intel

iRMX® I Application Loader System Calls Reference Manual

Order Number: 462917-001

intel®

iRMX® I Application Loader System Calls Reference Manual

Order Number: 462917-001

Intel Corporation
3065 Bowers Avenue
Santa Clara, California 95051

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

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

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

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

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^2	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.

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

PREFACE

This manual documents the system calls of the Application Loader, a subsystem of the iRMX® I Operating System. The information provided in this manual is intended to serve as a reference to the system calls and provides detailed descriptions of each call.

READER LEVEL

This manual is intended for programmers who are familiar with the PL/M-86 programming language and with the concepts and terminology introduced in the *iRMX® I Nucleus User's Guide* and the *iRMX® I Application Loader User's Guide*

CONVENTIONS

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

Throughout this manual, system calls are shown using a generic shorthand (such as A$LOAD instead of RQ$A$LOAD). This convention is used to allow easier alphabetic arrangement of the calls. The actual PL/M-86 external procedure names must be used in all calling sequences.

You can also invoke the system calls from assembly language programs, but you must adhere to the PL/M-86 calling sequences when doing so. For more information on these calling sequences refer to the *iRMX® I Programming Techniques Reference Manual.*

CONTENTS

Chapter 1. iRMX® I Application Loader System Calls

Index

iRMX® I APPLICATION LOADER SYSTEM CALLS 1

1.1 INTRODUCTION

This manual describes the PL/M-86 calling sequences for the system calls of the Application Loader.

Throughout this manual, PL/M-86 data types, such as BYTE, WORD, and SELECTOR are used. In addition, the iRMX® I data type TOKEN is used. Data types always appear in capital letters. If your compiler supports the SELECTOR data type, a TOKEN can be declared literally as SELECTOR. Because TOKEN is not a PL/M-86 data type, you must declare it to be literally a SELECTOR every place you use it. Definitions of both PL/M-86 and iRMX I data types are given in the *iRMX® I Nucleus User's Guide*, Appendix A. The word "token" in lowercase refers to a value that the iRMX I Operating System returns to a TOKEN (the data type) when it creates an object.

NOTE

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

1.2 RESPONSE MAILBOX PARAMETER

Two system calls described in this manual are asynchronous. These are the A$LOAD and A$LOADIOJOB system calls. Your task must specify a mailbox whenever it invokes an asynchronous system call. The purpose of this mailbox is to receive a Loader Result Segment.

In general, the Loader Result Segment indicates the result of the loading operation. The format of a Loader Result Segment depends on which system call was invoked, so details about Loader Result Segments are included in descriptions of the A$LOAD and A$LOADIOJOB system calls.

Avoid using the same response mailbox for more than one concurrent invocation of asynchronous system calls. This is necessary because it is possible for the Application Loader to return Loader Result Segments in an order different than the order of invocation. On the other hand, it is safe to use the same mailbox for multiple invocations of asynchronous system calls if only one task invokes the calls and the task always obtains the result of one call via RQ$RECEIVE$MESSAGE before making the next call.

1.3 CONDITION CODES

The Application Loader returns a condition code whenever a system call is invoked. If the call executes without error, the Application Loader returns the code E$OK. If an error occurs, the Application Loader returns a condition code.

This manual includes, for each of the Application Loader's system calls, descriptions of the condition codes that the system call can return. The system call manuals for the other layers of the iRMX I Operating System do the same thing for those layers. You can use the condition code information to write code to handle exceptional conditions that arise when system calls fail to perform as expected. See the *iRMX® I Nucleus User's Guide* for a discussion of condition codes and how to write code to handle them.

1.3.1 Condition Codes For Synchronous System Calls

For system calls that are synchronous (S$LOAD$IO$JOB and S$OVERLAY), the Application Loader returns a single condition code each time the call is invoked. If your system has an exception handler, it will receive this code when an exceptional condition occurs, depending on how the exception$mode parameter is set. For more information see the *iRMX® I Nucleus User's Guide* and the *iRMX® I Interactive Configuration Utility Reference Manual.*

1.3.2 Condition Codes For Asynchronous System Calls

For system calls that are asynchronous (A$LOAD and A$LOADIOJOB), the Application Loader returns two condition codes each time the call is invoked. One code is returned after the sequential part of the system call is executed, and the other is returned after the concurrent part of the call is executed. Your task must process these two condition codes separately.

The *iRMX® I Application Loader User's Guide* describes the sequential and concurrent portions of asynchronous system calls.

1.3.2.1 Sequential Condition Codes

The Application Loader returns the sequential condition code in the word pointed to by the except$ptr parameter. If your system has an exception handler, it will receive this code when an exceptional condition occurs, depending upon how the exception$mode parameter is set.

1.3.2.2 Concurrent Condition Codes

The Application Loader returns the concurrent condition code in the Loader Result Segment it sends to the response mailbox. If the code is E$OK, the asynchronous loading operation ran successfully. If the code is other than E$OK, a problem occurred during the asynchronous loading operation, and your task must decide what to do about the problem. Regardless of the exception mode setting for the application, the exception handler is not invoked by concurrent condition codes, so your program must handle it.

1.4 SYSTEM CALL DICTIONARY

The following list is a summary of the iRMX I Application Loader system calls, together with a brief description of each call and the page where the description of the call begins.

Call	Description	Page
	ASYNCHRONOUS CALLS	
A$LOAD	Loads object code or data into memory	4
A$LOAD$IO$JOB	Creates an I/O job, loads the job's code, and causes the job's task to run.	15
	SYNCHRONOUS CALLS	
S$LOAD$IO$JOB	Creates an I/O job, loads the job's code, and causes the job's task to run.	26
S$OVERLAY	Loads an overlay into memory.	34

A$LOAD

The A$LOAD system call loads an object file from secondary storage into memory.

```
CALL RQ$A$LOAD(connection, response$mbox, except$ptr);
```

Input Parameters

connection — A TOKEN for a connection to the file that is to be loaded. The connection must satisfy all of the following requirements:

- It must have been created in the calling task's job.
- It must be a connection to a named file.
- The calling user must have had READ access to the file.
- It must be closed.

If all of these connection requirements are not met, the Application Loader returns an exception code.

Output Parameter

response$mbox — A TOKEN for the mailbox to which the Application Loader sends the Loader Result Segment after the concurrent part of the system call finishes running. The format of the Loader Result Segment is given in the following DESCRIPTION section.

except$ptr — A POINTER to a WORD where the Application Loader will place the condition code generated by the sequential part of the system call.

Description

A$LOAD allows your task to load object code files from secondary storage into memory. The object code to be loaded must be of the Single Task Loadable (STL) type with LODFIX records.

Unlike the A$LOAD$IO$JOB and S$LOADIOJOB system calls, A$LOAD cannot automatically cause the code to be executed as a task. The caller must explicitly cause the code to be executed.

Using A$LOAD to Load a Main Module

If you use A$LOAD to load a main module that will run as a task, there are two cases to consider.

1. The usual case is when you are loading PIC or LTL code, or you are loading absolute code generated with the NOINITCODE control of the LOC86 command. In this case, the Application Loader returns, in the Loader Result Segment, parameters defining the entry point and stack requirements for the loaded code. Your application needs these parameters when invoking the CREATE$TASK, CREATE$JOB, or CREATEIOJOB system call.

   If the Application Loader has been configured to load only absolute code, it will not load main modules generated with the NOINITCODE control. In this event, the Application Loader returns the E$LOADER$SUPPORT condition code. (See the *iRMX® I Interactive Configuration Utility Reference Manual* for information about configuring the Application Loader.)

2. If your object code is absolute code generated without the NOINITCODE control of the LOC86 command, you must allow the iRMX I Nucleus to create a stack for you. To do this, specify NIL for the stack pointer parameter of the CREATE$TASK or the CREATE$JOB system call.

   This action causes the Nucleus to create a stack for the loaded code. However, because the loaded code contains a main module, it also contains code that switches the stack register values so the Nucleus-created stack is ignored. This stack switching allows the loaded code to use the stack allocated by the SEGSIZE control.

   To minimize the amount of memory wasted by stack switching, specify a small stack size (128 decimal bytes) in the CREATE$TASK, CREATE$JOB, or CREATEIOJOB system calls. This stack need not be large because it is used only if the task is interrupted and stack switching occurs.

   Stack switching has an undesirable but avoidable side effect. If you use the iRMX I Dynamic Debugger, it will always indicate that the stack for the loaded code has overflowed. The overflow indication is caused by the main module switching stacks, rather than by an actual overflow. This means that you cannot tell whether overflow actually has occurred. To avoid this side effect, write your source code as a procedure or use the LOC86 NOINITCODE control.

Using A$LOAD To Load A Procedure

If you write code as a procedure that you intend to load and run, it can be loaded only by A$LOAD. Although the process of loading a procedure is more restrictive than that of loading a main module, you can avoid the stack-switching side effects described in the previous section.

To successfully load code that is written as a procedure, adhere to the following rules:

- Generate the procedure as absolute code and do not use the NOINITCODE control of the LOC86 command.
- Adhere to the PL/M-86 LARGE model of segmentation. This means that you must either compile the procedure using the LARGE size control, or you must follow the calling conventions of the LARGE model. For information about the PL/M-86 LARGE model of segmentation, refer to the PL/M-86 user's guide.
- When invoking the LOC86 command to assign absolute addresses to your object code, use the START control to select one of the PUBLIC symbols in your procedure as an entry point. Also specify SEGSIZE(STACK(0)) to set the stack to zero length. For more information about the START and SEGSIZE controls, refer to the *8086 Family Utilities User's Guide.*
- When you invoke the CREATE$TASK, CREATE$JOB, or CREATEIOJOB system call, allow the operating system to allocate a stack for the new task. Do this by setting the stack pointer parameter to NIL. Be certain that you specify a stack size parameter that is large enough for the task. For guidelines to determining stack sizes, refer to the *iRMX® I Programming Techniques Reference Manual.*
- When you invoke the CREATE$TASK, CREATE$JOB, or CREATEIOJOB system call, set the data segment base parameter to SELECTOR$OF(NIL). The reason for this is that a procedure adhering to the LARGE model of segmentation always initializes its own data segment.

For information about the CREATE$TASK or the CREATE$JOB system calls, refer to *iRMX® I Nucleus System Calls Reference Manual.*

For information about the CREATEIOJOB system call, refer to the *iRMX® Extended I/O System Calls Manual.* For information about the iRMX I Dynamic Debugger, refer to the *iRMX® I Dynamic Debugger Reference Manual.*

Asynchronous Behavior

The A$LOAD system call is asynchronous. It allows the calling task to continue running while the loading operation is in progress. When the loading operation is finished, the Application Loader sends a Loader Result Segment to the mailbox designated by the response$mbox parameter. Refer to the *iRMX® I Application Loader User's Guide* for an explanation of how asynchronous system calls work.

File Sharing

The Application Loader does not expect exclusive access to the file. However, other tasks sharing the file are affected by the following:

- The other tasks should not attempt to share the connection passed to the Application Loader, but instead should obtain their own connections to the file.
- The Application Loader specifies "share with readers only" when opening the connection, so, during the loading operation, other tasks can access the file only for reading.

Considerations Relating To Code Type

If the file being loaded contains absolute code, the Application Loader will not create iRMX I segments for the code. Rather, it will simply load the program into the memory locations specified for the target file. It is the user's responsibility to prevent code from loading over existing information, including the operating system code. Refer to the *iRMX® I Interactive Configuration Utility Reference Manual* to see how to do this by reserving areas of memory.

In contrast, if the file being loaded is position-independent code (PIC) or load-time locatable code (LTL), the Application Loader will create iRMX I segments for containing the loaded program. However, the Application Loader does not delete these segments; when your task no longer needs the loaded program, your task should delete the segments.

Effects Of Model Of Segmentation

The Application Loader will return (in the Loader Result Segment) a token for each of the code, data, and stack segments.

This is enough segment information for programs compiled as COMPACT, because only one segment of each type will be created. But if the program adheres to the LARGE or MEDIUM model of segmentation, more than one code segment and more than one data segment can be created, although only one token will be returned for each in the Loader Result Segment.

This means that if the code follows the LARGE or MEDIUM model, the calling task cannot know the location of all of the loaded program's code or data segments. Consequently, the calling task cannot delete all of the data or code segments after the program has executed. You can avoid this problem in either of two ways. Either be certain that the program being loaded adheres to the COMPACT model of segmentation, or use the A$LOAD$IO$JOB or S$LOADIOJOB system calls instead of the A$LOAD system call.

The A$LOAD Loader Result Segment

The Application Loader uses memory from the pool of the calling task's job to create the Loader Result Segment for this system call. The calling task should delete the segment after it is no longer needed. Creating multiple segments without deleting them can eventually result in an E$MEM exception code.

The Loader Result Segment has the following form:

```
STRUCTURE   (except$code              WORD,
             record$count             WORD,
             error$rec$type           BYTE,
             undefined$ref            WORD,
             init$ip                  WORD,
             code$seg$base            TOKEN,
             stack$offset             WORD,
             stack$seg$base           TOKEN,
             stack$size               WORD,
             data$seg$base            TOKEN);
```

where:

except$code — A WORD containing the condition code for the concurrent part of the system call. If the code is other than E$OK, some problem occurred during the loading operation.

record$count — A WORD containing the number of records read by the Application Loader on this invocation of A$LOAD. If the loading operation terminates prematurely, record$count contains the number of the last record read.

errorrectype — A BYTE identifying the type of record causing premature termination of the loading operation, except that a value of 0 means no record caused premature termination. Object record types are documented in the Intel publication *8086 Relocatable Object Module Formats*.

undefined$ref — A WORD specifying whether the Application Loader found undefined external references while loading the job. An undefined external reference usually results from a linking error. The Application Loader continues to run even if a target file contains undefined external references.

The value of undefined$ref depends upon your configuration of the Application Loader. (See the *iRMX® I Interactive Configuration Utility Reference Manual* for information about configuring the Application Loader.)

- If the Application Loader is configured to load LTL and overlay code, as well as PIC and absolute code, undefined$ref contains the number of undefined external references detected during the loading operation. (Note that undefined$ref equals the number of undefined external references even if the Application Loader is loading PIC or absolute code.)
- If the Application Loader is configured to load only absolute code or only PIC or absolute code, the Application Loader sets undefined$ref to 1 or to 0. It is 1 if the Application Loader finds undefined external references; otherwise, it is 0.

init$ip

A WORD containing the initial value for the loaded program's instruction pointer (IP register). The calling task can use this information in either of two ways:

- When invoking the CREATE$TASK, CREATE$JOB, or CREATEIOJOB system call.
- As the destination of a jump within the code segment of the loaded program.

Init$ip is 0 if the file does not specify an initial value for the instruction pointer, as can happen when the file contains no main module.

codesegbase

A TOKEN containing the base address for the code segment with the entry point. The value in codesegbase can be used with init$ip as a POINTER to the entry point of the loaded program. The Application Loader places 0 into this field if the loaded program does not contain a main module.

stack$offset

A WORD containing the offset of the bottom of the stack, relative to the beginning of the stack segment. The calling task can use the sum of this value and the stack$size to initialize the stack pointer (SP register).

The Application Loader sets stack$offset to zero under each of these circumstances:

- The stack actually starts at offset 0.
- There is no main module.
- The loaded code is a main module that dynamically initializes the SP and SS registers.

stacksegbase — A TOKEN containing the base of the stack segment for the loaded program. The calling task can use this value to initialize the stack segment (SP register).

The Application Loader sets stacksegbase to 0 under each of these circumstances:

- If there is no main module. (In this case, the target file does not specify a stack base).
- If the loaded code is a main module that dynamically initializes the SP and SS registers.

stack$size — A WORD specifying the number of bytes required for the loaded program's stack. The calling task can initialize the stack pointer (SP register) to the sum of stack$offset and stack$size when invoking the CREATE$TASK, CREATE$JOB, or CREATEIOJOB system call.

The Application Loader sets this value to 0 whenever both the stack$offset and stack$seg$base are 0. When all three stack-related parameters are 0 and the target file contains a main module, the loaded code must set the stack pointer (SP register) and stack segment (SS register).

datasegbase — A TOKEN containing the initial base address of the data segment (DS register).

The Application Loader sets this value to 0 under each of these circumstances:

- If the target file contains no main module.
- If the main module dynamically sets the DS register after the program starts running.

Condition Codes

The A$LOAD system call can return condition codes at two different times. Codes returned to the calling task immediately after invocation of the system call are sequential condition codes. Codes returned after the concurrent part of the system call has finished running are concurrent condition codes. The following list is divided into two parts, one for sequential codes and one for concurrent codes.

Sequential Condition Codes

The Application Loader can return any of the following condition codes to the WORD pointed to by the except$ptr parameter of this system call.

E$OK	0000H	No exceptional conditions.
EBADHEADER	0062H	The object file contains an invalid header record.
E$CHECKSUM	0064H	The header record of the target file contains a checksum error.
E$CONN$NOT$OPEN	0034H	The Application Loader opened the connection but some other task closed the connection before the loading operation was begun.
E$CONN$OPEN	0035H	The calling task specified a connection that was already open.
E$EOF	0065H	The Application Loader encountered an unexpected End-Of-File while reading a record.
E$EXIST	0006H	At least one of the following is true: • The connection parameter is not a token for an existing object. • The msg$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.
EIOHARD	0052H	A hard I/O error occurred. This means that another try is probably useless.
EIOOPRINT	0053H	The device containing the target file was off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. This means that the I/O System tried to perform the operation and failed, but another try might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.

EIOWRPROT	0054H	The volume is write-protected.
E$LIMIT	0004H	At least one of the following is true: • The calling task's job has already reached its object limit. • Either the calling task's job, or the job's default user object, is already involved in 255 (decimal) I/O operations.
E$LOADER$SUPPORT	006FH	To load the target file requires capabilities not configured into the Application Loader. For example, it might be attempting to load PIC when configured to load only absolute code.
E$MEM	0002H	The memory available to the calling task's job or the Basic I/O System is not sufficient to complete the call.
ENOTFILE$CONN	0032H	The calling task specified a connection to a device rather than to a named file.
E$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 a token for an object that is not a connection.

Concurrent Condition Codes

After the Application Loader attempts the loading operation, it returns a condition code in the except$code field of the Loader Result Segment. The Application Loader can return the following condition codes in this manner.

E$OK	0000H	No exceptional conditions.
EBADGROUP	0061H	The target file contains an invalid group definition record.
EBADSEGDEF	0063H	The target file contains an invalid segment definition record.

E$CHECKSUM	0064H	At least one record of the target file contains a checksum error.
E$EOF	0065H	The call encountered an unexpected End-Of-File.
E$EXIST	0006H	At least one of the following is true: • The mailbox specified in the response$mbox parameter was deleted before the loading operation was completed. • The device containing the file to be loaded was detached before the loading operation was completed.
E$FIXUP	0066H	The target file contains an invalid fixup record.
E$FLUSHING	002CH	The device containing the target file is being detached.
EIOHARD	0052H	A hard I/O error occurred. This means that another try is probably useless.
EIOOPRINT	0053H	The device containing the target file was off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. This means that the I/O System tried to perform the operation and failed, but another try might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOWRPROT	0054H	The volume is write-protected.
E$LIMIT	0004H	The calling task's job has already reached its object limit.
ENOLOADER$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 Application Loader to run.
ENOMEM	0068H	The Application Loader attempted to load PIC or LTL groups or segments, but the memory pool of the calling task's job does not currently contain a block of memory large enough to accommodate these groups or segments.

E$NOSTART	006CH	The target file does not specify the entry point for the program being loaded.
E$PARAM	8004H	The target file has a stack smaller than 16 bytes.
ERECFORMAT	0069H	At least one record in the target file contains a format error.
ERECLENGTH	006AH	The target file contains a record longer than the Application Loader's internal buffer. The Application Loader's buffer length is specified during the configuration of the Application Loader. See the *iRMX® I Interactive Configuration Utility Reference Manual* for information about configuring the Application Loader.
ERECTYPE	006BH	At least one of the following is true: • At least one record in the target file is of a type that the Application Loader cannot process. • The Application Loader encountered records in a sequence that it cannot process.
ESEGBOUNDS	0070H	The Application Loader created a segment into which to load code. One of the data records specified a load address outside of that segment.

The A$LOAD$IO$JOB system call reads the header record of an executable file in secondary storage and creates an I/O job. The job's initial task then performs the concurrent part of the call, which is the loading of the remainder of the file.

```
job = RQ$A$LOAD$IO$JOB(connection, pool$min, pool$max,
                       except$handler, job$flags, task$priority,
                       task$flags, msg$mbox, except$ptr);
```

Input Parameters

connection

A TOKEN for a connection to the file that the Application Loader will load. The connection must be a connection to a named file. Also, the connection must be closed, the user object specified when the connection was created must have had READ access, and the connection must have been created in the calling task's job.

The Application Loader opens the connection for sharing with readers only, so, during the loading operation, other tasks may access the file only for reading.

pool$min

A WORD containing a value the Application Loader uses to compute the pool$min size for the new I/O job that will be created for the loaded program.

pool$max

A WORD containing a value the Application Loader uses to compute the pool size for the new I/O job.

except$handler

A POINTER to a structure of the following form:

```
STRUCTURE(
        exception$handler$offset    WORD,
        exception$handler$base      TOKEN,
        exception$mode              BYTE);
```

This parameter is used as the input to RQ$CREATE$IO$JOB, when it is called to create a new job for the loaded code. If the exception handler pointer field is NIL, the new job will have the same exception handler as its parent. For more details, see the description of this parameter in RQ$CREATE$IO$JOB in the *iRMX® Extended I/O System Calls Manual.*

job$flags

A WORD specifying whether the Nucleus is to check the validity of objects used as parameters in system calls. Setting bit 1 (where bit 0 is the low-order bit) to 0 specifies that the Nucleus is to check the validity of objects. All bits other than bit 1 must be set to 0.

task$priority

A BYTE which,

- if equal to 0, indicates that the new job's initial task is to have a priority equal to the maximum priority of the initial job of the Extended I/O System.
- if not equal to 0, contains the priority of the initial task of the new job. If this priority is higher (numerically lower) than the maximum priority of the initial job of the Extended I/O System, an E$PARAM error occurs.

task$flags

A WORD indicating whether the initial task uses floating-point instructions, and whether to start the task immediately.

Set bit 0 (the low-order bit) to 1 if the task uses floating-point instructions; otherwise set it to 0.

Bit 1 indicates whether the initial task in the job should run immediately, or whether it should be suspended until a STARTIOJOB system call is issued to start it. Set it to 0 if the task is to be made ready immediately; set it to 1 if the task is to be suspended.

Set bits 2 through 15 to 0.

msg$mbox

A TOKEN for a mailbox that receives the Loader Result Segment after the loading operation is completed. This parameter is similar to the corresponding parameter in the CREATEIOJOB system call in the Extended I/O System, with these exceptions:

- You must always specify a valid mailbox TOKEN for this parameter.
- SELECTOR$OF(NIL) or zero may not be used as a value for the TOKEN.
- Each call to A$LOAD$IO$JOB requires a unique mailbox.

The second purpose of this parameter is to receive an exit message from the newly created I/O job. The description of the CREATEIOJOB system call in the *iRMX® Extended I/O System Calls* manual shows the format of an exit message.

The format of the Loader Result Segment is provided later in this description.

Output Parameters

job	A TOKEN, returned by the Application Loader, for the newly created I/O job. This token is valid only if the Application Loader returns an E$OK condition code to the WORD pointed to by the except$ptr parameter.
except$ptr	A POINTER to a WORD where the Application Loader is to place the condition code generated by the sequential part of the system call.

Description

This system call operates in two phases. The first phase occurs during the sequential part of this system call. (Refer to the *iRMX® I Application Loader User's Guide* for a discussion of the sequential and concurrent parts of an asynchronous system call.) During this first phase, the Application Loader does the following:

- Checks the validity of the header record of the target file and calculates the required memory pool that will be given to the new job.
- Creates an I/O job. This I/O job is a child of the calling task's job. The initial task of this job is a loader task that will asynchronously load the object file.
- Returns a condition code reflecting the success or failure of the first phase. The Application Loader places this condition code in the WORD pointed to by the except$ptr parameter. If the condition code is not E$OK, the job token returned is not valid and the asynchronous part of the call did not execute.

The second phase occurs during the concurrent part of the system call. This part runs as the initial task in the new job and does the following:

- Loads the file designated by the connection parameter.
- Creates the task that will execute the loaded code. If there are no errors while the file is being loaded and if bit 1 of the task$flags parameter is 0, the concurrent part makes the task in the new job ready to run. If bit 1 of task$flags is 1, the task will be suspended until an RQ$START$IO$JOB is issued for this task.
- Sends a Loader Result Segment to the mailbox specified by the msg$mbox parameter. One element in this segment is a condition code indicating the success or failure of the second phase.
- If the object file does not contain overlays, the loader task will delete itself at this point. If it does contain overlays, the loader task will be suspended, until a request to load an overlay is issued.

NOTE

This system call should be invoked only by tasks running within I/O jobs. Failure to heed this restriction causes the Application Loader to return an E$CONTEXT exception code.

Pool Size For The New Job

The Application Loader uses the following information to compute the size of the memory pool for the new I/O job:

- The pool$min parameter, as a number of 16-byte paragraphs.
- The pool$max parameter, as a number of 16-byte paragraphs.
- An Application Loader configuration parameter specifying the default dynamic memory requirements. (Refer to the *iRMX® I Interactive Configuration Utility Reference Manual* for information about configuring the Application Loader.)
- Memory requirements specified in the target file.

The Application Loader gives you three options for setting the size of the I/O job's memory pool:

1. You can set both pool$min and pool$max to 0. If you do this, the Application Loader decides how large a pool to allocate to the new I/O job. The Application Loader uses the requirements of the target file and the default memory pool size -- established when the system is configured -- to make this decision. Unless you have unusual requirements, you should choose this option.
2. You can use either pool$min or pool$max or both to override the Application Loader's decision on pool size. If the Application Loader's decision lies outside the bound(s) that you specify, the Application Loader adjusts its decision so it complies with your bounds.
3. If you set pool$max to 0FFFFH, the Application Loader allows the new I/O job to borrow memory from the calling task's job. The initial size of the memory pool is based on the pool$min parameter.

If you select Option 1 or 2, the Application Loader creates an I/O job with the minimum pool size equal to the maximum pool size. This means that the new I/O job will not be able to borrow memory from the calling task's job. If you want the I/O job to be able to borrow memory, select Option 3.

This system call is asynchronous. It allows the calling task to continue running while the loading operation is in progress. When the loading operation is finished the Application Loader sends a Loader Result Segment to the mailbox designated by the msg$mbox parameter. Refer to the *iRMX® I Application Loader User's Guide* for a detailed description of asynchronous system call behavior.

Format Of The Loader Result Segment

The Loader Result Segment has the form described below. This structure is deliberately compatible with the structure of the message returned when an I/O job exits. (See the *iRMX® Extended I/O System User's Guide* for a description of exit messages.)

```
STRUCTURE   (termination$code       WORD,
             except$code            WORD,
             job$token              TOKEN,
             return$data$len        BYTE,
             record$count           WORD,
             error$rec$type         BYTE,
             undefined$ref          WORD,
             mem$requested          WORD,
             mem$received           WORD);
```

where:

termination$code — A WORD indicating the success or failure of the loading operation.

- A value of 100H indicates that the loading operation succeeded.
- A value of 2 indicates that the loading operation failed. In this case, your system should delete the newly created I/O job; the Application Loader doesn't do so.

except$code — A WORD containing the concurrent condition code. Codes and interpretations follow this description.

job$token — A TOKEN for the newly created I/O job.

return$data$len — A BYTE that indicates the length of the remainder of the data structure minus 13 bytes. This BYTE is always set to 9.

record$count — A WORD containing the number of records read by the Application Loader. If the load operation terminates prematurely, this value indicates the last record read.

errorrectype — A BYTE identifying the type of record causing the termination of the loading operation.

- A value of 0 means that no record caused termination.
- A non-0 value indicates the type of the record that caused premature termination. Object record types are documented in the Intel publication *8086 Relocatable Object Module Formats*.

undefined$ref — This value tells whether the Application Loader found undefined external references while loading the job. An undefined external reference usually results from a linking error. The Application Loader continues to run even if a target file contains undefined external references. The value of undefined$ref depends upon the configuration of the Application Loader. (See the *iRMX® I Hardware and Software Installation Guide* and the *iRMX® I Interactive Configuration Utility Reference Manual* for information about configuring the Application Loader.)

- If the Application Loader is configured to load LTL code, as well as PIC and absolute code, undefined$ref contains the number of undefined external references the Application Loader detected during the loading operation. (Note that undefined$ref equals the number of undefined external references even if the Application Loader is loading PIC or absolute code.)
- If the Application Loader is configured to load only PIC or absolute code or only absolute code, the Application Loader sets undefined$ref to 1 or to 0. It is 1 if the Application Loader found undefined external references; otherwise, it is 0.

mem$requested — A WORD indicating the number of 16-byte paragraphs the target file requested for the new job, including the memory needed for all segments and that needed for the job's memory pool.

mem$received — A WORD indicating the number of 16-byte paragraphs actually allocated to the new job.

Condition Codes

This system call can return condition codes at two different times. Codes returned to the calling task immediately after the invocation of the system call are considered sequential condition codes. Codes returned after the concurrent part of the system call has finished running are considered concurrent condition codes. The following list is divided into two parts -- one for sequential codes and one for concurrent codes.

Sequential Condition Codes

The Application Loader returns one of the following condition codes to the WORD pointed to by the except$ptr parameter:

E$OK	0000H	No exceptional conditions.
EBADHEADER	0062H	The object file contains an invalid header record.

E$CHECKSUM	0064H	The header record of the target file contains a checksum error.
E$CONN$NOT$OPEN	0034H	The Application Loader opened the connection, but some other task closed the connection before the loading operation was begun.
E$CONN$OPEN	0035H	The specified connection was already open.
E$CONTEXT	0005H	The calling task's job is not an I/O job.
E$EOF	0065H	The Application Loader encountered an unexpected End-Of-File while reading a record.
E$EXIST	0006H	At least one of the following is true: • The connection parameter is not a token for an existing object. • The calling task's job has no global job. Refer to the *iRMX® Extended I/O System User's Guide* for a definition of global job. • The msg$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.
EIOHARD	0052H	A hard I/O error occurred. This means that another try is probably useless.
EIOOPRINT	0053H	The device containing the target file is off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. This means that the I/O System tried to perform the operation and failed, but another try might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOWRPROT	0054H	The volume is write-protected.
EJOBPARAM	8060H	The pool$max parameter is both non-zero and smaller than the pool$min parameter.

EJOBSIZE	006DH	The pool$max parameter is non-0 and too small for the target file.
E$LOADER$SUPPORT	006FH	The target file requires capabilities not configured into the Application Loader. For example, the Application Loader might be attempting to load PIC code when configured to load only absolute code.
E$MEM	0002H	The memory available to the calling task's job or the Basic I/O System is not sufficient to complete the call.
ENOLOADER$MEM	0067H	The memory pool of the newly created I/O job does not currently have a block of memory large enough to allow the Application Loader to run.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.
ENOTFILE$CONN	0032H	The specified connection is to a device rather than to a named file.
E$PARAM	8004H	At least one of the following is true: • The task$priority parameter is higher (numerically lower) than the newly created I/O job's maximum priority. This maximum priority is specified during the configuration of the Extended I/O System (if the job is a descendant of the Extended I/O System) or of the Human Interface (if the job is a descendant of the Human Interface). • The value of the except$mode field within the except$handler structure lies outside the range 0 through 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 a token for an object that is not a connection.

Concurrent Condition Codes

After the Application Loader attempts the loading operation, it returns a condition code in the except$code field of the Loader Result Segment. The Application Loader can return the following condition codes:

E$OK	0000H	No exceptional conditions.
EBADGROUP	0061H	The target file contains an invalid group definition record.
EBADSEGDEF	0063H	The target file contains an invalid segment definition record.
E$CHECKSUM	0064H	At least one record of the target file contains a checksum error.
E$EOF	0065H	The call encountered an unexpected End-Of-File.
E$EXIST	0006H	At least one of the following is true: • The mailbox specified in the msg$mbox parameter was deleted before the loading operation was completed. • The device containing the target file was detached before the loading operation was completed.
E$FACCESS	0026H	The default user of the newly created I/O job does not have "read" access to the target file.
E$FIXUP	0066H	The target file contains an invalid fixup record.
E$FLUSHING	002CH	The device containing the target file is being detached.
EIOHARD	0052H	A hard I/O error occurred. This means that another try is probably useless.
EIOOPRINT	0053H	The device containing the target file is off-line. Operator intervention is required.

EIOSOFT	0051H	A soft I/O error occurred. This means that the I/O System tried to perform the operation and failed, but another try might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOWRPROT	0054H	The volume is write-protected.
E$LIMIT	0004H	At least one of the following is true: • The task$priority parameter is higher (numerically lower) than the newly created I/O job's maximum priority. This maximum priority is specified during the configuration of the Extended I/O System (if the job is a descendant of the Extended I/O System) or during configuration of the Human Interface (if the job is a descendant of the Human Interface). • Either the newly created I/O job, or its default user, is already involved in 255 (decimal) I/O operations. • The calling task's object directory is full. • The root object directory is full.
ENOLOADER$MEM	0067H	There is not sufficient memory available to the newly created I/O job or the Basic I/O System to allow the Application Loader to run.
ENOMEM	0068H	The Application Loader is attempting to load PIC or LTL groups or segments, but the memory pool of the newly created I/O job does not currently contain a block of memory large enough to accommodate these groups or segments.
E$NOSTART	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.
ERECFORMAT	0069H	At least one record in the target file contains a format error.

ERECLENGTH	006AH	The target file contains a record longer than the Application Loader's internal buffer. The internal buffer length is specified during the configuration of the Application Loader. Refer to the *iRMX® I Hardware and Software Installation Guide* and the *iRMX® I Interactive Configuration Utility Reference Manual* for information about configuring the Application Loader.
ERECTYPE	006BH	At least one of the following is true: • At least one record in the target file is of a type that the Application Loader cannot process. • The Application Loader encountered records in a sequence that it cannot process.
ESEGBOUNDS	0070H	The Application Loader created a segment into which to load code. One of the data records specified a load address outside of the new segment.

The S$LOAD$IO$JOB system call synchronously loads an object file from secondary storage to memory and creates an I/O job for it.

RQSLOADIOJOB creates a new job using RQ$CREATE$IO$JOB and loads the specified object file. The loaded file's code becomes the initial task of the new job. The calling task is suspended during the loading operation. If the task$flags parameter specifies delayed activation, a STARTIOJOB call must be issued to start the new task. If the task$flags parameter specifies immediate activation, the task becomes ready at the end of the loading operation.

```
job = RQ$S$LOAD$IO$JOB(path$ptr, pool$min, pool$max,
                       except$handler, job$flags, task$priority,
                       task$flags, msg$mbox, except$ptr);
```

Input Parameters

path$ptr — A POINTER to a STRING containing a path name for the named file with the object code to be loaded. The path name must conform to the Extended I/O System path name syntax for named files. If you are not familiar with iRMX I path name syntax, refer to the *iRMX® Extended I/O System User's Guide.*

pool$min — A WORD containing a value that the Application Loader uses to compute the pool size for the new I/O job. See the DESCRIPTION section for details.

pool$max — A WORD containing a value that the Application Loader uses to compute the pool size for the new I/O job. See the DESCRIPTION section for details.

except$handler — A POINTER to a structure of the following form:

```
STRUCTURE (
    exception$handler$offset    WORD,
    exception$handler$base      TOKEN,
    exception$mode              BYTE)
```

The Application Loader expects you to designate an exception handler to be used both for the new task and for the new job's default exception handler. If you want to designate the system default exception handler, do so by setting exception$handler$base to 0. If you set the base to any other value, then the Application Loader assumes that the first two words of this structure point to the first instruction of your exception handler.

Set the exception$mode to tell the Application Loader when to pass control to the new task's exception handler. Encode the mode as follows:

Value	When Control Passes To Exception Handler
0	Control never passes to handler
1	On programmer errors only
2	On environmental conditions only
3	On all exceptional conditions

For more information regarding exception handlers and the exception mode, refer to the *iRMX® I Nucleus User's Guide*.

job$flags

A WORD specifying whether the Nucleus is to check the validity of objects used as parameters in system calls. Setting bit 1 (where bit 0 is the low-order bit) to 0 specifies that the Nucleus is to check the validity of objects. All bits other than bit 1 must be set to 0.

task$priority

A BYTE which,

- if equal to 0, indicates that the new job's initial task is to have a priority equal to the maximum priority of the initial job of the Extended I/O System.
- if not equal to 0, contains the priority of the initial task of the new job. If this priority is higher (numerically lower) than the maximum priority of the initial job of the Extended I/O System, an E$PARAM error occurs.

task$flags

A WORD indicating whether the initial task uses floating-point instructions, and whether to start the task immediately.

Set bit 0 (the low-order bit) to 1 if the task uses floating-point instructions; otherwise set it to 0.

Bit 1 indicates whether the initial task in the job should run immediately, or whether it should be suspended until a STARTIOJOB system call is issued to start it. Set bit 1 to 0 if the task is to be made ready immediately; set it to 1 if the task is to be suspended.

Set bits 2 through 15 to 0.

msg$mbox — A TOKEN for a mailbox that receives an exit message from the newly created I/O job. This parameter is similar to the CREATEIOJOB system call documented in the *iRMX® Extended I/O System Calls* manual, with these exceptions:

- You must always specify a valid mailbox TOKEN for this parameter.
- SELECTOR$OF(NIL) or zero may not be used as a value for the TOKEN.
- Each call to S$LOAD$IO$JOB requires a unique mailbox.

Output Parameters

job — A TOKEN, returned by the Application Loader, for the newly created I/O job. This token is valid only if the Application Loader returns an E$OK condition code to the WORD specified by the except$ptr parameter.

except$ptr — A POINTER to a WORD where the Application Loader is to place a condition code.

Description

This system call performs the same function as A$LOAD$IO$JOB. The only difference between the calls is that S$LOAD$IO$JOB is synchronous. That is, the calling task resumes running only after the call has completed its attempt to create an I/O job and a user task in that job.

The Application Loader does not necessarily have exclusive access to the file being loaded. During the loading operation, however, if other tasks are also using the file, they may access the file only for reading.

NOTE

This system call should be invoked only by tasks running within I/O jobs. Failure to heed this restriction causes the Application Loader to return an E$CONTEXT exception code.

Pool Size For The New Job

The Application Loader uses the following information to compute the size of the memory pool for the new I/O job:

- The pool$min parameter, as a number of 16-byte paragraphs.
- The pool$max parameter, as a number of 16-byte paragraphs.
- An Application Loader configuration parameter specifying the default dynamic memory requirements. (Refer to the *iRMX® I Hardware and Software Installation Guide* and the *iRMX® I Interactive Configuration Utility Reference Manual* for information about configuring the Application Loader.)
- Memory requirements specified in the target file.

The Application Loader gives you three options for setting the size of the I/O job's memory pool:

1. You can set both pool$min and pool$max to zero. If you do this, the Application Loader decides how large a pool to allocate to the new I/O job. The Application Loader uses the requirements of the target file and the default memory pool size--established when the system is configured--to make this decision. Unless you have unusual requirements, you should choose this option.
2. You can use either pool$min or pool$max or both to override the Application Loader's decision on pool size. If the Application Loader's decision lies outside the bound(s) that you specify, the Application Loader adjusts it to comply with your bounds.
3. If you set pool$upper$bound to 0FFFFH, the Application Loader allows the new I/O job to borrow memory from the calling task's job.

If you select Option 1 or 2, the Application Loader creates an I/O job with the minimum pool size equal to the maximum pool size. This means that the new I/O job will not be able to borrow memory from the calling task's job. If you want the I/O job to be able to borrow memory, select Option 3.

Condition Codes

The Application Loader returns one of the following condition codes to the WORD specified by the except$ptr parameter of this system call:

E$OK	0000H	No exceptional conditions.
EBADGROUP	0061H	The target file contains an invalid group definition record.
EBADHEADER	0062H	The object file contains an invalid header record.

EBADSEGDEF	0063H	The target file contains an invalid segment definition record.
E$CHECKSUM	0064H	At least one record in the target file contains a checksum error.
E$CONTEXT	0005H	The calling task's job is not an I/O job.
E$EOF	0065H	The call encountered an unexpected End-Of-File.
E$EXIST	0006H	At least one of the following is true: • The msg$mbox parameter is not a token for an existing object. • The calling task's job has no global job. (Refer to the *iRMX® Extended I/O System User's Guide* for a definition of global job.) • The device containing the target file was detached.
E$FACCESS	0026H	The default user object for the new I/O job does not have "read" access to the specified file.
E$FIXUP	0066H	The target file contains an invalid fixup record.
E$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.
EIOHARD	0052H	A hard I/O error occurred. This means that another try is probably useless.
EIOJOB	0047H	The calling task's job is not an I/O job.
EIOOPRINT	0053H	The device containing the target file is off-line. Operator intervention is required.

EIOSOFT	0051H	A soft I/O error occurred. This means that the I/O System tried to perform the operation and failed, but another try might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOWRPROT	0054H	The volume is write-protected.
EJOBPARAM	8060H	The pool$max parameter is nonzero and smaller than the pool$min parameter.
EJOBSIZE	006DH	The pool$max parameter is nonzero and too small for the target file.
E$LIMIT	0004H	Either the newly created I/O job or its default user object is already involved in 255 (decimal) I/O operations.
E$LOADER$SUPPORT	006FH	The target file requires capabilities not configured into the Application Loader. For example, it might be attempting to load PIC when configured to load only absolute code.
E$MEM	0002H	The target file contains either PIC segments or groups, or LTL segments or groups. In any case, the memory pool of the new I/O job does not have a block of memory large enough to allow the Application Loader to load these records.
ENOLOADER$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 Application Loader to run.
ENOSTART	006CH	The target file does not specify the entry point for the program being loaded.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.

E$PARAM	8004H	At least one of the following is true: • The task$priority parameter is higher (numerically lower) than the newly created I/O job's maximum priority. This maximum priority is specified during the configuration of the Extended I/O System (if the job is a descendant of the Extended I/O System) or of the Human Interface (if the job is a descendant of the Human Interface). • The value of the except$mode field within the except$handler structure lies outside the range 0 through 3. • The target file requested a stack smaller than 16 bytes.
E$PATHNAME$SYNTAX	003EH	The specified pathname contains one or more invalid characters.
ERECFORMAT	0069H	At least one record in the target file contains a format error.
ERECLENGTH	006AH	The target file contains a record longer than the Application Loader's internal buffer. The Application Loader's buffer length is specified during the configuration of the Application Loader. (See the *iRMX® I Hardware and Software Installation Guide* and the *iRMX® I Interactive Configuration Utility Reference Manual* for information about configuring the Application Loader.)
ERECTYPE	006BH	At least one of the following is true: • At least one record in the target file is of a type that the Application Loader cannot process. • The Application Loader encountered records in a sequence that it cannot process.
ESEGBOUNDS	0070H	The Application Loader created a segment into which to load code. One of the data records specified a load address outside of the new segment.

E$TIME	0001H	The calling task's job is not an I/O job.
E$TYPE	8002H	The connection parameter is a token for an object that is not a connection.

S$OVERLAY

In programs with overlays, the root module of the program calls S$OVERLAY to load overlay modules. The root module must be loaded using one of the system calls that create an I/O job.

```
CALL RQ$S$OVERLAY(name$ptr, except$ptr);
```

Input Parameter

name$ptr — A POINTER to a STRING containing the name of an overlay. The overlay name should have only uppercase letters, both in this string and when you specify the name in the LINK86 OVERLAY control. For information about LINK86, refer to the *8086 Family Utilities User's Guide*.

Output Parameter

except$ptr — A POINTER to a WORD in which the Application Loader will place a condition code.

Description

Root modules issue this system call when they want to load an overlay module. This call can be used with LTL code or PIC, but it cannot be used with absolute code. The *iRMX® I Application Loader User's Guide* describes overlays.

Synchronous Behavior

This system call is synchronous. The calling task resumes running only after the system call has completed its attempt to load the overlay.

File Sharing

The Application Loader does not expect exclusive access to the file containing the overlay module. However, while the overlay is being loaded, if other tasks are also using the file, they can access the file only for reading.

Condition Codes

The Application Loader returns one of the following condition codes to the calling task:

E$OK	0000H	No exceptional conditions.
E$CHECKSUM	0064H	At least one record in the target overlay contains a checksum error.
E$EOF	0065H	The call encountered an unexpected End-Of-File.
E$EXIST	0006H	The specified device does not exist.
E$FIXUP	0066H	The target file contains an invalid fixup record.
E$FLUSHING	002CH	The device containing the target file is being detached.
EIOHARD	0052H	A hard I/O error occurred. This means that another try is probably useless.
EIOOPRINT	0053H	The device containing the target overlay is off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. This means that the I/O System tried to perform the operation and failed, but another try might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
E$LIMIT	0004H	Either the calling task's job, or its default user object, is already involved in 255 (decimal) I/O operations.
E$NOMEM	0068H	The overlay module contains either PIC segments or groups, or LTL segments or groups. In any case, the memory pool of the new I/O job does not have a block of memory large enough to allow the Application Loader to load the overlay module.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.
ERECFORMAT	0069H	At least one record in the target overlay contains a format error.

ERECLENGTH	006AH	The target overlay contains a record longer than the Application Loader's maximum record length. The Application Loader's maximum record length is a parameter specified during the configuration of the Application Loader.
ERECTYPE	006BH	At least one of the following is true: • At least one record in the target overlay is of a type that the Application Loader cannot process. • The Application Loader encountered records in a sequence that it cannot process.
E$OVERLAY	006EH	The overlay name indicated by the name$ptr parameter does not match any overlay module name, as specified with the OVERLAY control of the LINK86 command.
ESEGBOUNDS	0070H	The Application Loader created a segment into which to load code. One of the data records specified a load address outside of the new segment.

INDEX

A

C

F

L

M

O

P

R

S

REQUEST FOR READER'S COMMENTS

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

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

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

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

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

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

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

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS _____ PHONE ( ) _____

CITY _____ STATE _____ ZIP CODE _____

(COUNTRY)

Please check here if you require a written reply. ☐

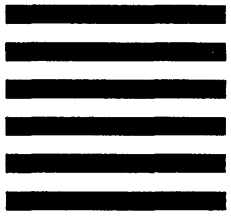
VE'D LIKE YOUR COMMENTS . . .

his document is one of a series describing Intel products. Your comments on the back of this form will elp us produce better manuals. Each reply will be carefully reviewed by the responsible person. All ɔmments 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 ɔmments. 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

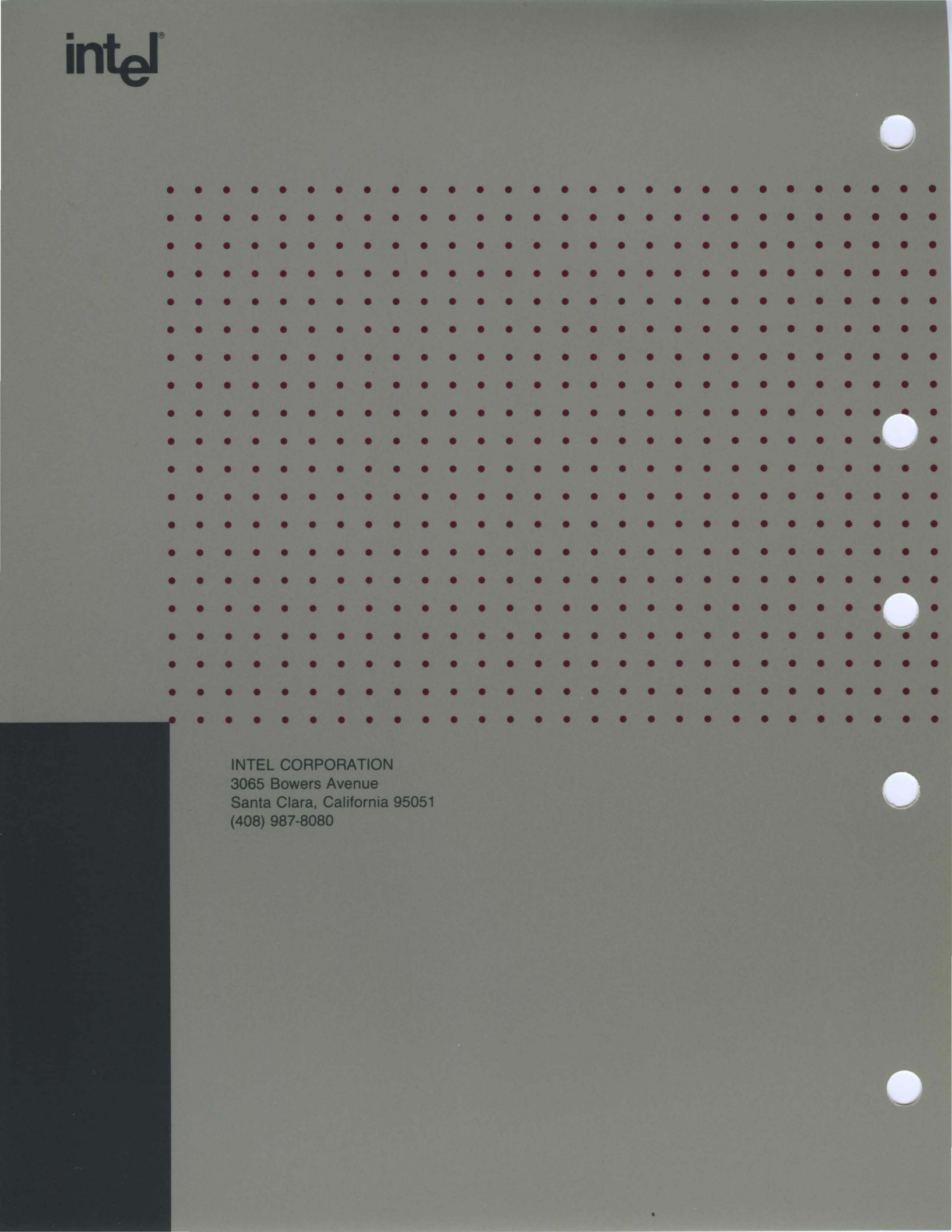
NORWAY
Intel Norway A/S
P.O. Box 92
Hvamveien 4
N-2013, Skjetten

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