
E_SEPARATOR	82H	The parsing buffer contains a command separator.
E_CONTINUED	83H	The parsing buffer contains a continuation character.
E_INVALID_NUMERIC	84H	A numeric value contains non-numeric characters.
E_LIST	85H	A value in the value list is missing.
E_WILDCARD	86H	A wild-card character appears in an invalid context, such as in an intermediate component of a pathname.

E_PREPOSITION	87H	The command line contains an invalid preposition.
E_PATH	88H	The command line contains an invalid pathname.
E_CONTROL_C	89H	The user typed a <Ctrl-C> to abort the command.
E_CONTROL	8AH	The command line contains an invalid control character.
E_UNMATCHED_LISTS	8BH	The number of files in the input and output pathname lists is not the same.
E_INVALID_DATE	8CH	The operator entered an invalid date.
E_NO_PARAMETERS	8DH	A command expected parameters, but the operator didn't supply any.
E_VERSION	8EH	The HI is not compatible with the version of the command the operator invoked.
E_GET_PATH_ORDER	8FH	A command called **c_get_output_pathname** before calling **c_get_input_pathname**.
E_PERMISSION	90H	The user does not have permission to access the requested resource.
E_INVALID_TIME	91H	The operator entered an invalid time.

UDI Environmental Conditions

E_UNKNOWN_EXIT	0C0H	The program exited normally.
E_WARNING_EXIT	0C1H	The program issued warning messages.
E_ERROR_EXIT	0C2H	The program detected errors.
E_FATAL_EXIT	0C3H	A fatal error occurred in the program.
E_ABORT_EXIT	0C4H	The operating system aborted the program.
E_UDI_INTERNAL	0C5H	A UDI internal error occurred.

Nucleus Communications Service Environmental Conditions

E_CANCELLED	0E1H	A **send_rsvp** transaction has been remotely canceled.
E_HOST_ID	0E2H	The specified host ID does not refer to a board that is currently in message space.
E_NO_LOCAL_BUFFER	0E3H	The buffer pool does not contain a buffer large enough to hold the message.
E_NO_REMOTE_BUFFER	0E4H	The remote port's buffer pool does not have a buffer large enough to hold the message and message fragmentation is turned off.
E_RESOURCE_LIMIT	0E6H	Either the number of simultaneous messages or simultaneous transactions has been reached. These fields are set during system configuration.
E_TRANS_ID	0E8H	The specified transaction ID is not valid.
E_DISCONNECTED	0E9H	The port sending the message has previously issued an **rq_connect** to a remote port. The board on which the remote port is located has been reset.
E_TRANS_LIMIT	0EAH	A transmission resource limitation has been encountered.

Paging Subsystem Environmental Conditions

E_VMEM	0F0H	Insufficient virtual memory available in the virtual segment to satisfy this request.
E_ALLOCATED	0F1H	Physical memory is already allocated to this area of the virtual segment.
E_NOT_ALLOCATED	0F2H	No physical memory is allocated to this area of the virtual segment.

Programmer Errors

Nucleus Programmer Errors

E_ZERO_DIVIDE	8000H	A task attempted a divide by zero.
E_OVERFLOW	8001H	An overflow interrupt occurred.
E_TYPE	8002H	A token referred to an existing object that is not of the required type.
EBOUNDS	8003H	A 16-bit address (offset) exceeds the 64 KB boundary.
E_PARAM	8004H	A parameter that is neither a token nor an offset has an invalid value.
E_BAD_CALL	8005H	An OS extension received an invalid function code.
E_ARRAY_BOUNDS	8006H	Hardware or software has detected an array overflow.
E_NDP_ERROR	8007H	An NPX error occurred. OS extensions can return the status of the NPX to the exception handler.
E_ILLEGAL_OPCODE	8008H	The processor tried to execute an invalid instruction.
E_EMULATOR_TRAP	8009H	An ESC instruction was encountered with the emulator bit set in the machine status word.
E_CHECK_EXCEPTION	800AH	A task has exceeded the bounds of a CASE statement.
E_CPU_XFER_DATA_LIMIT	800BH	The NPX tried to access an address that is out of segment boundaries.
E_PROTECTION	800DH	A general protection error occurred.
E_NOT_PRESENT	800EH	A request has been made to load a segment register whose segment is not present.
E_BAD_ADDR	800FH	The logical address is illegal. Either the selector does not point to a valid segment, or the offset is not within the segment boundaries.

I/O System Programmer Errors

E_NOUSER	8021H	No default user is defined.
E_NOPREFIX	8022H	No default prefix is defined.
E_BAD_BUFF	8023H	Illegal usage of memory buffers in read or write requests occurred.
E_NOT_LOG_NAME	8040H	The specified object is not a device connection or file connection.
E_NOT_DEVICE	8041H	A token referred to an existing object that is not, but should be, a device connection.
E_NOT_CONNECTION	8042H	A token referred to an existing object that is not, but should be, a file connection.

Application Loader Programmer Error

E_JOB_PARAM	8060H	The maximum memory pool size specified for the job is less than the minimum pool size specified.

Human Interface Programmer Errors

E_PARSE_TABLES	8080H	There is an error in the internal parsing tables.
E_JOB_TABLES	8081H	An internal HI table was overwritten, causing it to contain an invalid value.
E_DEFAULT_SO	8083H	The default output name string is invalid.
E_STRING	8084H	The pathname to be returned exceeds 255 characters in length.
E_ERROR_OUTPUT	8085H	The command invoked by **c_send_command** includes a call to **c_send_eo_response**, but the command connection does not permit **c_send_eo_response** calls.

UDI Programmer Errors

E_RESERVE_PARAM	80C6H	The calling program tried to reserve memory for more than 12 files or buffers.

E_OPEN_PARAM	80C7H	The calling program requested more than two buffers when opening a file.

Communication System Programmer Errors

E_PROTOCOL	80E0H	A signal port was specified instead of a data port, or vice versa.
E_PORT_ID_USED	80E1H	The port ID specifies a port that is in use.
E_NUC_BAD_BUF	80E2H	The specified pointer is invalid, or points to a buffer that is not large enough.

Paging Subsystem Programmer Errors

E_VSEG	80F0H	The calling task does not belong to the same job that created the virtual segment.
E_ALIGNMENT	80F1H	The address is not properly aligned, typically on a 4 Kbyte boundary.

□ □ □

Index

iRMX® System Call Reference
469157-004

WE'D LIKE YOUR OPINION

Please rate the following:	Excellent	Good	Fair	Poor
■ Manual organization	❐	❐	❐	❐
■ Technical accuracy	❐	❐	❐	❐
■ Completeness	❐	❐	❐	❐
■ Clarity of concepts and wording	❐	❐	❐	❐
■ Quality of examples and illustrations	❐	❐	❐	❐
■ Overall ease of use	❐	❐	❐	❐

Comments: ____

____

____

____

____

____

Please list any errors you found (include page number): ________________________

____

____

____

____

____

Name ____

Company Name ____

Address ____

____

May we contact you? ____________________ Phone ___________________________

Thank you for taking the time to fill out this form.

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

OPD Technical Publications, HF2-72
Intel Corporation
5200 NE Elam Young Parkway
Hillsboro, OR 97124-9978

Please fold here and close the card with tape. Do not staple.

WE'D LIKE YOUR COMMENTS....

This document is one of a series describing Intel products. Your comments on the other side of this form will help us produce better manuals. Each reply will be reviewed. All comments and suggestions become the property of Intel Corporation.

If you are in the United States and are sending only this card, postage is prepaid.

If you are sending additional material or if you are outside the United States, please insert this card and any enclosures in an envelope. Send the envelope to the above address, adding "United States of America" if you are outside the United States.

Thanks for your comments.

International Sales Offices

AUSTRALIA
Intel Australia Pty. Ltd.
Unit 1A
2 Aquatic Drive
Frenchs Forest, NSW, 2086
Sydney

Intel Australia Pty. Ltd.
711 High Street
1st Floor
East Kw. Vic., 3102
Melbourne

BRAZIL
Intel Semiconductores do Brazil LTDA
Avenida Paulista, 1159-CJS 404/405
CEP 01311-Sao Paulo - S.P.

CANADA
Intel Semiconductor of Canada, Ltd.
999 Canada Place
Suite 404, #11
Vancouver V6C 3E2
British Columbia

Intel Semiconductor of Canada, Ltd.
2650 Queensview Drive
Suite 250
Ottawa K2B 8H6
Ontario

Intel Semiconductor of Canada, Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Ontario

Intel Semiconductor of Canada, Ltd.
1 Rue Holiday
Suite 115
Tour East
Pt. Claire H9R 5N3
Quebec

CHINA/HONG KONG
Intel PRC Corporation
China World Tower, Room 517-518
1 Jian Guo Men Wai Avenue
Beijing, 100004
Republic of China

Intel Semiconductor Ltd.
32/F Two Pacific Place
88 Queensway
Central
Hong Kong

FINLAND
Intel Finland OY
Ruosilantie 2
00390 Helsinki

FRANCE
Intel Corporation S.A.R.L.
1, Rue Edison-BP 303
78054 St. Quentin-en-Yvelines
Cedex

GERMANY
Intel GmbH
Dornacher Strasse 1
85622 Feldkirchen bei Muenchen
Germany

INDIA
Intel Asia Electronics, Inc.
4/2, Samrah Plaza
St. Mark's Road
Bangalore 560001

ISRAEL
Intel Semiconductor Ltd.
Atidim Industrial Park-Neve Sharet
P.O. Box 43202
Tel-Aviv 61430

ITALY
Intel Corporation Italia S.p.A.
Milanofiori Palazzo E
20094 Assago
Milano

JAPAN
Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 300-26

Intel Japan K.K.
Hachioji ON Bldg.
4-7-14 Myojin-machi
Hachioji-shi, Tokyo 192

Intel Japan K.K.
Bldg. Kumagaya
2-69 Hon-cho
Kumagaya-shi, Saitama 360

Intel Japan K.K.
Kawa-asa Bldg.
2-11-5 Shin-Yokohama
Kohoku-ku, Yokohama-shi
Kanagawa, 222

Intel Japan K.K.
Ryokuchi-Eki Bldg.
2-4-1 Terauchi
Toyonaka-shi, Osaka 560

Intel Japan K.K.
Shinmaru Bldg.
1-5-1 Marunouchi
Chiyoda-ku, Tokyo 100

Intel Japan K.K.
Green Bldg.
1-16-20 Nishiki
Naka-ku, Nagoya-shi
Aichi 460

KOREA
Intel Korea, Ltd.
16th Floor, Life Bldg.
61 Yoido-dong, Youngdeungpo-Ku
Seoul 150-010

MEXICO
Intel Technologica de Mexico
S.A. de C.V.
Av. Mexico No. 2798-9B, S.H.
44620 Guadalajara, Jal.,

NETHERLANDS
Intel Semiconductor B.V.
Postbus 84130
3009 CC Rotterdam

RUSSIA
Intel Technologies, Inc.
Kremenchugskaya 6/7
121357 Moscow

SINGAPORE
Intel Singapore Technology, Ltd.
101 Thomson Road #08-03/06
United Square
Singapore 1130

SPAIN
Intel Iberia S.A.
Zurbaran, 28
28010 Madrid

SWEDEN
Intel Sweden A.B.
Dalvagen 24
171 36 Solna

TAIWAN
Intel Technology Far East Ltd.
Taiwan Branch Office
8th Floor, No. 205
Bank Tower Bldg.
Tung Hua N. Road
Taipei

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.
Pipers Way
Swindon, Wiltshire SN3 1RJ

If you need to contact Intel Customer Support

Contacting us is easy. Be sure that you have the following information available:

- Your phone and FAX numbers ready
- Complete description of your hardware or software configuration(s)
- Your product's product code
- Current version of all software you use
- Complete problem description

Type of Service	How to contact us	
FaxBACK* fax-on-demand system 24 hrs a day, 7 days a week	Using any touch-tone phone, have technical documents sent to your fax machine. Know your fax number before calling.	U.S. and Canada: (800) 628-2283 (916) 356-3105 Europe: +44-1793-496646
Intel PC and LAN Enhancement Support BBS 24 hrs a day, 7 days a week	Information on products, documentation, software drivers, firmware upgrades, tools, presentations, troubleshooting.	U.S and Canada: (503) 264-7999 Europe: +44-1793-432955 Autobaud detect 8 data bits, no parity, 1 stop
CompuServe* Information Service 24 hrs a day, 7 days a week	Worldwide customer support: information and technical support for designers, engineers, and users of 32-bit iRMX® OS and Multibus product families.	Worldwide Locations: (check your local listing) Type: GO INTELC once online.
Customer Support	Intel Multibus Support engineers offering technical advice and troubleshooting information on the latest Multibus products.	U.S. and Canada: (800) 257-5404 (503) 696-5025 FAX: (503) 681-8497 Hrs: M-F; 8-5 PST Europe: +44-1793-641469 FAX: +44-1793-496385 Hrs: M-F; 9-5:30 GMT
Hardware Repair	Multibus board and system repair.	U.S. and Canada: (800) 628-8686 (602) 554-4904 FAX: (602) 554-6653 Hrs: M-F; 7-5 PST Europe: +44-1793-403520 FAX: +44-1793-496156 Hrs: M-F; 9-5:30 GMT
Sales	Intel Sales engineers offering information on the latest iRMX and Multibus products and their availability.	Worldwide: Contact your local Intel office or distributor U.S. and Canada: (800) 438-4769 (503) 696-5025 FAX: (503) 681-8497 Hrs: M-F; 8-5 PST
Correspondence Mail letters to:	Worldwide: Intel Customer Support Mailstop HF3-55 5200 NE Elam Young Parkway Hillsboro, Oregon 97124-6497	Europe: European Application Support Intel Corporation (UK) Ltd. Pipers Way Swindon, Wiltshire England SN3 1RJ

* Third-party trademarks are the property of their respective owners.