intel

iRMX®
UDI System Calls
Reference Manual

Order Number: 462919-001

intel

iRMX®
UDI System Calls
Reference Manual

Order Number: 462919-001

Intel Corporation
3065 Bowers Avenue
Santa Clara, California 95051

Copyright © 1980, 1989, Intel Corporation, All Rights Reserved

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

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

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

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

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, or as defined in ASPR 7-104.9 (a) (9).

No part of this document may be copied or reproduced in any form or by any means without prior written consent of Intel Corporation.

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

Above	iLBX	iPSC	Plug-A-Bubble
BITBUS	i_m	iRMX	PROMPT
COMMputer	iMDDX	iSBC	Promware
CREDIT	iMMX	iSBX	QUEST
Data Pipeline	Insite	iSDM	QueX
Genius	int_el	iSSB	Ripplemode
i	Intel376	iSXM	RMX/80
i	Intel386	Library Manager	RUPI
I^2ICE	int_elBOS	MCS	Seamless
ICE	Intelevision	Megachassis	SLD
iCEL	$int_eligent$ Identifier	MICROMAINFRAME	UPI
iCS	$int_eligent$ Programming	MULTIBUS	VLSiCEL
iDBP	Intellec	MULTICHANNEL	376
iDIS	Intellink	MULTIMODULE	386
	iOSP	OpenNET	386SX
	iPDS	ONCE	
	iPSB		

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

Copyright© 1980, 1989, Intel Corporation. All Rights Reserved.

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

PREFACE

This manual describes Intel's Universal Development Interface as it applies to the iRMX® I and the iRMX II Operating Systems. The manual includes a brief introduction to the UDI and its relationship to the iRMX Operating Systems.

CONTENTS

iRMX® UDI System Calls

1.1 Introduction 1
1.2 Descriptions Of System Calls 3
1.3 UDI System Calls Dictionary 4
DQ$ALLOCATE 7
DQ$ATTACH 8
DQ$CHANGE$ACCESS 10
DQ$CHANGE$EXTENSION 13
DQ$CLOSE 15
DQ$CREATE 16
DQ$DECODE$EXCEPTION 17
DQ$DECODE$TIME 18
DQ$DELETE 20
DQ$DETACH 21
DQ$EXIT 22
DQ$FILE$INFO 24
DQ$FREE 28
DQGETARGUMENT 29
DQGETCONNECTION$STATUS 32
DQGETEXCEPTION$HANDLER 34
DQGETMSIZE 36
DQGETSIZE 37
DQGETSYSTEM$ID 38
DQGETTIME 39
DQ$MALLOCATE 40
DQ$MFREE 42
DQ$OPEN 43
DQ$OVERLAY 46
DQ$READ 48
DQ$RENAME 50
DQ$RESERVE$IO$MEMORY 52
DQ$SEEK 54
DQ$SPECIAL 56
DQ$SWITCH$BUFFER 59
DQ$TRAP$CC 62
DQ$TRAP$EXCEPTION 63
DQ$TRUNCATE 64
DQ$WRITE 65

Index

Tables

1. Standard UDI Condition Codes and Their Meanings 1
2. UDI System Calls Dictionary 4

1 iRMX® UDI SYSTEM CALLS

1.1 INTRODUCTION

This manual describes the requirements and behavior of UDI system calls in the iRMX® I and iRMX II Operating System environments.

Table 1. Standard UDI Condition Codes and Their Meanings

Hex Value	Mnemonic	UDI Calls	Meaning
0000H	E$OK	All but DQ$EXIT	No exceptional conditions.
0002H	E$MEM	DQ$ALLOCATE DQ$ATTACH DQ$CREATE DQ$OPEN DQ$RESERVE$IO$- MEMORY DQ$MALLOCATE	Insufficient memory for the requested operation.
0020H	E$FEXIST	DQ$RENAME	The specified file exists.
0021H	E$FNEXIST	DQ$ATTACH DQ$DELETE DQ$RENAME DQ$CHANGE$ACCESS	The specified file does not exist.

(continued)

Table 1. Standard UDI Condition Codes and Their Meanings (continued)

Hex Value	Mnemonic	UDI Calls	Meaning
0023H	E$SUPPORT	DQ$ATTACH DQ$CHANGE$ACCESS DQ$CREATE DQ$DECODE$TIME DQ$FILE$INFO DQGETCONNECTION$- STATUS DQ$OPEN DQ$OVERLAY DQ$READ DQ$RENAME DQ$RESERVEIOMEMORY DQ$SEEK DQ$SPECIAL DQ$TRUNCATE DQ$WRITE	An unsupported operation was attempted.
0026H	E$FACCESS	DQ$CHANGE$ACCESS DQ$DELETE DQ$OPEN	Access to the specified file is denied.
0028H	E$SHARE	DQ$OPEN	The specified file may not be shared.
0029H	E$SPACE	DQ$CREATE DQ$WRITE	The operation attempted to add a directory entry to a full directory.
0081H	E$STRING- $BUFFER	DQGETARGUMENT DQ$CHANGE$EXTENSION	A string is over 45 characters long or an argument is over 80 characters long.

1.2 DESCRIPTIONS OF SYSTEM CALLS

This section describes the individual UDI calls in detail. Immediately preceding the detailed descriptions, the UDI Call Dictionary (Table 2) arranges the calls in functional groups, and lists the page numbers of the more detailed descriptions.

Every system call description contains the following information in this order:

- The name of the system call.
- A brief summary of the function of the call.
- The form of the call as it is invoked from a PL/M program, with symbolic names for each parameter.
- Definition of input and output parameters.
- A complete explanation of the system call, including any information you will need to use it.
- Condition codes--a list of the error codes that can be incurred.

1.3 UDI SYSTEM CALLS DICTIONARY

Table 2. UDI System Calls Dictionary

UDI Call	Function Performed	Page
	PROGRAM CONTROL CALLS	
DQ$EXIT	Exits from the current application job.	22
DQ$OVERLAY	Causes the specified overlay to be loaded.	46
DQ$TRAP$CC	Captures control when CONTROL-C is typed.	62
	FILE-HANDLING CALLS	
DQ$ATTACH	Creates a connection to a specified file.	8
DQ$CHANGE$-ACCESS	Changes access rights associated with a file or directory.	10
DQ$CHANGE$-EXTENSION	Changes the extension of a file name in memory.	13
DQ$CLOSE	Closes the specified file connection.	15
DQ$CREATE	Creates a file for use by the application.	16
DQ$DELETE	Deletes a file.	20
DQ$DETACH	Closes a file and deletes its connection.	21
DQ$FILE$INFO	Returns data about a file connection	24
DQGETCON-NECTION$STATUS	Returns status of a file connection.	32
DQ$OPEN	Opens a file for a particular type of access.	43
DQ$READ	Reads the next sequence of bytes from a file.	48

Table 2. UDI System Calls Dictionary (continued)

UDI Call	Function Performed	Page
	FILE-HANDLING CALLS (continued)	
DQ$RENAME	Renames the specified file.	50
DQ$SEEK	Moves the current position pointer of a file.	54
DQ$SPECIAL	Sets terminal line-edit/transparent mode.	56
DQ$TRUNCATE	Truncates a file to the specified length.	64
DQ$WRITE	Writes a sequence of bytes to a file.	65
	MEMORY MANAGEMENT CALLS	
DQ$ALLOCATE	Requests a memory segment of a specified size.	7
DQ$FREE	Returns a memory segment to the system.	28
DQGETMSIZE	Returns the size of the specified memory block.	36
DQGETSIZE	Returns the size of the specified segment.	37
DQ$MALLOCATE	Requests a logically contiguous memory segment of a specified size.	40
DQ$MFREE	Returns memory allocated by DQ$MALLOCATE to the Free Space Pool.	42
DQ$RESERVE$-IO$MEMORY	Requests memory to be set aside for overhead to be incurred by I/O operations.	52

Table 2. UDI System Calls Dictionary (continued)

UDI Call	Function Performed	Page
	EXCEPTION-HANDLING CALLS	
DQ$DECODE$-EXCEPTION	Converts an exception numeric code into its equivalent mnemonic.	17
DQGETEXCEPT-ION$HANDLER	Returns a POINTER to the address of the program currently being used to process errors.	34
DQ$TRAP$-EXCEPTION	Identifies a custom exception processing program for a particular type of error.	63
	UTILITY AND COMMAND PARSING	
DQ$DECODE$-TIME	Returns system time and date in both binary and ASCII-character format	18
DQGETARGUMENT	Returns an argument from a STRING.	29
DQGET-SYSTEM$ID	Returns the identity of the environment for the UDI.	38
DQGETTIME	Obsolete: included for compatibility.	39
DQ$SWITCH$-BUFFER	Selects a new buffer from which to process commands.	59

DQ$ALLOCATE requests a memory segment from the free memory pool.

```
seg$t = DQ$ALLOCATE (size, except$ptr);
```

Input Parameter

size — A WORD which,

- if not zero, contains the size, in bytes, of the requested segment.
- if zero, indicates that the size of the request is 65536 (64K) bytes.

Output Parameters

seg$t — A TOKEN, into which the operating system places the base address of the memory segment. If the request fails because the memory requested is not available, this value will be undefined and the system will return an E$MEM exception code.

except$ptr — A POINTER to a WORD where the system places the condition code.

Description

The DQ$ALLOCATE system call is used to request additional memory from the free space pool of the program. Tasks may use the additional memory for any desired purpose.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$MEM	0002H	Insufficient memory to create a segment of the desired size.

In addition to the condition codes listed above, DQ$ALLOCATE can return the condition codes associated with the Nucleus system calls RQGETPOOL$ATTRIBUTES and RQ$CREATE$SEGMENT. See the *iRMX® II Nucleus System Calls Reference Manual or the iRMX I® Nucleus System Calls Reference Manual* for details.

DQ$ATTACH

The DQ$ATTACH system call creates a connection to an existing file.

```
connection$t = DQ$ATTACH (path$ptr, except$ptr);
```

Input Parameter

path$ptr	A POINTER to a STRING containing the pathname of the file to be attached.

Output Parameters

connection$t	A TOKEN for the connection to the file.
except$ptr	A POINTER to a WORD where the system places the condition code.

Description

This system call allows a program to obtain a connection to any existing file. When the DQ$ATTACH call returns a connection, all existing connections to the file remain valid.

Your program can use the DQ$RESERVE$IO$MEMORY call to reserve memory that the UDI can use for its internal data structures when the program calls DQ$ATTACH and for buffers when the program calls DQ$OPEN. The advantage of reserving memory is that the memory is guaranteed to be available when needed. If memory is not reserved, a call to DQ$ATTACH might not be successful because of a memory shortage. See the description of DQ$RESERVE$IO$MEMORY later in this chapter for more information about reserving memory.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$FNEXIST	0021H	The specified file does not exist.
E$MEM	0002H	Insufficient memory for the requested operation.
E$SUPPORT	0023H	An unsupported operation was attempted.

In addition to the condition codes listed above, DQ$ATTACH can return the exception codes associated with the Extended I/O System call RQSATTACH$FILE. See the *iRMX® Extended I/O System Calls Reference Manual* for details.

The DQ$CHANGE$ACCESS enables you to change the access rights of the owner of a file (or directory), or the access rights of the WORLD user.

```
CALL DQ$CHANGE$ACCESS (path$ptr, user, access, except$ptr);
```

Input Parameters

path$ptr — A POINTER to a STRING containing a pathname of the file.

user — A BYTE specifying the user whose access is to be changed.

Value	User
0	Owner of the file
1	WORLD (all users on the system)
2	GROUP (ignored by iRMX II Operating System
3-255	Reserved

If you specify a value of 3-255, an E$SUPPORT exception will be returned.

access — A BYTE specifying the type of access to be granted the user. This WORD is to be encoded as follows. (Bit 0 is the low-order bit.)

Bit	Meaning
0	User can delete the file or directory
1	Read (the file) or List (the directory)
2	Append (to the file) or Add entry (to the directory)
3	Update (read and write to the file) or Change Access (to the directory)
4	User can execute the file. Set to the value of bit one for compatibility with other operating systems.
5-7	Reserved. If you specify bits 5-7, an E$SUPPORT exception will be returned.

Output Parameter

except$ptr — A POINTER to a WORD where the system places the condition code.

Description

In the general iRMX environment, every program is associated with a user object, usually referred to as the default user for the program. The default user consists of one or more user IDs. Each file has an associated collection of user ID-access mask pairs, where each mask defines the access rights the corresponding user ID has to the file. When the program calls DQ$CREATE to create a file or DQ$ATTACH to get another connection to a file, the resulting connection receives all access rights corresponding to user IDs that are both associated with the file and in the default user. The purpose of the DQ$CHANGE$ACCESS system call is to change, for a particular file, the access rights associated with a particular user ID. This has the effect of changing the access granted when the program makes subsequent calls to DQ$ATTACH to get further connections to the file.

In the UDI subset of the iRMX environment, a default user has two IDs. One of them, called the owner ID, is associated with the program. The other, called the WORLD, is associated universally with all programs. DQ$CHANGE$ACCESS can change the access mask of either the owner ID or the WORLD.

Changing the access rights for a user ID has no effect on connections already obtained by the program. However, all subsequently obtained connections reflect the changed access rights.

For more information about user IDs, default users, access masks, WORLD, access rights, owner IDs, and how connections are related to all of these entities, refer to the *iRMX® Basic I/O System User's Guide.*

NOTE

DQ$CHANGE$ACCESS affects only connections made after the call is issued. It does not affect existing connections to the file.

Condition Codes

E$OK	000H	No exceptional conditions.
E$SUPPORT	0023H	The value specified for the user parameter is greater than two. You tried to set bits 5-7 of the access parameter.
E$FACCESS	0026H	Access to the specified file is denied.

In addition to the condition codes listed above, DQ$CHANGE$ACCESS can return the same condition codes as the Extended I/O System call RQSCHANGE$ACCESS. See the *iRMX® Extended I/O System Calls Reference Manual* for details.

DQ$CHANGE$EXTENSION changes or adds the extension at the end of a file name stored in memory (not the file name on the mass storage volume).

```
CALL DQ$CHANGE$EXTENSION (path$ptr, extension$ptr, except$ptr);
```

Input Parameters

path$ptr	A POINTER to a STRING containing a pathname of the file to be renamed.
extension$ptr	A POINTER to a series of three bytes containing the characters to be added to the pathname. This is not a STRING. You must include three bytes, even if some are blank.

Output Parameter

except$ptr	A POINTER to a WORD where the system places the condition code.

Description

This is a facility for editing strings that represent file names in memory. If the existing file name has an extension, DQ$CHANGE$EXTENSION replaces that extension with the specified three characters. Otherwise, DQ$CHANGE$EXTENSION adds the three characters as an extension.

For example, a compiler can use DQ$CHANGE$EXTENSION to edit a string containing the name, such as :AFD1:FILE.SRC, of a source file to the name, such as :AFD1:FILE.OBJ, of an object file, and then create the object file.

Note that iRMX file names may contain multiple periods, but if they do, the extension, if any, consists of the characters following the last period. Note also that an extension may contain more than three characters, but any extension created or changed by DQ$CHANGE$EXTENSION has at most three (non-blank) characters.

The three-character extension may not contain delimiters recognized by DQGETARGUMENT but may contain trailing blanks. If the first character pointed to by extension$ptr is a space, DQ$CHANGE$EXTENSION deletes the existing extension including the period, if any, preceding the extension.

Condition Codes

E$OK	000H	No exceptional conditions.
E$STRING$BUFFER	0081H	The filename is more than 14 characters (including the "." and extension, if any).

DQ$CLOSE waits for completion of I/O operations (if any) taking place on the file, empties the output buffers, and frees all buffers associated with the connection.

```
CALL DQ$CLOSE (connection$t, except$ptr);
```

Input Parameter

connection$t — A TOKEN for a file connection that is currently open.

Output Parameter

except$ptr — A POINTER to a WORD where the system places the condition code.

Description

The DQ$CLOSE system call closes a connection that has been opened by the DQ$OPEN system call. It performs the following actions, in order:

1. Waits until all currently running I/O operations for the connection are completed.
2. Ensures that information, if any, 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	No exceptional conditions.

In addition to the condition code listed above, DQ$CLOSE can return the same condition codes associated with the Extended I/O System call RQSCLOSE. See the *iRMX® Extended I/O System Calls Reference Manual* for details.

DQ$CREATE

DQ$CREATE creates a new file and establishes a connection to the file.

```
connection$t = DQ$CREATE (path$ptr, except$ptr);
```

Input Parameter

path$ptr	A POINTER to a STRING containing a pathname for the file to be created.

Output Parameters

connection$t	A TOKEN for the connection to the file.
except$ptr	A POINTER to a WORD where the system places the condition code.

Description

This call creates a new file with the name you specify and returns a connection to it. If a file of the same name already exists, it is truncated to a length of zero and the data in it is destroyed.

To prevent accidentally destroying a file, call DQ$ATTACH before calling DQ$CREATE. If the file does not exist, DQ$ATTACH returns an E$FNEXIST exception code.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$MEM	0002H	Insufficient memory remains to complete the call.
E$SPACE	0029H	Insufficient space exists on a direct-access device.
E$SUPPORT	0023H	An unsupported operation was attempted.

In addition to the condition codes listed above, DQ$CREATE can return the condition codes associated with the Extended I/O system calls RQSCREATE$FILE and RQ$S$DELETE$FILE. See the *iRMX® Extended I/O System Calls Reference Manual* for details.

DQ$DECODE$EXCEPTION translates an exception code into its mnemonic.

```
CALL DQ$DECODE$EXCEPTION (exception$code, buff$ptr, except$ptr);
```

Input Parameter

exception$code	A WORD containing the numeric exception code that is to be translated.

Output Parameters

buff$ptr	A POINTER to a STRING (at least 81 bytes long) into which the system returns the mnemonic.
except$ptr	A POINTER to a WORD where the system places the condition code.

Description

Your program can call DQ$DECODE$EXCEPTION to exchange a numeric exception code for its hexadecimal equivalent followed by its mnemonic. For example, if you pass DQ$DECODE$EXCEPTION a value of 2 in the except$code parameter, the system returns the following string to the area pointed to by the buff$ptr parameter:

```
0002H:  E$MEM
```

The hexadecimal values and mnemonics for condition codes are listed in Table 1. This system call can decode any iRMX exception code value. See the *Operator's Guide To The iRMX® Human Interface* for more details.

Condition Codes

E$OK	0000H	No exceptional conditions.

In addition to the condition code listed above, DQ$DECODE$EXCEPTION can return the condition codes associated with the Human Interface system call, RQCFORMAT$EXCEPTION. See the *iRMX® Human Interface System Calls Reference Manual* for details.

DQ$DECODE$TIME returns the current system time and date as ASCII date and time strings. You can also use DQ$DECODE$TIME to return the current time and date in binary format or as a decoded ASCII string.

```
CALL  DQ$DECODE$TIME (date$time$ptr, except$ptr);
```

Output Parameters

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

```
DECLARE DT STRUCTURE(
                SYSTEM$TIME DWORD,
                DATE (8)    BYTE,
                TIME (8)    BYTE);
```

where

SYSTEM$TIME is an operating-system-dependent DWORD containing the current time and date. To get the current time and date, the value in SYSTEM$TIME must be zero when the DQ$DECODE$TIME call is issued. To decode a binary time value, the time value must be stored in SYSTEM$TIME before making the call. (See the following Description section for format information.)

SYSTEM$TIME receives the time as the number of seconds since midnight, January 1, 1978.

DATE receives the date portion of the time, in the form of ASCII characters.

TIME receives the time-of-day portion of the time, in the form of ASCII characters.

If the value in SYSTEM$TIME is not 0 when DQ$DECODE$TIME is called, DQ$DECODE$TIME accepts that value as the number of seconds since midnight, January 1, 1978, decodes the value, and returns it in the DATE and TIME fields.

except$ptr — A POINTER to a WORD where the system places the condition code.

Description

This system call returns the indicated date and time, each as a series of ASCII bytes. (Note that they are not STRINGs.)

DATE has the form MM/DD/YY for month, day, and year. The two slashes (/) are in the third and sixth bytes. For example, the date January 15th of 1982 would be returned as

```
01/15/82
```

TIME has the form HH:MM:SS for hours, minutes, and seconds, with separating colons (:). The value for hours ranges from 0 through 23. For example, the time 20 seconds past 3:12 PM would be returned as

```
15:12:20
```

If, when you call DQ$DECODE$TIME, the SYSTEM$TIME parameter is zero, the call first gets the system time (number of seconds since midnight, January 1, 1978) and then decodes it into the series of bytes as just described.

But if SYSTEM$TIME is not zero on input, DQ$DECODE$TIME uses it as the time to decode.

One thing your program can do with DQ$DECODE$TIME is first to call DQ$FILE$INFO to get two DWORD values associated with a file (the last time the file was updated and the time the file was created). Then the program can call DQ$DECODE$TIME to interpret the times.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$SUPPORT	0023H	An unsupported operation was attempted.

In addition to the condition code listed above, DQ$DECODE$TIME can return the condition codes associated with the Basic I/O System call RQGETTIME, see the *iRMX® Basic I/O System Calls Reference Manual* for details.

DQ$DELETE

DQ$DELETE deletes an existing file.

```
CALL DQ$DELETE (path$ptr, except$ptr);
```

Input Parameter

path$ptr	A POINTER to a STRING containing a pathname of the file to be deleted.

Output Parameter

except$ptr	A POINTER to a WORD where the system places the condition code.

Description

A program can use this system call to delete a file. The immediate action this call takes is to mark the file for deletion. It does this rather than abruptly deleting the file, because it will not delete any file as long as there are existing connections to the file. DQ$DELETE will delete the file only when there are no longer any connections to the file, that is, when all existing connections have been detached. On the other hand, once the file is marked for deletion, no more connections may be obtained for the file by way of DQ$ATTACH.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$FNEXIST	0021H	The specified file does not exist.
E$FACCESS	0026H	Access to the specified file is denied.

In addition to the condition codes listed above, DQ$DELETE can return the condition codes associated with the Extended I/O System call RQSDELETE$FILE. See the *iRMX® Extended I/O System Calls Reference Manual* for details.

DQ$DETACH deletes a connection (but not the file) established by DQ$ATTACH or DQ$CREATE.

```
CALL DQ$DETACH (connection$t, except$ptr);
```

Input Parameter

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

Output Parameter

except$ptr — A POINTER to a WORD where the system places the condition code.

Description

This system call deletes a file connection. If the connection is open, the DQ$DETACH system call automatically closes it first (see DQ$CLOSE). DQ$DETACH also deletes the file if the file has been marked for deletion, and this is the last existing connection to the file. The results of specifying an invalid connection are operating-system-dependent.

Condition Codes

E$OK	0000H	No exceptional conditions.

In addition to the condition code listed above, DQ$DETACH can return the condition codes associated with the Universal Development Interface system call DQ$CLOSE and the Extended I/O system call RQSDELETE$CONNECTION. See the DQ$CLOSE system call in this manual, or the *iRMX® Extended I/O System Calls Reference Manual* for details.

DQ$EXIT transfers control from your program to the iRMX Operating System. It does not return any value to the calling program, not even a condition code.

```
CALL DQ$EXIT (completion$code);
```

Input Parameter

completion$code — A WORD containing the encoded reason for termination of the program. See the following description for information about this value.

Description

DQ$EXIT terminates a program. Before the actual termination, all of the program's connections are closed and detached, and all memory allocated to the program by DQ$ALLOCATE is returned to the memory pool.

DQ$EXIT does not return a condition code to the calling program.

If the calling program is running as an I/O job, the calling task, normally the command line interpreter (CLI), receives an iRMX condition code based on the value your program supplied in the completion$code field when it called DQ$EXIT. This assumes the following sequence of events:

1. The CLI calls RQ$CREATE$IO$JOB, specifying a response mailbox in the call.
2. Your program, running as a task in the created I/O job, performs its duties and then calls DQ$EXIT, specifying an completion$code value.
3. DQ$EXIT converts the completion$code value into an iRMX condition code, as follows:

completion-$code Value	iRMX Condition Code	Associated Mnemonic	Meaning
0	0000H	E$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 of termination not known.

4. DQ$EXIT calls RQ$EXITIOJOB, specifying the iRMX condition code in the user$fault$code field.
5. RQ$EXIT$IO$JOB places the condition code into the user$fault$code field of a message. Then RQ$EXIT$IO$JOB sends the message to the response mailbox set up by the earlier call to RQ$CREATE$IO$JOB.
6. The CLI, when it obtains the message from the response mailbox, can take appropriate actions. Note that it can call DQ$DECODE$EXCEPTION first, to convert the condition code into its associated mnemonic.

The CLI program supplied with the iRMX Operating Systems ignores these UDI condition codes when they are returned in the user$fault$code field of the response message. These condition codes are ignored because the UDI is not required to be in iRMX Operating Systems, so the iRMX CLI assumes that it is not. Therefore, if you want the CLI to take actions based on that code, you must provide your own CLI.

For more information about RQ$CREATE$IO$JOB and RQ$EXITIOJOB see the *iRMX® Extended I/O System Calls Reference Manual*; for more information on the format of the response message, see the *iRMX Extended I/O System User's Guide*.

DQ$FILE$INFO returns information about a file.

```
CALL  DQ$FILE$INFO (connection$t, mode, file$info$ptr, except$ptr);
```

Input Parameters

connection$t — A TOKEN containing a connection for the file.

mode — An encoded BYTE specifying whether DQ$FILE$INFO is to return the User ID of the owner of the file. Encode as follows:

Value	Meaning
0	Do not return owner's User ID.
1	Return the owner's User ID.
2-255	Return E$SUPPORT exception.

Output Parameters

file$info$ptr — A POINTER to a structure into which the requested information is to be returned. The form of the structure is

```
DECLARE FILE$INFO STRUCTURE(
                          OWNER(15)       BYTE,
                          LENGTH          DWORD,
                          TYPE            BYTE,
                          OWNER$ACCESS    BYTE,
                          WORLD$ACCESS    BYTE,
                          CREATE$TIME     DWORD,
                          LAST$MOD$TIME   DWORD,
                          GROUP$ACCESS    BYTE,
                          RESERVED(19)    BYTE);
```

where

OWNER — A STRING containing (if requested) the User ID of the file's owner.

LENGTH — A DWORD that gives the size of the file in bytes.

TYPE — A value indicating the type of file, as follows:

Value	File Type
0	Data file
1	Directory file
2	System-specific file
3-255	Reserved

OWNER$ACCESS — An encoded BYTE whose bits specify the type of access granted to the owner, as follows. When a bit is set, it means the type of access is granted; otherwise the type of access is denied. (Bit 0 is the low-order bit.)

Bit	Associated Access Type
0	Delete
1	Read (a data file) or Display (a directory)
2	Append (to a data file) or Add Entry (to the directory)
3	Update (read and write to a file) or Change Access (to the directory)
4	Execute the specified file. (Set to the value of bit 1 for compatibility with other operating systems.)
5-7	Reserved

WORLD$ACCESS — An encoded BYTE whose bits specify the type of access granted to the WORLD (all users on the system). When a bit is set, it means the type of access is granted; otherwise the type of access is denied. (Bit 0 is the low-order bit.)

Bit	Associated Type of Access
0	Delete
1	Read (a data file) or Display (a directory)
2	Write (to a data file) or Add Entry (to a directory)
3	Update (read and write to a file) or Change Access (to a directory)
4	Execute the specified file. (Set to the value of bit 1 for compatibility with other operating systems.)
5-7	Reserved

CREATE$TIME	The date and time that the file or directory was created, expressed as the number of seconds since midnight, January 1, 1978. (You can convert this date/time to ASCII characters by calling DQ$DECODE$TIME.)
LASTMODTIME	The date and time that the file or directory was last modified. For data files, modified means written to or truncated; for directories, modified means an entry was changed or an entry was added. (You can convert this date/time to ASCII characters by calling DQ$DECODE$TIME.)
GROUP$ACCESS	An encoded byte that is always set to the value of WORLD$ACCESS. The iRMX UDI does not use GROUP$ACCESS.

except$ptr — A POINTER to a WORD where the system places the condition code.

Description

The DQ$FILE$INFO system call returns information, as described above, about a data file or a directory file.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$SUPPORT	0023H	The mode parameter has a value greater than 1.

In addition to the condition codes listed above, DQ$FILE$INFO can return the condition codes associated with the Nucleus system calls RQ$CREATE$MAILBOX and RQ$RECEIVE$MESSAGE and the Basic I/O system call RQAGET$FILE$STATUS. See the *iRMX® II Nucleus System Calls Reference Manual* or the *iRMX® I Nucleus System Calls Reference Manual* and the *iRMX® Basic I/O System Calls Reference Manual* for details.

DQ$FREE

DQ$FREE returns to the system a segment of memory obtained earlier by DQ$ALLOCATE.

```
CALL DQ$FREE (seg$t, except$ptr);
```

Input Parameter

seg$t	A TOKEN containing the memory segment to be deleted. The TOKEN is returned by a DQ$ALLOCATE call and is no longer valid for this procedure once this call is made.

Output Parameter

except$ptr	A POINTER to a WORD where the system places the condition code.

Description

The DQ$FREE system call returns the specified segment to the memory pool from which it was allocated. A subsequent attempt to use this deleted segment may cause errors or unexpected results, since the memory may have been otherwise allocated.

Condition Codes

E$OK	0000H	No exceptional conditions.

In addition to the condition code listed above, DQ$FREE can return the condition codes associated with the Nucleus system call RQ$DELETE$SEGMENT. See the *iRMX® II Nucleus System Calls Reference Manual or the iRMX® I Nucleus System Calls Reference Manual* for details.

The DQGETARGUMENT system call returns arguments, one at a time, from a command line entered at the system console. This command line is either that which invoked the program containing the DQGETARGUMENT call or a command line entered while the program was running.

```
delimit$char =  DQ$GET$ARGUMENT (argument$ptr, except$ptr);
```

Input Parameter

argument$ptr	A POINTER to a STRING (at least 81 bytes long) that will receive the argument.

Output Parameters

delimit$char	A BYTE which receives the delimiter character.
except$ptr	A POINTER to a WORD where the system places the condition code.

Description

Your program can call GET$ARGUMENT to get arguments from a command line. Each call returns an argument and the delimiter character following the argument.

Your program can use this command in two ways. One way is to get arguments from the command line used to invoke the program at the console. In this case, you can assume that the command line is already in a buffer that has automatically been provided for this purpose.

The other way to use this command is to get arguments from command lines that are entered in response to requests from your program. In this case, your program must supply a buffer when calling DQ$READ. This is the buffer you want used when your program calls DQGETARGUMENT. To set this up, your program must call DQ$SWITCH$BUFFER before the call to DQGETARGUMENT.

A delimiter is returned only if the exception code is zero. The following delimiters are recognized by the iRMX Operating System:

```
, ) ( = # ! % + - & ; < > [ ] \ ' | ~
```

as well as a space ( ) and all characters with ASCII values in the range 0 through 20H, or between 7FH and 0FFH.

Before returning arguments in response to DQGETARGUMENT, the system does the following editing on the contents of the command buffer:

- It strips out ampersands (&) and semicolons (;).
- Where multiple blanks are adjacent to each other between arguments, it replaces them with a single blank. (Tabs are treated as blanks.)
- It converts lowercase characters to uppercase unless they are part of a quoted string.
- It treats the command line and the buffer (after a DQ$SWITCH$BUFFER system call) as if they were preceded by a null delimiter.

When returning arguments in response to DQGETARGUMENT, the system considers strings enclosed between matching pairs of single or double quotes to be literals. The enclosing quotes are not returned as part of the argument.

Example

The following example illustrates the arguments and delimiters returned by successive calls to DQGETARGUMENT. The example assumes that the contents of the buffer are

```
PLM286 LINKER.PLM PRINT(:LP:) NOLIST
```

The following shows what is returned if DQGETARGUMENT is called five times.

Call Number	Argument Returned	Delimiter Returned
1	(06H)PLM286	space
2	(0AH)LINKER.PLM	space
3	(05H)PRINT	(
4	(04H):LP:	)
5	(06H)NOLIST	cr

Note that the argument returned has the form of an iRMX string, with the first byte devoted to specifying the length of the string. In the second call, there are ten characters in the argument, so the first byte contains 0AH.

Note that the last delimiter for the example is a carriage return (cr). This is how your program can determine that there are no more arguments in the command line.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$STRING$BUFFER	0081H	An argument has been found that is longer than 80 characters. This only indicates that another call to DQGETARGUMENT is needed to obtain the rest of the argument.

DQGETCONNECTION$STATUS

The DQGETCONNECTION$STATUS system call returns information about a file connection.

```
CALL DQ$GET$CONNECTION$STATUS (connection$t, info$ptr, except$ptr);
```

Input Parameter

connection$t — A TOKEN containing the connection whose status is desired.

Output Parameters

info$ptr — A POINTER to a structure into which the operating system is to place the status information. The structure has the following format:

```
DECLARE INFO STRUCTURE(
                    OPEN        BYTE,
                    ACCESS      BYTE,
                    SEEK        BYTE,
                    FILE$PTR    DWORD);
```

where

OPEN — A Boolean that is 0FFH (TRUE) if the connection is open; 0000H (FALSE) otherwise.

ACCESS — Access privileges of the connection. The right is granted if the corresponding bit is set to 1. (Bit 0 is the low-order bit.)

Bit	Access
0	Delete
1	Read
2	Write
3	Update (read and write)
4	Execute (Set to the value of bit 1 for compatibility with other operating systems.)
5-7	Reserved

SEEK — Types of seek supported.

Value	Meaning
0	No seek allowed
3	Seek forward and backward

Other values are not meaningful.

FILE$PTR — This DWORD integer marks the current position in the file. The position is expressed as the number of bytes from the beginning of the file, the first byte being byte 0. 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 WORD where the system places the condition code.

Description

DQGETCONNECTION$STATUS returns information about a file. You might use this system call, for example, if your program has performed several read or write operations and you must determine where the file pointer is now located.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$SUPPORT	0023H	An unsupported operation was attempted.

In addition to the condition code listed above, DQGETCONNECTION$STATUS can return the condition codes associated with the Extended I/O system call RQSGET$CONNECTION$STATUS. See the *iRMX® Extended I/O System Calls Reference Manual* for details.

DQGETEXCEPTION$HANDLER returns the address of the current exception handler.

```
CALL DQ$GET$EXCEPTION$HANDLER (current$handler$ptr, except$ptr);
```

Output Parameters

current$handler$ptr	A POINTER to a STRUCTURE into which this system call returns the entry point of the current exception handler. This STRUCTURE has the same form as a long POINTER. DQ$TRAP$EXCEPTION specifies this entry point if it is called.
except$ptr	A POINTER to a WORD where the system places the condition code.

Description

DQGETEXCEPTION$HANDLER is a system call that returns the address of the current exception handler to your program. This is the address specified in the most recent call, if any, to DQ$TRAP$EXCEPTION. Otherwise, the value returned is the address of the system default exception handler.

This routine always returns a long POINTER, even if called from a program compiled under the SMALL model of segmentation. You can use this long POINTER in two ways:

- You can use it to make an indirect call to the current exception handler.
- After temporarily substituting another exception handler, you can use it to restore the current exception handler.

DQGETEXCEPTION$HANDLER is used in conjunction with DQ$TRAP$EXCEPTION and DQ$DECODE$EXCEPTION. See the descriptions of these calls for more information.

Condition Codes

E$OK	0000H	No exceptional conditions.

In addition to the condition code listed above, DQGETEXCEPTION$HANDLER can return the condition codes associated with the Nucleus system call RQGETEXCEPTION$HANDLER. See the *iRMX® II Nucleus System Calls Reference Manual or the iRMX® I Nucleus System Calls Reference Manual* for details.

DQGETMSIZE returns the size, in BYTES, of the memory block specified.

```
size = DQ$GET$MSIZE(seg$ptr, exception$ptr);
```

Input Parameter

seg$ptr	A POINTER that indicates an area of memory that was allocated earlier by a call to DQ$MALLOCATE.

Output Parameters

size	A DWORD which receives the size (in BYTES) of the memory block previously allocated by DQ$MALLOCATE.
exception$ptr	A POINTER to a WORD where the system call places the condition code.

Description

The DQGETMSIZE system call returns the size, in bytes, of a segment allocated by the DQ$MALLOCATE system call.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$SUPPORT	0023H	An unsupported operation was attempted.

In addition to the condition codes listed above, DQGETMSIZE can return the condition codes associated with the Nucleus system call RQGETSIZE. See the *iRMX® II Nucleus System Calls Reference Manual or the iRMX® I Nucleus System Calls Reference Manual* for details.

DQGETSIZE returns the size of a previously allocated memory segment.

```
size = DQ$GET$SIZE (seg$t, except$ptr);
```

Input Parameter

seg$t	A TOKEN for a segment of memory allocated by the DQ$ALLOCATE call.

Output Parameters

size	A WORD which, if not zero, contains the size, in bytes, of the segment identified by the seg$t parameter. if zero, indicates that the size of the segment is 65536 (64K) bytes.
except$ptr	A POINTER to a WORD where the system places the condition code.

Description

The GET$SIZE system call returns the size, in bytes, of a segment.

Condition Codes

E$OK	0000H	No exceptional conditions.

In addition to the condition code listed above, DQGETSIZE can return the condition codes associated with the Nucleus system call RQGETSIZE. See the *iRMX® II Nucleus System Calls Reference Manual or the iRMX® I Nucleus System Calls Reference Manual* for details.

DQGETSYSTEM$ID returns the identity of the operating system providing the environment for the UDI.

```
CALL DQ$GET$SYSTEM$ID (id$ptr, except$ptr);
```

Output Parameters

id$ptr	A POINTER to a 21-BYTE buffer into which DQGETSYSTEM$ID places a STRING identifying the operating system.
except$ptr	A POINTER to a WORD where the system places the condition code.

Description

This system call returns the strings

`RMX1` or `RMXII` indicating iRMX I or iRMX II respectively.

Condition Codes

E$OK	0000H	No exceptional conditions.

DQGETTIME returns the current date and time in character format. This procedure is obsolete.

```
CALL DQ$GET$TIME (date$time$ptr, except$ptr);
```

This system call is included only for compatibility with previous versions of the UDI. Use the more general DQ$DECODE$TIME system call for this function.

DQ$MALLOCATE

DQ$MALLOCATE requests that a specific amount of logically contiguous free memory be added to the existing memory available to the calling program.

```
seg$ptr = DQ$MALLOCATE (size, except$ptr);
```

Input Parameter

size	A DWORD that specifies the number of BYTES of memory being requested.

Output Parameters

seg$ptr	A POINTER that indicates the starting address of the acquired memory.
except$ptr	A POINTER to a word in which the system places the condition code.

Description

The DQ$MALLOCATE system call requests a specific amount of logically contiguous memory be added to the memory pool of the calling program. If the call is successful, the procedure returns a POINTER to the first byte of the acquired memory. If the call fails, the procedure returns a POINTER of undefined value and an exception code.

Multiple calls to DQ$MALLOCATE will result in multiple segments being allocated.

NOTE

DQ$MALLOCATE cannot be used in the PL/M SMALL model of compilation.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$MEM	0002H	Insufficient memory is available to fill the request.
E$SUPPORT	0023H	An unsupported operation was attempted.

In addition to the condition codes listed above, DQ$MALLOCATE can return the condition codes associated with the Nucleus system calls RQGETPOOL$ATTRIBUTES and RQ$CREATE$SEGMENT. See the *iRMX® II Nucleus System Calls Reference Manual or the iRMX® I Nucleus System Calls Reference Manual* for details.

DQ$MFREE returns memory allocated, by DQ$MALLOCATE, to the available memory pool.

```
CALL DQ$MFREE (seg$ptr, exception$ptr);
```

Input Parameters

seg$ptr	A POINTER to a block of memory that is to be returned to the available memory pool.

Output Parameters

exception$ptr	A POINTER to a WORD where the system places the condition code.

Description

The DQ$MFREE system call is used to return to available memory space a block of memory that was previously allocated using the DQ$MALLOCATE system call. Any memory freed by this call is no longer available to the calling program. Further attempts to use this area of memory may result in unexpected results since the memory referenced may have reallocated to another process.

In using the DQ$MFREE system call you must return an entire block of memory; it is not possible to return a portion of the memory allocated by a previous call to DQ$MALLOCATE.

Condition Codes

E$OK	0000H	No exceptional conditions.

In addition to the condition code listed above, DQ$MFREE can return the condition codes associated with the Nucleus system call RQ$DELETE$SEGMENT. See the *iRMX® II Nucleus System Calls Reference Manual or the iRMX® I Nucleus System Calls Reference Manual* for details.

The DQ$OPEN system call opens a file for I/O operations, specifies how the file will be accessed, and specifies the number of buffers needed to support the I/O operations.

```
CALL DQ$OPEN (connection$t, mode, num$buf, except$ptr);
```

Input Parameters

connection$t — A TOKEN for the file connection to be opened.

mode — A BYTE specifying how the connection will be used to access the file. This value is encoded as follows:

Value	Meaning
1	Read only
2	Write only
3	Update (both reading and writing)
4	Reserved
5-7	Available for Xenix systems; ignored by iRMX systems
8-255	Reserved

num$buf — A BYTE containing the number of buffers needed for this connection. Specifying a value larger than 0 implicitly requests that "double buffering" (that is, read-ahead and/or write-behind) is to be performed automatically. Specifying a value greater than 2, results in an E$SUPPORT error.

Output Parameter

except$ptr — A POINTER to a WORD where the system places the condition code.

Description

This system call prepares a connection for use with DQ$READ, DQ$WRITE, DQ$SEEK, and DQ$TRUNCATE commands. Your program can have up to six connections open simultaneously.

The DQ$OPEN system call does the following:

- Creates the requested buffers.
- Sets the connection's file pointer to zero. This is a place marker that tells where in the file the next I/O operation is to begin.
- Starts reading ahead if num$buf is greater than zero and the access parameter is "Read only" or "Update."

Selecting Access Rights

The system does not allow reading using a connection open for writing only nor writing using a connection open for reading only. If you are not certain how the connection will be used, specify updating. However, if the specified connection does not support the specified type of access, an exception code is returned.

Selecting the Number of Buffers

The process of deciding how many buffers to request is based on three considerations--compatibility, memory, and performance.

COMPATIBILITY. If you expect to run your UDI program on other systems, which support the UDI, you should request no more than two buffers.

MEMORY. The amount of memory used for buffers is directly proportional to the number of buffers. You can save memory by using fewer buffers.

PERFORMANCE. The performance consideration is more complex. Up to a certain point, the more buffers you allocate, the faster your program can run. The actual break-even point, where more buffers don't improve performance, depends on many variables. Often, the only way to determine the break-even point is to experiment. However, the following statements are true of every system:

- To overlap I/O with computation, you must request at least two buffers.
- If performance is not at all important but memory is, request no buffers.

Requesting zero buffers means that no buffering is to occur. That is, each DQ$READ or DQ$WRITE is followed immediately by the physical I/O operation necessary to perform the requested reading or writing. Interactive programs should open :CI: and :CO: with a request for no buffers.

If your program normally calls DQ$SEEK before calling DQ$READ or DQ$WRITE, it should request one buffer.

Your program can use the DQ$RESERVE$IO$MEMORY call to reserve memory that the UDI can use for its internal data structures when the program calls DQ$ATTACH and for buffers when the program calls DQ$OPEN. The advantage of reserving memory is that the memory is guaranteed to be available when needed. If memory is not reserved, a call to DQ$OPEN might not be successful because of a memory shortage. See the description of DQ$RESERVE$IO$MEMORY later in this chapter for more information about reserving memory.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$SUPPORT	0023H	At least one of the following is true: • The mode parameter is 4 or 8-255. • The num$buffs parameter is greater than two.
E$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.

In addition to the condition codes listed above, DQ$OPEN can return the condition codes associated with the Extended I/O system call RQSOPEN. See the *iRMX® II Extended I/O System Calls Reference Manual* for details.

In systems using overlays, the root module calls DQ$OVERLAY to load an overlay module.

```
CALL DQ$OVERLAY (name$ptr, except$ptr);
```

Input Parameter

name$ptr — A POINTER to a STRING containing the name of an overlay module. The name must be in uppercase.

Output Parameter

except$ptr — A POINTER to a WORD where the system places the condition code.

Description

A root module, in an overlay system, calls DQ$OVERLAY each time it wants to load an overlay module.

If your assembly language or PL/M-286 program uses the DQ$OVERLAY procedure, you should ensure that you bind the UDI library to your program correctly. The *80286 Utilities* manual describes the OVL286 utility in detail. The following steps describe the process for loading iRMX programs in overlay form.

1. Use BND286 to create linkable overlay files from compiled modules belonging to each overlay.
2. Use BND286 to create a nonpacked STL module from the linkable overlay files. The BND286 NOPACK control must be used.
3. Write an overlay definition file to describe the structure of the overlays in the program.
4. Use OVL286 to create an overlaid executable file from the linkable overlay files, the loadable module, and the overlay definition file.

To maintain portability to other operating systems that support the UDI, you should call no more than one level of overlay invoked only from the root of the application.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$SUPPORT	0023H	An supported operation was attempted.

In addition to the condition code listed above, DQ$OVER$LAY can return the condition codes associated with the Extended I/O system call RQSOVERLAY. See the *iRMX® II Extended I/O System Calls Reference Manual* for details.

The DQ$READ system call copies bytes from a file into a buffer.

```
bytes$read = DQ$READ (connection$t, buff$ptr, count, except$ptr);
```

Input Parameters

connection$t	A TOKEN for the connection to the file. This connection must be open for reading or for both reading and writing, and the file pointer of the connection must point to the first byte to be read.
buff$ptr	A POINTER to the buffer that is to receive the data from the file.
count	A WORD containing the requested number of bytes to be read from the file.

Output Parameters

bytes$read	A WORD containing the number of bytes actually read. This number is always equal to or less than count.
except$ptr	A POINTER to a WORD where the system places the condition code.

Description

This system call reads a collection of contiguous bytes from the file associated with the connection. The bytes are placed into the buffer specified in the call. If bytes$read is less than count and the exception code returned from the DQ$READ system call is E$OK, an end of file was encountered. If you type an interrupt or a terminate character from the console, for example a CONTROL-C, while the operating system performs a read operation, an E$OK exception code is returned and bytes$read is set to zero.

The Buffer

The buff$ptr parameter tells the operating system where to place the bytes when they are read. Your program must provide this buffer. DQ$READ copies as many bytes as it is instructed to copy (unless it encounters the end of the file). If the buffer is not long enough, copying continues beyond the end of the buffer.

Number of Bytes Read

The number of bytes that your program requests is the maximum number of bytes that DQ$READ copies into the buffer. However, there are circumstances under which the system reads fewer bytes.

- If the DQ$READ detects an end of file before reading the number of bytes requested, it returns only the bytes preceding the end of file. In this case, the bytes$read parameter is less than the count parameter, yet no exceptional condition is indicated.
- If an exceptional condition occurs during the reading operation, information in the buffer and the value of the bytes$read parameter are meaningless and should be ignored.
- If a CONTROL-C (interrupt or terminate) character is typed at the console (see description).

Connection Requirements

The connection must be open for reading or updating. If it is not, DQ$READ returns an exceptional condition.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$SUPPORT	0023H	An unsupported operation was attempted.

In addition to the condition codes listed above, DQ$READ can return the condition codes associated with the Extended I/O system call RQSREAD$MOVE (except E$FLUSHING). See the *iRMX® II Extended I/O System Call Reference Manual* for details.

The DQ$RENAME system call changes the pathname of a file.

```
CALL DQ$RENAME (path$ptr, new$path$ptr, except$ptr);
```

Input Parameters

path$ptr	A POINTER to a STRING that specifies the pathname of the file to be renamed.
new$path$ptr	A POINTER to a STRING that specifies the new pathname for the file. This path must not refer to an existing file.

Output Parameter

except$ptr	A POINTER to a WORD where the system places the condition code.

Description

This system call allows your programs to change the pathname of a data or a directory file. Be aware that when you rename a directory, you are changing the pathnames of all files contained in the directory. When you rename a file to which a connection exists--this is permitted--the connection to the renamed file remains established.

A file's pathname may be changed in any way, provided the file or directory remains on the same volume. Successfully renaming a file without appropriate access permission depends on the operating system.

If your operating system does not allow renaming a file to another volume or storage device, an E$SUPPORT exception is returned.

Condition Codes

E$OK	000H	No exceptional conditions.
E$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.

In addition to these condition codes, DQ$RENAME can return the condition codes associated with the Extended I/O System call RQSRENAME$FILE. See the *iRMX® II Extended I/O System Calls Reference Manual* for details.

DQ$RESERVE$IO$MEMORY

The DQ$RESERVE$IO$MEMORY system call lets your program reserve enough memory to ensure that it can open and attach the files it will be using.

```
CALL DQ$RESERVE$IO$MEMORY (number$files, number$buffers, except$ptr);
```

Input Parameters

number$files — A WORD whose value indicates the maximum number of files the program will have attached simultaneously. This value must not be greater than 12. Moreover, no more than 6 of these files may be open simultaneously.

number$buffers — A WORD whose value indicates the total number of buffers (up to a maximum of 12) that will be needed at one time. For example, if your program will have two files open at the same time, and each of them has two buffers (specified when they are opened), number$files should be two and number$buffers four.

If you specify a value for number$files or number$buffers that exceeds the limits explained above, an E$SUPPORT exception will be returned. If you specify a zero for both number$files and number$buffers, the memory reserved earlier will be returned to the memory pool.

Output Parameter

except$ptr — A POINTER to a WORD where the system places the condition code.

Description

DQ$RESERVE$IO$MEMORY sets aside memory on behalf of the calling program, guaranteeing that it will be available when needed later for attaching and opening files. This memory is used for internal UDI data structures when the program requests file connections via DQ$ATTACH and for buffers when the program opens file connections via DQ$OPEN. Memory reserved in this way is not eligible to be allocated by DQ$ALLOCATE or DQ$MALLOCATE. Your program should call DQ$RESERVE$IO$MEMORY before making any calls to DQ$ALLOCATE or DQ$MALLOCATE.

For an application to be portable across all operating systems that support UDI, it should not allow I/O without first explicitly reserving the memory by calling DQ$RESERVE$IO$MEMORY. In the call to DQ$RESERVE$IO$MEMORY, you may specify as many as 12 files (that can be attached using the reserved memory) and as many as 12 buffers (that can be requested when opening files).

NOTE

If a program calls DQ$RESERVE$IO$MEMORY after making one or more calls to DQ$ATTACH or DQ$OPEN, the memory used by those calls is immediately applied against the file and buffer counts specified in the DQ$RESERVE$IO$MEMORY call, possibly exhausting the memory supply being requested.

If your program calls DQ$RESERVE$IO$MEMORY more than once in a program, it simply increases or decreases the amount of memory reserved, unless your requests total more than 12 files or 12 buffers. If the requests exceed the maximum number of files or buffers, the maximum is reserved and no error is returned.

Restriction

This system call is effective only if your program uses exclusively UDI system calls to communicate with the iRMX Operating System.

Portability across operating systems that support the UDI cannot be guaranteed if your application requires more than 12 files attached simultaneously or a group of simultaneously open files whose total number of buffers exceeds 12.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$MEM	0002H	Insufficient memory remains to complete the call.
E$SUPPORT	0023H	At least one of the following is true: • The value specified for number$files is greater than 12. • The value specified for number$buffers is greater than 12.

DQ$SEEK moves the file pointer associated with the specified connection.

```
CALL DQ$SEEK (connection$t, mode, offset, except$ptr)
```

Input Parameters

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

mode — A BYTE indicating the type of file pointer movement being requested, as follows:

Mode	Meaning
1	Move the pointer backward by the specified move count. If the move count is large enough to position the pointer past the beginning of the file, the pointer is set to the first byte of the file (position zero).
2	Set the pointer to the position specified by the move count. Position zero is the first position in the file. Moving the pointer beyond the end of the file is permitted.
3	Move the file pointer forward by the specified move count. Moving the pointer beyond the end of the file is permitted.
4	First move the pointer to the end of the file and then move it backward by the specified move count. If the specified move count would position the pointer beyond the front of the file, the pointer is set to the first byte in the file (position zero).

offset — A DWORD specifying either how far, in bytes, the file pointer is to be moved, or the exact position in the file to which the pointer is to be moved.

Output Parameter

except$ptr — A POINTER to a WORD where the system places the condition code.

Description

When performing non-sequential I/O, your programs can use this system call to position the file pointer before using the DQ$READ, DQ$TRUNCATE, or DQ$WRITE system calls. The location of the file pointer specifies where in the file a DQ$READ, DQ$WRITE, or DQ$TRUNCATE operation is to begin. If your program is performing sequential I/O on a file, it need not use this system call.

You can position the file pointer beyond the end of a file. If your program does this and then invokes the DQ$READ system call, DQ$READ behaves as though the read operation began at the end of file. If your program calls DQ$WRITE when the file pointer is beyond the end of the file, the file is extended and the data is written as requested. A subsequent DQ$READ returns an end of file condition. Attempting a seek past the end of a file without performing an explicit DQ$WRITE call and subsequently expecting the file to be lengthened, will produce indeterminate results.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$PARAM	0023H	The mode parameter was set to 0 or 5-255.

In addition to the condition code listed above, DQ$SEEK can return the condition codes associated with the Extended I/O system call RQSSEEK. See the *iRMX® Extended I/O System Calls Reference Manual* for details.

DQ$SPECIAL

DQ$SPECIAL sets options or specifies actions to be performed in the program execution environment.

```
CALL DQ$SPECIAL (mode, parameter$ptr, except$ptr);
```

Input Parameters

mode — A BYTE used to specify the options to be set or the actions to be performed. Values and meanings of mode are

Value	Meaning
1	Transparent
2	Line editing (default value)
3	Polling
4-5	Reserved
6	Baud rate

Each of these modes is explained in the Description section.

parameter$ptr — A POINTER. See complete explanation in the Description section.

Output Parameter

except$ptr — A POINTER to a WORD where the system places the condition code.

Description

This system call changes the mode in which your program receives input from a console input device. When your system starts to run, the mode is line editing (mode 2). By using DQ$SPECIAL, you can change to either of the other two modes, or back to line editing.

The meanings of the mode parameter values are as follows:

Value	Meaning
1	Transparent. Interactive programs must often obtain characters from the console exactly as they are typed. DQ$READ returns control to the calling program when the number of characters entered equals the number of characters specified in the read request.
2	Line Editing. This option enables you to correct typing errors with special keys before the application program receives the characters typed. Characters used for editing are operating-system-dependent. The RETURN character is always converted to CARRIAGE-RETURN-LINE-FEED (CRLF).
3	Polling. This option is nearly the same as Transparent (1) mode, except that in Polling mode DQ$READ returns control to your program immediately after it is called, regardless of whether any characters have been 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$READ call. Characters typed between successive calls to read the terminal are held in the "type-ahead" buffer. where parameter$ptr — A POINTER to a TOKEN for a connection to the :CI: file previously established by DQ$ATTACH.
4-5	Reserved, E$SUPPORT will be returned.
6	Baud Rate — Specifies baud rate selection for an asynchronous line. where parameter$ptr points to this structure:

```
DECLARE LINE BASED parameter$ptr STRUCTURE (
                          conn            TOKEN,
                          in$baud$rate    BYTE,
                          out$baud$rate   BYTE);
```

where

LINE.Conn	is a connection previously established by a DQ$ATTACH call.
LINE.in$baud$rate	specifies the desired input baud rate.
LINE.out$baud$rate	specifies the desired output baud rate.

These values specify baud rate:

Byte Value	Baud Rate
0	Unspecified
1	300
2	600
3	1200
4	2400
5	4800
6	9600
7	19200
8-255	Reserved

Condition Codes

E$OK	000H	No exceptional conditions.
E$SUPPORT	0023H	The mode parameter represents an unsupported mode.

In addition to the condition codes listed above, DQ$SPECIAL can return the condition codes associated with the Extended I/O system call RQSSPECIAL. See the *iRMX® Extended I/O System Calls Reference Manual* for details.

DQ$SWITCH$BUFFER substitutes a new command line for the existing one.

```
char$offset = DQ$SWITCH$BUFFER (buff$ptr, except$ptr);
```

Input Parameter

buff$ptr — A POINTER to a buffer containing the "new" command line. That is, the one whose arguments are to be returned by subsequent calls to DQGETARGUMENT. The buffer must not exceed 32 K-bytes in length.

Output Parameters

char$offset — A WORD into which the UDI places a number. This number represents the number of bytes from the beginning of the "old" command line to the last character of the last argument so far processed by DQGETARGUMENT. In other words, the value in char$offset tells how many characters in the old command line have been processed by the time of this call.

except$ptr — A POINTER to a WORD where the system places the condition code.

Description

When your program is invoked from the console, the operating system places the invocation command into a buffer. Typically, your program will use DQGETARGUMENT to obtain the arguments in that command. If your program subsequently calls DQ$READ to obtain an additional command line from the console, it can call DQ$SWITCH$BUFFER to designate the buffer with the new command line as that from which arguments are to be obtained when DQGETARGUMENT is called.

You can use DQ$SWITCH$BUFFER any number of times to point to different strings in your program. However, you cannot use DQ$SWITCH$BUFFER to return to the command line that invoked the program, because only the operating system knows the location of that buffer. Therefore, you should use DQGETARGUMENT to obtain all arguments of the invocation command line before issuing the first call to DQ$SWITCH$BUFFER.

A second service of DQ$SWITCH$BUFFER is that it returns the location of the last byte of the last argument so far obtained from the old buffer by calls to DQGETARGUMENT. Therefore, in addition to using DQ$SWITCH$BUFFER to switch buffers, you can use it after one or more DQGETARGUMENT calls to determine where in the buffer the next argument starts. However, doing this "resets" the buffer, in the sense that the next call to DQGETARGUMENT would return the first argument in the buffer. To return to the desired point in the buffer, where you can continue to extract arguments, call DQ$SWITCH$BUFFER again, but when doing so, use the sum of the starting address of the buffer and the value returned by the previous call to DQ$SWITCH$BUFFER. The following is an example showing how to use the second service of DQ$SWITCH$BUFFER:

```
DECLARE
   E$OK   LITERALLY '0'
   E$FATAL$EXIT   LITERALLY '3'
   mybuffer$ptr         POINTER,
   buff$ptr             POINTER,
   arg$ptr              POINTER,
   buff                 STRUCTURE(
               offset     WORD,
               segment    WORD) AT (@buff$ptr),
   next$char            WORD,
   char$offset          WORD,
   condition$code       WORD,
   delimit$char         BYTE;
         •
         •
         •

/* initialize buff$ptr and next$char */

    buff$ptr = mybuff$ptr;
    next$char = 0;
         •
         •
         •

/* determine where in the buffer the next argument starts */

    char$offset = DQ$SWITCH$BUFFER( buff$ptr, @condition$code );
    IF condition$code <> E$OK THEN          /* do error processing  */
        CALL DQ$EXIT(E$FATAL$EXIT)
    next$char = char$offset + next$char;
```

(Example continued on next page)

```
/* return to desired point in buffer */

    buff.offset = buff.offset + char$offset;
    char$offset = DQ$SWITCH$BUFFER( buff$ptr, @condition$code );
        IF condition$code <> E$OK THEN          /* do error processing  */
            CALL DQ$EXIT(E$FATAL$EXIT)

/* get next argument */

    delimit$char = DQ$GET$ARGUMENT( arg$ptr, @condition$ptr );
        IF condition$code <> E$OK THEN          /* do error processing  */
            CALL DQ$EXIT(E$FATAL$EXIT)
        •
        •
        •
```

Condition Codes

E$OK	0000H	No exceptional conditions.

In addition to the condition code listed above, DQ$SWITCH$BUFFER can return the condition codes associated with the Human Interface system call RQCSET$PARSE$BUFFER. See the *iRMX® Human Interface System Calls Reference Manual* for details.

DQ$TRAP$CC lets you specify a procedure that gains control if an operator enters an interrupt character (such as CONTROL-C) at the console.

```
CALL DQ$TRAP$CC (cc$routine$ptr, except$ptr);
```

Input Parameter

cc$routine$ptr	A POINTER to the entry point of your interrupt procedure.

Output Parameter

except$ptr	A POINTER to a WORD where the system places the condition code.

Description

The action the default interrupt procedure takes depends on the operating system. Using the DQ$TRAP$CC system call lets you substitute an alternate interrupt procedure that will automatically receive control when you enter an interrupt character on the console. (See the *iRMX® Human Interface User's Guide* for more information.) The context of the program executing at the time you invoke DQ$TRAP$CC is saved by the operating system. Due to this context switch, the contents of the CPU registers at the time the interrupt procedure receives control may not be those associated with your program. The CPU registers may contain values for an internal task that was executing when the interrupt character was entered.

To ensure portability across other operating systems, a GOTO statement (PL/M, C, FORTRAN, etc.) must not branch outside the DQ$TRAP$CC procedure's routine.

Condition Codes

E$OK	0000H	No exceptional conditions.

DQ$TRAP$EXCEPTION substitutes an alternate exception handler for the default exception handler provided by the operating system.

```
CALL DQ$TRAP$EXCEPTION (handler$ptr, except$ptr);
```

Input Parameter

handler$ptr — A POINTER to a STRUCTURE containing a long pointer to the entry point of the alternate exception handler. The STRUCTURE has the form

```
DECLARE handler$ptr STRUCTURE (
                    offset  WORD,
                    base    TOKEN);
```

Output Parameter

except$ptr — A POINTER to a WORD where the system places the condition code.

Description

DQ$TRAP$EXCEPTION designates an alternate exception handler as the one to which control should pass when an exceptional condition occurs. The DQ$TRAP$EXCEPTION routine should restore the default exception handler before it terminates. Therefore, your program should call DQGETEXCEPTION$HANDLER before calling DQ$TRAP$EXCEPTION to get the default exception handler address.

See the section Condition Codes and Exception-Handling Calls at the beginning of this manual for an explanation of the conditions of the stack when your alternate exception handler receives control.

Condition Codes

E$OK	0000H	No exceptional conditions.

In addition to the condition code listed above, DQ$TRAP$EXCEPTION can return the condition codes associated with the Nucleus system call RQSETEXCEPTION$HANDLER. See the *iRMX® II Nucleus System Calls Reference Manual or the iRMX® I Nucleus System Calls Reference Manual* for details.

DQ$TRUNCATE moves the end-of-file to the current position of a named file connection's file pointer, thereby freeing the portion of the file lying beyond the file pointer.

```
CALL DQ$TRUNCATE (connection$t, except$ptr);
```

Input Parameter

connection$t — A TOKEN for an open connection to the named data file that is to be truncated. The file pointer of this connection marks the place where truncation is to occur. The byte indicated by the file pointer is the first byte to be dropped from the file.

Output Parameter

except$ptr — A POINTER to a WORD where the system places the condition code.

Description

This system call truncates a file at the current setting of the file pointer and releases all file space beyond the pointer for reallocation to other files. If the pointer is at or beyond the end of file, no truncation is performed. Unless the file pointer is already at the proper location, your program should use the DQ$SEEK system call to position the pointer before calling DQ$TRUNCATE.

The connection should have write, or read and write access rights, established when the connection was opened.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$SUPPORT	0023H	An unsupported operation was attempted.

In addition to the condition codes listed above, DQ$TRUNCATE can return the condition codes associated with the Extended I/O system call RQSTRUNCATE$FILE. See the *iRMX® Extended I/O System Calls Reference Manual* for details.

The DQ$WRITE system call copies a collection of bytes from a buffer into a file.

```
CALL DQ$WRITE (connection$t, buff$ptr, count, except$ptr);
```

Input Parameters

connection$t	A TOKEN containing the connection to the file into which the information is to be written.
buff$ptr	A POINTER to a buffer containing the data to be written to the specified file.
count	A WORD containing the number of bytes to be written from the buffer to the file.

Output Parameter

except$ptr	A POINTER to a WORD where the system places the condition code.

Description

This system call causes the operating system to write the specified number of bytes from the buffer to the file.

Number of Bytes Written

Occasionally, DQ$WRITE writes fewer bytes than requested by the calling program. This happens under the following two circumstances:

- When DQ$WRITE encounters an I/O error.
- When the volume to which your program is writing becomes full.

Where the Bytes Are Written

DQ$WRITE starts writing at the location specified by the connection's file pointer. After the writing operation is completed, the file pointer points to the byte immediately following the last byte written.

If your program must reposition the file pointer before writing, it can do so by using the DQ$SEEK system call.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$SUPPORT	0023H	An unsupported operation was attempted.
E$SPACE	0029H	Inadequate memory space remains to complete the write.

In addition to the condition code listed above, DQ$WRITE can return the condition codes associated with the Extended I/O system call RQSWRITE$MOVE. See the *iRMX® Extended I/O System Calls Reference Manual* for details.

INDEX

A

Access mask 11
Access rights 10
  from the ACCESS field of DQGETCONNECTION$STATUS 32
  needed to perform DQ$TRUNCATE 64
  OWNER$ACCESS field in DQ$FILE$INFO 25
  selecting 44

B

Baud rate
  how to set using DQ$SPECIAL 57
  value for mode parameter of DQ$SPECIAL 56
BND286, using to create overlay files 46
Buffer 29
  DQ$CLOSE 15
  for DQGETSYSTEM$ID 38
  for DQ$READ 29
  for the buff$ptr parameter of DQ$READ 48
  number required for DQ$OPEN 43
  the buff$ptr parameter of DQ$SWITCH$BUFFER 59
  the buff$ptr parameter of DQ$WRITE 65
  the number$buffers parameter of DQ$RESERVEIOMEMORY 52

C

CI (console input) 44
CO (console output) 44
Command line 30
  parsing with DQGETARGUMENT 29
Compatibility
  DQGETTIME system call 39
  number of buffers permitted in the DQ$OPEN system call 44
  setting the ACCESS bit of DQ$CHANGE$ACCESS for 10
  setting the ACCESS field of the DQGETCONNECTION$STATUS system call 32
  setting the WORLD$ACCESS field of DQ$FILE$INFO system call 25
Condition codes 3
Condition codes, table of 1, 2
Connection
  Boolean test for state 32
  creating using DQ$CREATE 16
  default access rights 11
  deleting using DQ$DETACH 21

freeing buffers associated with a connection 15
getting information using DQGETCONNECTION$STATUS 32
moving the file pointer 54
requirements for DQ$READ 49
truncating the associated file 64
Connection, specifying the number of buffers required for 43
CONTROL-C 4, 48, 49, 57, 62

D

Data structure
for DQ$DECODE$TIME 18
for DQ$FILE$INFO 24
for DQGETCONNECTION$STATUS 32
for DQ$SPECIAL 57
for DQ$TRAP$EXCEPTION 63
DATE 18, 19, 39
Default user 11
Delimiter 29, 30
example of delimiters returned from DQGETARGUMENT 30
DQ$ALLOCATE 7
DQ$ATTACH 8
DQ$CHANGE$ACCESS 10
DQ$CHANGE$EXTENSION 13
DQ$CLOSE 15
DQ$CREATE 16
DQ$DECODE$EXCEPTION 17
DQ$DECODE$TIME 18
DQ$DELETE 20
DQ$DETACH 21
DQ$EXIT 22
DQ$FILE$INFO 24
DQ$FREE 28
DQGETARGUMENT 29
DQGETCONNECTION$STATUS 32
DQGETEXCEPTION$HANDLER 34
DQGETMSIZE 36
DQGETSIZE 37
DQGETSYSTEM$ID 38
DQGETTIME 39
DQ$MALLOCATE 40
DQ$MFREE 42
DQ$OPEN 43
DQ$OVERLAY 46
DQ$READ 48
DQ$RENAME 50

DQ$RESERVE$IO$MEMORY 52
DQ$SEEK 54
DQ$SPECIAL 56
  baud rate 57
  line editing 57
  polling 57
DQ$SWITCH$BUFFER 59
DQ$TRAP$CC 62
DQ$TRAP$EXCEPTION 63
DQ$TRUNCATE 64
DQ$WRITE 65

E

End of file 48, 49, 55, 64
Examples
  delimiters returned by DQGETARGUMENT 30
  DQ$SWITCH$BUFFER 60
Exception handling
  getting the address of the current exception handler 34
  using your own exception handler 63

F

File
  changing the pathname 50
  creation 16
  deletion 20
  extension 13
  information 24, 32
  operations 43, 48, 52, 64, 65
  pointer 54, 64
  size 25
Free space pool, requesting additional memory from 7

I

Interactive programs
  getting characters from the console 57
  opening CI and CO for interactive programs 44
Interrupt procedure 62

L

Line editing mode 57

M

Memory
  block 36, 42
  pool 7, 28, 40, 42, 52

reservation 45, 52
Mode
file pointer seeks 54
parameter of DQ$FILE$INFO 24
parameter of DQ$OPEN 43
terminal 56
Model of segmentation 34, 40

O

Object
file 13
user 11
Object file 13
Operating system identification 38
OSC sequences 57
OVL286, using to create programs that use overlays 46
Owner ID 11
Owner of a file 10

P

Performance 44
PL/M 3, 40
PL/M-286 46
Polling 56
Portability 46, 53, 62
Program control
DQ$EXIT 22
DQ$OVERLAY 46
DQ$TRAP$CC 62
system calls 4

R

Reserving memory 45, 52
Root module 46

S

Segment 7, 28, 36, 37
System calls
descriptions 3
dictionary 4
exception-handling 6
file-handling 4
memory management 5
program control 4
utility and command parsing 6

T

Task 7, 22, 62
Terminal modes
  polling 56
Terminating programs 22
TIME 18, 39
Transparent mode 57

U

UDI library 46
User
  default 11
  ID 11, 24
  object 11
  WORLD 10
User object 11

W

WORLD 10, 11, 25
WORLD user 10

REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative.

1. Please describe any errors you found in this publication (include page number).

2. Does this publication cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of publication for your needs? Is it at the right level? What other types of publications are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating). _____

NAME _____ DATE _____
TITLE _____
COMPANY NAME/DEPARTMENT _____
ADDRESS _____ PHONE ( ) _____
CITY _____ STATE _____ ZIP CODE _____
(COUNTRY)

Please check here if you require a written reply. ☐

'VE'D LIKE YOUR COMMENTS . . .

'his document is one of a series describing Intel products. Your comments on the back of this form will ıelp us produce better manuals. Each reply will be carefully reviewed by the responsible person. All omments and suggestions become the property of Intel Corporation.

' you are in the United States, use the preprinted address provided on this form to return your omments. No postage is required. If you are not in the United States, return your comments to the Intel ales office in your country. For your convenience, international sales office addresses are printed on ıe last page of this document.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 79 HILLSBORO, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
OMSO Technical Publications, MS: HF3-72
5200 N.E. Elam Young Parkway
Hillsboro, OR 97124-9978

INTERNATIONAL SALES OFFICES

INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, California 95051

BELGIUM
Intel Corporation SA
Rue des Cottages 65
B-1180 Brussels

DENMARK
Intel Denmark A/S
Glentevej 61-3rd Floor
dk-2400 Copenhagen

ENGLAND
Intel Corporation (U.K.) LTD.
Piper's Way
Swindon, Wiltshire SN3 1RJ

FINLAND
Intel Finland OY
Ruosilante 2
00390 Helsinki

FRANCE
Intel Paris
1 Rue Edison-BP 303
78054 St.-Quentin-en-Yvelines Cedex

ISRAEL
Intel Semiconductors LTD.
Atidim Industrial Park
Neve Sharet
P.O. Box 43202
Tel-Aviv 61430

ITALY
Intel Corporation S.P.A.
Milandfiori, Palazzo E/4
20090 Assago (Milano)

JAPAN
Intel Japan K.K.
Flower-Hill Shin-machi
1-23-9, Shinmachi
Setagaya-ku, Tokyo 15

NETHERLANDS
Intel Semiconductor (Netherland B.V.)
Alexanderpoort Building
Marten Meesweg 93
3068 Rotterdam

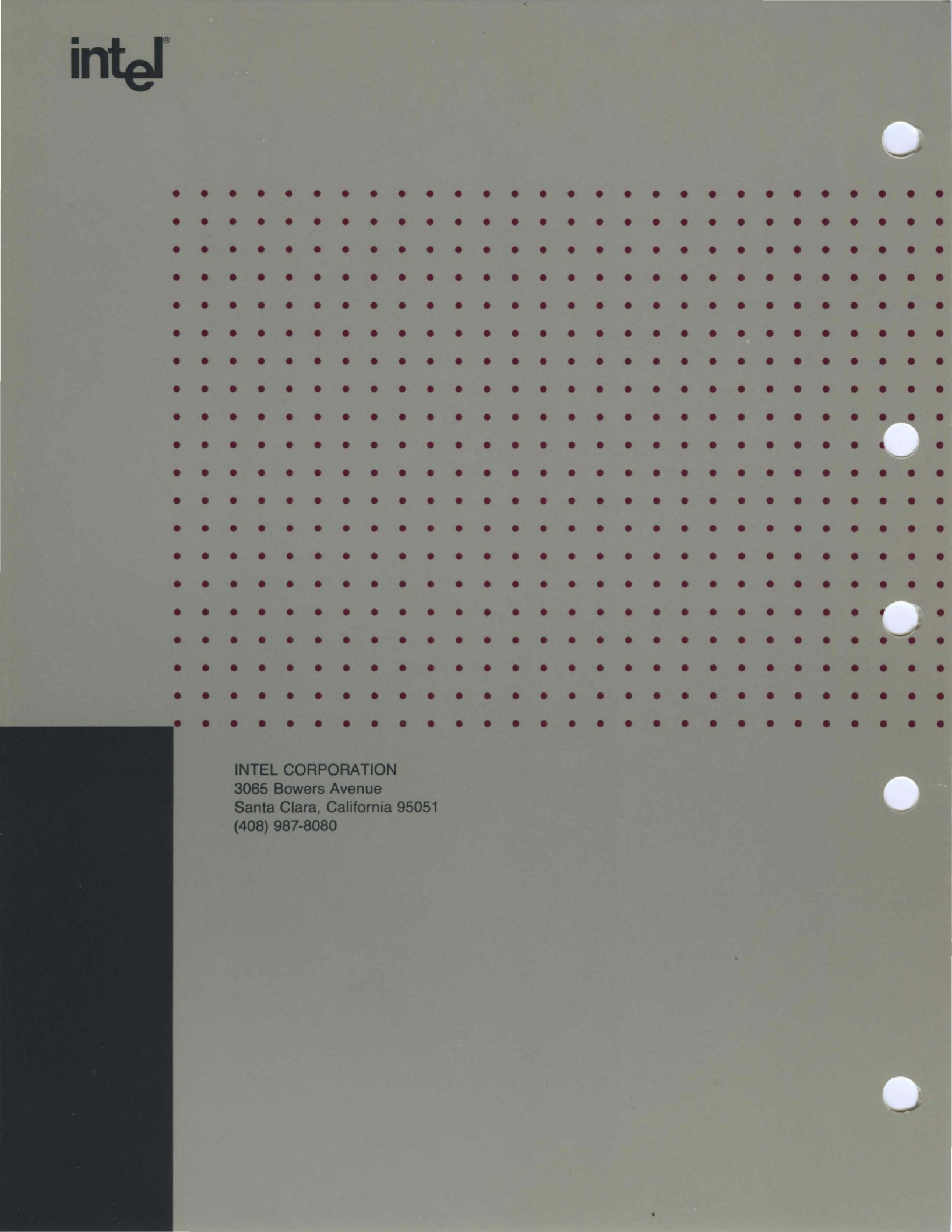
NORWAY
Intel Norway A/S
P.O. Box 92
Hvamveien 4
N-2013, Skjetten

SPAIN
Intel Iberia
Calle Zurbaran 28-IZQDA
28010 Madrid

SWEDEN
Intel Sweden A.B.
Dalvaegen 24
S-171 36 Solna

SWITZERLAND
Intel Semiconductor A.G.
Talackerstrasse 17
8125 Glattbrugg
CH-8065 Zurich

WEST GERMANY
Intel Semiconductor G.N.B.H.
Seidlestrasse 27
D-8000 Munchen



INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, California 95051
(408) 987-8080