iRMX™ 86 CRASH ANALYZER REFERENCE MANUAL

CONTENTS

FIGURES

CHAPTER 1
INTRODUCTION

ORGANIZATION OF THE MANUAL

This manual is divided into four chapters. Some of the chapters contain introductory or overview material which you might not need to read if you are already familiar with the Crash Analyzer. Other chapters contain invocation information and reference material to which you can refer as you analyze the problems in your software following the failure of a system or an application program. You can use this section to determine which of the other chapters you should read.

The organization of the manual is as follows:

Chapter 1	This chapter describes the organization of the manual and introduces the Crash Analyzer. It describes the features and the environment of the Crash Analyzer. You should read this chapter if you are going through the manual for the first time.
Chapter 2	This chapter explains how to configure, initialize, and load the Dumper portion of the Crash Analyzer. You should read this chapter if you are going through the manual for the first time or if you need to re-load the Dumper.
Chapter 3	This chapter describes how to invoke the Crash Analyzer. You should read this chapter to learn how to invoke the Dumper and the Analyzer. You may also want to use this chapter as a reference to the options available when you invoke the Dumper or Analyzer.
Chapter 4	This chapter describes the formats and explains the fields in the print out that the Crash Analyzer generates. The individual formats are arranged in the order they appear in the print out. You should refer to this chapter during system analysis for specific information about the displays.

REASONS FOR USING THE CRASH ANALYZER

The Crash Analyzer aids you in debugging iRMX 86 applications. It provides you with an analysis of software problems following the failure either of your system or an application program in an iRMX 86 environment. The Crash Analyzer helps you to determine the reasons for system failure by:

- Producing a dump file containing a memory image of the crash situation.
- Analyzing the dump file and producing a detailed, formatted report of the crash situation.
- Listing system objects in detail and checking for inconsistencies where possible.

The following section further describes the parts of the Crash Analyzer and the environments in which they run.

PARTS OF THE CRASH ANALYZER

The Crash Analyzer is a single product consisting of two parts:

- The Dumper which produces a disk copy of a memory image. This copy is called the dump file. The Dumper runs in the iRMX 86 application system that you are debugging.
- The Analyzer which reads the dump file and creates a formatted printout file. This printout file contains clearly labeled, formatted information about system data structures. The Analyzer runs on a Series III Microcomputer Development System. It requires a secondary storage device to contain the dump file and the formatted report.

REQUIREMENTS OF THE CRASH ANALYZER

In order to use the Crash Analyzer, the iSDM 86 Monitor must be part of your system.

HOW THE CRASH ANALYZER IS SUPPLIED

The Crash Analyzer is available on Release Diskettes in ISIS-II format or iRMX 86 format. You must perform the loading and configuration of the Dumper on a Series III Microcomputer Development System. Therefore, if you use the iRMX 86 format, you may have to copy some files to an ISIS-II format.

***

CHAPTER 2
CONFIGURING, INITIALIZING, AND LOADING THE DUMPER

In order to use the Crash Analyzer, you must configure, initialize, and load the Dumper. This chapter describes the options available when you use the iRMX 86 Interactive Configurator (ICU) to configure the Dumper into your system. It then describes how to initialize and load the Dumper module from a microcomputer development system, using the iSDM 86 Monitor. The remainder of the chapter describes what to do if you must re-load the Dumper.

USING THE INTERACTIVE CONFIGURATOR TO CONFIGURE THE DUMPER

The iRMX 86 CONFIGURATION GUIDE explains how to use the ICU to include the Dumper in your system. One of the options you have when configuring you system is whether to locate the Dumper in RAM or ROM. After you configure the Dumper, be sure to look up the address for RQ$DUMP$BOOT$INIT in the Dumper locate map SDUMPR.MP2. You will need this address to re-initialize the dumper if you have to reset the system or if you accidentally destroy data structures necessary to the Dumper and the iSDM 86 Monitor.

INITIALIZING AND LOADING THE DUMPER

When you configure and bootstrap load the system, it automatically initializes and loads the Dumper. You are now prepared to run the Crash Analyzer if it is necessary. If a system program or an application should fail, all you have to do is activate the iSDM 86 Monitor on your screen and invoke the Dumper. A common way to bring up the iSDM 86 Monitor is to press the non-maskable interrupt. This procedure causes the iSDM 86 to display a prompt (.). When you invoke the Dumper, the system automatically initializes and loads the Dumper. See Chapter 3 to learn how to invoke the Dumper.

NOTE

Avoid using the reset switch. If you do use the reset switch, you will have to re-initialize and possibly reload the Dumper.

RE-INITIALIZING AND RE-LOADING THE DUMPER

If you have to reset the system, or if you accidentally destroyed some data structures necessary to the Dumper and the iSDM 86 Monitor, you can still use the Dumper to create a valid dump file. To do this, you must re-initialize and possibly reload the Dumper. Rather than re-loading your system, you can initialize the Dumper by using the iSDM 86 Monitor go command (G) and the address for RQ$DUMP$BOOT$INIT (as listed in the Dumper locate map SDUMPR.MP2).

```
.G <bbbb>:<oooo>
```

The system responds with the Dumper sign-on message, a breakpoint, and a prompt as follows:

```
iRMX 86 Dumper initialized

*BREAK* at <bbbb>:<oooo>
```

where <bbbb> and <oooo> are base and offset addresses for internal iSDM 86 Monitor structures.

If the sign-on message does not appear, the system responds with the following error message:

```
Bad Command
```

If the system returns a "Bad Command" error message, you must re-load the Dumper into memory by entering the iSDM 86 Monitor load (L) command at your microcomputer development system as follows:

```
.L :fx:filename
```

where ":fx:" is the disk identifier that corresponds to the disk on which the ICU placed the Dumper and "filename" is the name of the file that contains the Dumper on the diskette.

After you have re-loaded the Dumper, you can enter the go command (as shown previously in this section) to re-initialize the Dumper.

***

CHAPTER 3
INVOKING THE CRASH ANALYZER

After you configure, initialize, and load the Dumper, you can invoke both the Dumper and the Analyzer portions of the Crash Analyzer any time you need them. (Refer to Chapter 2 for more information on configuring, initializing and loading.) This chapter presents a situation in which you would want to use the Dumper and the Analyzer. This situation is a general scenario of the steps you might take when an application fails. Following the scenario are detailed descriptions of how to invoke the Dumper and the Analyzer.

SCENARIO OF HOW TO USE THE CRASH ANALYZER

This section presents a general scenario of how, and in what kind of a situation, to use the Crash Analyzer. Your iRMX 86 System may differ slightly from the example used in Figure 3-1 but the procedure for using the Crash Analyzer is the same.

THE EQUIPMENT AND THE PROBLEM

Figure 3-1 shows a system consisting of the following equipment:

- A target system with an iAPX 86, 88-based processor board, memory, any necessary controllers, and a compatible terminal.
- A Series III Microcomputer Development System and a compatible printer.

Your system can be any iRMX 86 System but you must connect the Series III Microcomputer Development System to the target system with the iSDM 86 Monitor.

Suppose you are running an application on an iRMX 86 System and for some reason your application fails. You can use the Crash Analyzer to help find out why the application failed.

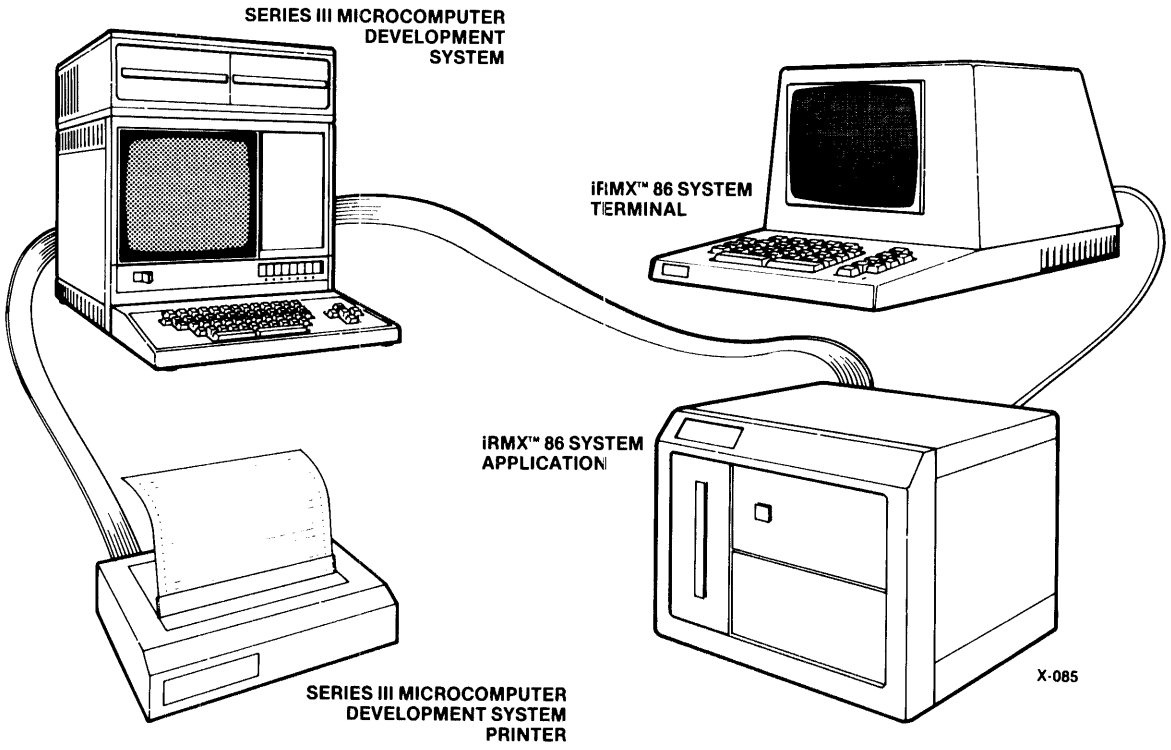


Figure 3-1. Example System

USING THE CRASH ANALYZER

This section describes the general steps you should take when you want to use the Crash Analyzer. The steps refer you to detailed explanations of the specific invocations.

1. Activate the iSDM 86 Monitor and invoke the Dumper. A common way to bring up the iSDM 86 Monitor is to press the non-maskable interrupt on your target system.

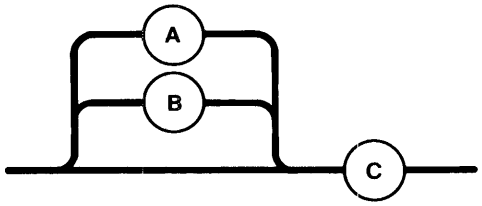
2. Invoke the Dumper on the target System. See "Invoking the Dumper" in this chapter. The Dumper uses the iSDM 86 link to create a disk file on the Series III Microcomputer Development System. This disk file (called the dump file) contains a copy of the system's memory.

3. Invoke the Analyzer on the Series III Microcomputer Development System. See "Invoking the Analyzer" in this chapter. The Analyzer reads the dump file and produces a formatted print file which it sends to the printer or a disk file. This print file contains clearly labeled information about the system data structures.

4. Use listings in Chapter 4 to help you understand the information in the print file.

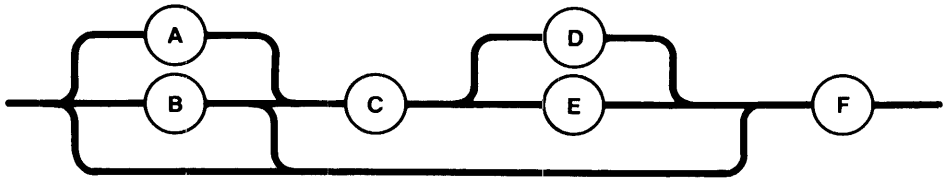
PICTORIAL REPRESENTATION OF SYNTAX

This manual uses a schematic device to illustrate the syntax of commands. The schematic consists of what looks like an aerial view of a model railroad setup, with syntactic entities scattered along the track. Imagine that a train enters the system at the left, drives around as much as it can or wants to (sharp turns and backing up are not allowed), and finally departs at the right. The command it generates in so doing consists, in order, of the syntactic entities that it encounters on its journey. The following pictorial syntax shows two ways (A or B) of reaching "C.":



x-116

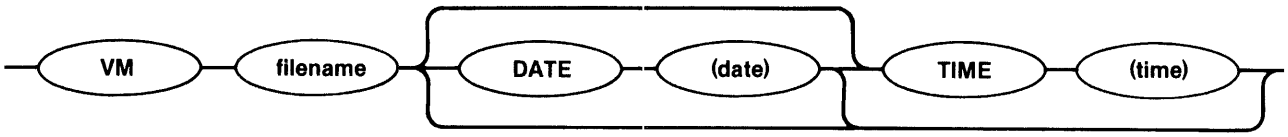
The schematics do get more complicated, but just remember that you can begin at any point on the left side of the track and take any route to get to the end as long as you do not back up. Some of the possible combinations of syntactic elements are: ACDF, BCEF, BF, AF, and F.



x-117

INVOKING THE DUMPER

You can invoke the dumper by interrupting into the iSDM 86 Monitor (on an iRMX 86 application system) and using the VM command.



X-086

PARAMETERS

filename — The name of the ISIS-II file to which you want to dump the disk copy of the system memory. The beginning portion of this name can consist of a logical name enclosed in colons (such as :F1:). This indicates the drive on which to place the file. If you omit the logical name, the Dumper places the file resides in the default drive (:F0:).

DATE — If you want the analysis header (explained in Chapter 4) to include a date, you must enter the word "DATE" immediately preceding the actual date.

(date) — The date that you invoke the Dumper. This parameter can be up to 20 characters in length and in any form you wish. The characters you enter for the date must be enclosed by parentheses. The date you enter is placed in the dumpfile and printed during analysis.

The date is an optional parameter; if you do not specify a date, the Crash Analyzer omits the it in the analysis header. See Chapter 4 for more information about the analysis header.

TIME — If you want the Analysis Header (explained in Chapter 4) to include a time, you must enter the word "TIME" immediately preceding the actual time.

(time) — The time that you invoke the Dumper. This parameter can be up to 10 characters in length and in any form you wish. The characters you enter for the time must be enclosed by parentheses. The time that you enter is placed in the dumpfile and printed during analysis.

The time is an optional parameter; if you do not specify a time, the Crash Analyzer omits the time in the analysis header. See Chapter 4 for more information about the analysis header.

The dumper displays the following message immediately after you invoke it:

```
Start iRMX 86 system dump V<x.x>
```

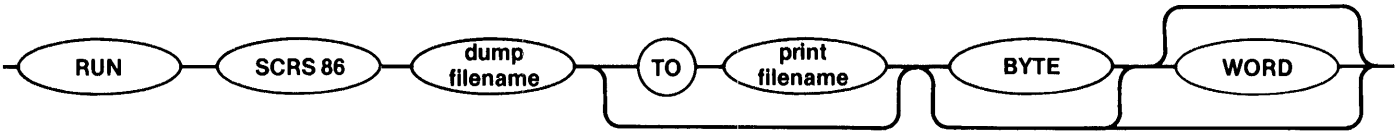
When the Dumper finishes creating the dump file, it displays the following message:

```
Dump complete to file <filename>
```

where <filename> is the file name you specified in the VM command. The iSDM 86 Monitor then issues a new prompt (.).

INVOKING THE ANALYZER

You can invoke the Analyzer on the Series III Microcomputer Development System by using the following command.



X-087

PARAMETERS

RUN	The Series III RUN command.
SCRS86	The name of the Analyzer.
dump-filename	The name of the file that is the source of the system memory image to be analyzed. This is the same file you specified when you invoked the Dumper.
TO	If you want to include a print file name, you must enter the word "TO" preceding the print file name you select.
print-filename	The name under which the Analyzer places the analyzed output. If you do not specify a "print-filename", the Analyzer uses the "dump filename" with "PRT" as the extension. Do not use the name of a device alone, unless the name specifies a device printer such as :LP:.

BYTE

An optional format in which the Analyzer may print the contents of the iRMX 86 segments. If you specify the BYTE option, the Analyzer prints the contents of the iRMX 86 segments in BYTE format. An example of the BYTE format is as follows:

```
contents:  BBBB:0000  xx  xx  xx...*aaaaaaaaaaaa*
```

where:

BBBB:0000	The base and offset address of the iRMX 86 segments.
xx	A pair of hexadecimal digits representing a byte.
a...a	The ASCII representation of the corresponding byte (if printable). If the byte value cannot be printed, the Analyzer places a period (.) in its place.

WORD

An optional format in which the Analyzer may print the contents of the iRMX 86 segments. If you specify the WORD option, the Analyzer prints the contents the iRMX 86 segments in hexadecimal WORD format. An example of the WORD format is as follows:

```
contents:  BBBB:0000  xxxx  xxxx  xxxx  xxxx...
```

where:

BBBB:0000	The base and offset address of the iRMX 86 segments.
xxxx	Four hexadecimal digits representing a word.

If you specify both BYTE and WORD, the iRMX 86 segments are displayed in both formats. If you do not specify either BYTE or WORD, the contents of the iRMX 86 segments do not appear in the print file.

ERROR MESSAGES

The following error messages appear on the your screen when you make an error in invoking the Analyzer or during a file operation. These errors cause the Analyzer to terminate all operations and display the error message.

Message	Description
Argument size exceeds 80 characters	When you invoked the Analyzer, one of the arguments exceeded 80 characters in length.
<filename>, error during <operation type> <filename>, EXCEPTION <nnnn>H <message>	The Analyzer encountered an exceptional condition when it tried to perform an operation on the file name. The <operation type> is one of the following: open create close detach read write seek The Analyzer also displays the exception code <nnnn>H and the mnemonic for the exception in <message>. Refer to the iRMX 86 Operator's Manual to find out what the exception codes mean.
<filename>, illegal file name	The file name you specified when you invoked the Analyzer is not a valid ISIS II file name.
<filename>, is not an iRMX 86 dump file	The file name you specified when you invoked the Analyzer refers to a file that was not originally created by the Dumper.
<filename>, no such file	The Analyzer cannot find the file you specified.
<keyword>, invalid keyword	You specified a format option other than WORD or BYTE when you invoked the Analyzer.
Non-blank delimiter in input string	When you invoked the Analyzer, you used a delimiter other than a blank. The only delimiter the Analyzer accepts is a blank.

Null dump file name	You did not specify a name for the dump file when you invoked the Analyzer.
Null output file name	When you invoked the Analyzer, you included "TO" but you did not specify a print file name.

***

CHAPTER 4
LISTING FORMAT

This chapter describes the format and explains the fields in the listing that the Analyzer outputs. These individual sections of the listing are arranged in the order they appear in the printout. For quick reference, this chapter includes a Table of Contents that lists the pages on which the different sections of the listing appear.

Note: this manual will not explain some of the outputs on the display field since those outputs are meant for Intel in-house use only.

LISTINGS

ANALYSIS HEADER

The section of the listing shown in Figure 4-1 identifies the file being dumped along with the time and date of the dump.

```
*************************************************************************
*
*  iRMX 86 Crash Analyzer  -  V<x.x>
*
*  Date:  <date of dump>
*
*  Time:  <time of dump>
*
*  Dumpfile:  <dumpfile name>
*
*************************************************************************
```

Figure 4-1. Analysis Header

The fields in Figure 4-1 are as follows:

<date of dump>	The date of the dump. You specified the data in this field when you invoked the Dumper.
<time of dump>	The time of the dump. You specified the data in this field when you invoked the Dumper.
<dumpfile name>	The name of the dump file. You specified this field when you invoked the Dumper.

See Chapter 3 for more information on the previous fields.

CURRENT PROCESSOR STATE

The section of the listing in Figure 4-2 displays the state of both the CPU running the system and the Numeric Processor Extension at the time you invoked the Dumper. If the registers of the processor are not available to the analyzer, it prints the message, "Registers not available" in place of the processor state.

```
%
%-------------------------------------------------------------------
%
%       Current processor state
%
%-------------------------------------------------------------------
%

    *
    *     CPU state
    *

    AX = <xxxx> SP = <xxxx> CS = <xxxx> IP = <xxxx>
    BX = <xxxx> BP = <xxxx> DS = <xxxx> FL = <Ox Dx Ix Tx Sx Zx Ax Px Cx>
    CX = <xxxx> SI = <xxxx> SS = <xxxx>
    DX = <xxxx> DI = <xxxx> ES = <xxxx>

    *
    *     NPX state
    *

    CW = <xxxx>         SW = <xxxx>      TW = <xxxx>
    IP = <xxxxx>        OC = <xxx>       OP = <xxxxx>
    ST(0) = <xxxxxxxxxxxxxxxxxxxx>
    ST(1) = <xxxxxxxxxxxxxxxxxxxx>
    ST(2) = <xxxxxxxxxxxxxxxxxxxx>
    ST(3) = <xxxxxxxxxxxxxxxxxxxx>
    ST(4) = <xxxxxxxxxxxxxxxxxxxx>
    ST(5) = <xxxxxxxxxxxxxxxxxxxx>
    ST(6) = <xxxxxxxxxxxxxxxxxxxx>
    ST(7) = <xxxxxxxxxxxxxxxxxxxx>
```

Figure 4-2. Current Processor State

The fields pertaining to the CPU in Figure 4-2 are as follows:

AX, SP, CS, IP, BX, BP, DS, FL, CX, SI, SS, DX, DI, ES	The hexadecimal values in the CPU's registers. The names of the processor's registers are as follows:

Name	Description
AX	"A" Register
SP	Stack Pointer
CS	Code Segment
IP	Instruction Pointer
BX	"B" Register
BP	Base Pointer
DS	Data Segment
FL	Flags
CX	"C" Register
SI	Source Index
SS	Stack Segment
DX	"D" Register
DI	Destination Index
ES	Extra Segment

The fields pertaining to the Numeric Processor Extension (NPX) in Figure 4-2 appear only if your system includes one.

CW, SW, TW, IP, OC, OP, — The hexadecimal value in the NPX's register. The names of the Numeric Processor Extension registers are as follows:

Name	Description
CW	Control Word
SW	Status Word
TW	Tag Word
IP	Instruction Pointer
OC	Operation Code
OP	Operand Pointer

ST(0) through ST(7) — The hexadecimal value of the NPX stack registers. The Analyzer displays these 80-bit registers in temporary real format.

JOB TREE

The section of the listing in Figure 4-3 displays the tokens of all the jobs in your system. The offspring jobs are indented to show their position in the hierarchy.

```
%
%-----------------------------------------------------------------
%
%    iRMX 86 job tree
%
%-----------------------------------------------------------------
%

        <xxxx1>
           <xxxx2>
              <xxxx3>
                 <xxxx4>
           <xxxx5>
           <xxxx6>
```

Figure 4-3. Job Tree

The fields in Figure 4-3 are as follows:

<xxxx1>	The token for the root job.
<xxxx2> through <xxxx6>	The tokens for the offspring jobs of the root job. The offspring jobs are indented three spaces to show their position in the hierarchy.

LIST OF READY TASKS

The section of the listing in Figure 4-4 displays the token for the task that is running followed by the the tokens for the ready tasks.

```
%
%-------------------------------------------------------------
%
%    List of ready tasks
%
%-------------------------------------------------------------
%
```

	Task	Priority
Running task	<xxxx>J/<yyyy>T	<aa>
Ready tasks	<xxxx>J/<yyyy>T	<aa>
	.	
	.	
	.	
	<xxxx>J/<yyyy>T	<aa>

Figure 4-4. List Of Ready Tasks

The fields in Figure 4-4 are as follows:

<xxxx>J	The token representing the job which contains the task.
<yyyy>T	The token representing either the running task or a ready task.
<aa>	The priority of the task.

Depending on what the processor was doing at the time of the dump, ROOTJ/IDLET can appear in either the running task or the ready task list. ROOTJ/IDLET represents the operating system's idle task; the idle task is a low-priority task that runs when no other tasks are running. If ROOTJ/DELET appears in the ready task list, a task requested deletion of itself or its job. See the iRMX 86 NUCLEUS REFERENCE MANUAL for more information about deleting a task or job.

LIST OF SLEEPING TASKS

The section of the listing in Figure 4-5 displays the token for the tasks that are sleeping. The sleeping tasks are shown in increasing order of their remaining sleep-time.

```
%
%-----------------------------------------------------------------
%
%    List of sleeping tasks
%
%-----------------------------------------------------------------
%
```

Task	Priority	Delay remaining	Delay requested
<xxxx>J/<yyyy>T	<aa>	<bbbb>	<cccc>
.			
.			
.			
<xxxx>J/<yyyy>T	<aa>	<bbbb>	<cccc>

Figure 4-5. List Of Sleeping Tasks

The fields in Figure 4-5 are as follows:

<xxxx>J	The token representing the job which contains the task.
<yyyy>T	The token representing the sleeping task.
<aa>	The priority of the task.
<bbbb>	The remaining time the task is required to sleep. This sleep-time is expressed in intervals of the system clock, so you must know the value of the clock interval in your system.
<cccc>	The sleep-time the task requested. This sleep-time is expressed in intervals of the system clock, so you must know the value of the clock interval in your system.

If ROOTJ/DELET appears in the list of sleeping tasks, your system was not deleting a task or a job at the time of the dump. See the iRMX 86 NUCLEUS REFERENCE MANUAL for more information about deleting a task or job.

If there are no tasks sleeping at the time of the dump, the Analyzer prints the sleeping task header and the message, "No tasks are sleeping."

LIST OF EXTENSIONS IN SYSTEM

The section of the listing in Figure 4-6 displays information about each extension in your system. It also shows the deletion mailbox for each extension along with its containing job.

```
%
%-------------------------------------------------------------
%
%    List of extensions
%
%-------------------------------------------------------------
%
```

Extension token	Extension type	Containing job	Deletion mailbox
<uuuu>	<vvvv>	<wwww>J	<xxxx>J/<yyyy>M
•			
•			
•			
<uuuu>	<vvvv>	<wwww>J	<xxxx>J/<yyyy>M

Figure 4-6. List Of Extensions In System

The fields in Figure 4-7 are as follows:

<uuuu>	The token for the extension.
<vvvv>	The WORD containing the type code for the new type. This type code was specified when the extension object was set up with system call CREATE$EXTENSION.
<wwww>J	The token for the job that contains the extension.
<xxxx>J	The token for the job that contains the deletion mailbox.
<yyyy>M	The token for the deletion mailbox set up with CREATE$EXTENSION.

LIST OF INTERRUPT TASKS

The section of the listing in Figure 4-7 displays information about your system's interrupt tasks in increasing order of interrupt level.

```
%
%-------------------------------------------------------------
%
%    List of interrupt tasks
%
%-------------------------------------------------------------
%

   Level          Task                      Data segment
                                            base

   <dd>           <xxxx>J/<yyyy>T           <zzzz>
     .
     .
     .
   <dd>           <xxxx>J/<yyyy>T           <zzzz>
```

Figure 4-7. List Of Interrupt Tasks

The fields in Figure 4-7 are as follows:

<dd> A byte containing the interrupt level that the task services. The level is encoded as follows:

Bits	Value
7	0
6-4	First digit of the interrupt level (0-7).
3	If one, the level is a master level and bits 6-4 specify the entire level number. If zero, the level is a slave level and bits 2-0 specify the second digit.
2-0	Second digit of the interrupt level (0-7), if bit 3 is zero.

<xxxx>J The token for the job that contains the interrupt task.

<yyyy>T The token for the interrupt task.

<zzzz> The token for the interrupt handler's data segment.

JOB REPORT ORGANIZATION

The Analyzer prints an entire job report for each job in your system. Because each job report consists of a number of detailed displays, this report is divided into sections as follows:

- Job Report Header
  - Information about the Job
  - Job Descriptor
- Object Directory
  Tasks Waiting for Object Lookup
- Objects Contained by Job
- Pool Report for Job
- Segments in Job
- Task Report
  - Non-Interrupt Tasks
  - Interrupt Tasks
- Mailbox Report
- Semaphore Report
- Region Report
- Extension Objects in Job
- Composites in Job
  - Extension Sub-Header
  - Composite Object Report
  - Special Composite Objects
    - Physical File Driver Connection Report
    - Stream File Driver Connection Report
    - Named File Driver Connection Report
    - Dynamic Device Information Report
    - Logical Device Object Report
    - I/O Job Object Report

JOB REPORT HEADER AND JOB INFORMATION

The section of the listing in Figure 4-8 contains the Job Report Header and the token of the job being printed. This header is followed by a "deletion pending message" and by information about the attributes of the job. Next is internal information about the job descriptor.

```
*
**************************************************************
**************************************************************
*
*    Job report, token = <xxxx>
*
**************************************************************
**************************************************************
*
   <deletion pending message>

 Current tasks  <xxxx>       Max tasks    <xxxx>      Max priority  <xx>
 Current objs   <xxxx>       Max objects  <xxxx>      Parameter obj <xxxx>
 Except handler <xxxx:xxxx>  Except mode  <xx>        Parent job    <xxxx>
 Job flags      <xxxx>

*
*    Job descriptor
*

   BBBB:0000  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>
   BBBB:0000  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>
   BBBB:0000  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>
   BBBB:0000  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>  <xxxx>
```

Figure 4-8. Job Report Header And Job Information

The fields in Figure 4-8 are as follows:

<deletion pending message>	This message is present only if there is some type of deletion pending against the job. The messages are either, "DELETION PENDING" or "FORCED DELETION PENDING."
Current tasks	The number of tasks currently existing in the job.
Max tasks	The maximum number of tasks that can exist in the job at the same time. This value was set when the job was created with the system call RQ$CREATE$JOB.

Max priority — The maximum (lowest numerically) priority allowed for any task in the job. This value was set when the job was created with the system call RQ$CREATE$JOB.

Current objs — The number of objects currently existing in the job.

Max objects — The maximum number of objects that can exist in the job at the same time. This value was set when the job was created with the system call RQ$CREATE$JOB.

Parameter obj — The token for the object the parent job passed to this job. This value was set when the job was created with the system call RQ$CREATE$JOB.

Except handler — The start address of the job's exception handler. This address was set when the job was created with the system call RQ$CREATE$JOB.

Except mode — The value that indicates when control is to be passed to the new job's exception handler. It is encoded as follows:

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

This value was set when the job was created with the system call RQ$CREATE$JOB.

Parent job — The token for the parent job of this job.

Job flags — The job flags parameter that was specified when the job was created. The bits (where bit 15 is the high-order bit) have the following meanings:

Bit	Meaning
15-2	Reserved.
1	If 0, then whenever a task in the new job or any of its descendent jobs makes a Nucleus system call, the Nucleus checks the parameters for validity.

	If 1, the Nucleus does not check the parameters of Nucleus system calls made by tasks in the new job. However, if any offspring of the new job has been created with this bit set to 0, there will be parameter checking for the new job.
0	Reserved.

The job descriptor has information which is not useful to application and system programmers. You should ignore this information.

OBJECT DIRECTORY AND TASKS WAITING FOR OBJECT LOOKUP

The section of the listing in Figure 4-9 displays the names and tokens for the objects cataloged in the job's object directory. This information is followed the uncataloged names of the objects for which a task is waiting. For each such object, the Analyzer displays the requested name, the token for the task, and the token for the containing job of the first task waiting for object lookup.

```
*
*    Object directory
*

   Maximum entries: <aaaa>  Entries used: <bbbb>

   Name              Length     Hex representation          Object
                     in hex

   <name1>           <z>        <xx xx xx xx...>            <xxxx>J/<yyyy>t
     .
   <namen>           <z>        <xx xx xx xx...>            <xxxx>J/<yyyy>t

*
*    Tasks waiting for object lookup
*

   Name              Length     Hex representation          Task
   requested         in hex

   <namea>           <z>        <xx xx xx xx...>            <xxxx>J/<yyyy>T
     .
   <namez>           <z>        <xx xx xx xx...>            <xxxx>J/<yyyy>T
```

Figure 4-9. Object Directory And Tasks Waiting For Object Lookup

The fields in Figure 4-9 are as follows:

Maximum entries	The maximum allowable number of entries this job can have in its object directory.
Entries used	The number of entries used within the directory.
<name$_1$> through <name$_n$>	The names under which the objects are cataloged. The printable characters are shown for each name in the list. Characters which cannot be printed are replaced with a period (.) in this listing.
<z>	The length of the name in bytes.

<xx xx xx xx...> The hexadecimal representation of each letter in the name.

<xxxx>J The token for the job that contains the object.

<yyyy>t The token for the object where "t" is one of the following characters that identify iRMX 86 object types:

Character	Object Type
C	composite
G	segment
J	job
M	mailbox
R	region
S	semaphore
T	task
X	extension

The fields pertaining to the tasks waiting for object lookup in Figure 4-9 are as follows:

<$name_a$> through <$name_z$> The name of the object the task has requested.

<z> The length of the name in bytes.

<xx xx xx xx...> The hexadecimal representation of each letter in the name.

<xxxx>J The token for the job that contains the task.

<yyyy>T The token for the task.

OBJECTS CONTAINED BY JOB

The section of the listing in Figure 4-10 lists the tokens for a job's child jobs, task, mailboxes, semaphores, regions, segments, extensions, and composites.

```
*
*     Objects contained by job <xxxx>
*

      Child jobs:   <xxxx>  <xxxx>  <xxxx>... <xxxx>
      Tasks:        <xxxx>  <xxxx>  <xxxx>... <xxxx>
      Mailboxes:    <xxxx>  <xxxx>  <xxxx>... <xxxx>
      Semaphores:   <xxxx>  <xxxx>  <xxxx>... <xxxx>
      Regions:      <xxxx>  <xxxx>  <xxxx>... <xxxx>
      Segments:     <xxxx>  <xxxx>  <xxxx>... <xxxx>
      Extensions:   <xxxx>  <xxxx>  <xxxx>... <xxxx>
      Composites:   <xxxx>  <xxxx>  <xxxx>... <xxxx>
```

Figure 4-10. Objects Contained By Job

The fields in Figure 4-10 are as follows:

Child jobs	The tokens for the child jobs within the job.
Tasks	The tokens for the tasks within the job.
Mailboxes	The tokens for the mailboxes within the job. A lower-case "o" immediately following a token for a mailbox means that one or more objects are queued at the mailbox. A lower-case "t" immediately following a token for a mailbox means that one or more tasks are queued at the mailbox.
Semaphores	The tokens for all the semaphores within the job. A lower-case "t" immediately following a token for a semaphore means that one or more tasks are queued at the semaphore.
Regions	The tokens for all the regions within the job. A lower-case "b" (busy) immediately following a token for a region means that a task is accessing information guarded by the region.
Segments	The tokens for all the segments within the job.
Extensions	The tokens for all the extensions within the job.
Composites	The tokens for all the composites within the job.

POOL REPORT

The section of the listing in Figure 4-11 displays information about the job's memory pool. It also shows the base address and size of any unallocated memory areas.

```
%
%-------------------------------------------------------------------
%
%     Pool report for job xxxx
%
%-------------------------------------------------------------------
%

      Pool min      <xxxx>    Pool Max     <xxxx>    Initial size  <xxxx>
      Pool size     <xxxx>    Largest seg  <xxxx>

*
*     Available pool memory areas
*

      Base                    Size

      <BBBB>                  <ssss>
       .
       .
       .
      <BBBB>                  <ssss>

      Total available         <xxxx>
      Total allocated         <xxxx>
```

Figure 4-11. Pool Report

The fields in Figure 4-11 are as follows:

Pool min	The minimum size (in 16-byte paragraphs) of the job's memory pool. This value was set when the job was created.
Pool max	The maximum size (in 16-byte paragraphs) of the job's memory pool. This value was set when the job was created.
Initial size	The initial size (in 16-byte paragraphs) of the job's memory pool.
Pool size	The current size (in 16-byte paragraphs) of the job's memory pool.
Largest seg	The number of 16-byte paragraphs in the largest segment in the job's memory pool.

The fields pertaining to the available pool memory areas in Figure 4-11 are as follows:

<BBBB>	The base address of the unallocated memory area. Each memory area is located at an offset of 0 from the given base.
<ssss>	The size of the unallocated memory area in 16-byte paragraphs.
Total available	The total amount of unallocated memory in 16-byte paragraphs.
Total allocated	The total amount of allocated memory in 16-byte paragraphs.

SEGMENTS IN JOB

The section of the listing in Figure 4-12 displays information about the segments in the pool of the job. It displays the token and the size of the segment along with the contents of the segment.

```
%
%-------------------------------------------------------------------
%
%     Segments in job <xxxx>
%
%-------------------------------------------------------------------
%

   Segment        Size
   token

   <xxxx>         <yyyy>         descriptor: BBBB:0000 <xxxx> <xxxx>...
   <deletion pending message>
                                 contents:   BBBB:0000 <xx> <xx>...*a...a*
     .
     .
     .
   <xxxx>         <yyyy>         descriptor: BBBB:0000 <xxxx> <xxxx>...
                                 contents:   BBBB:0000 <xx> <xx>...*a...a*
```

Figure 4-12. Segments In Job

The fields in Figure 4-12 are as follows:

<deletion pending message>	This message is present only if there is some type of deletion pending against the object. The messages are either, "DELETION PENDING" or "FORCED DELETION PENDING."
<xxxx>	The token for the segment.
<yyyy>	The size of the segment in 16-byte paragraphs.
descriptor	The segment descriptor contains information which is not useful to application and system programmers. You should ignore this information.
contents	If you specified the BYTE option when you invoked the Analyzer, the contents of the segment will be displayed in byte format as shown in Figure 4-13.

BBBB:0000	The base and offset address of the segment.
<xx>	A pair of hexadecimal digits representing a byte.
a	The ASCII representation of the corresponding byte (if printable). If the byte cannot be printed, the Analyzer places a period (.) in its place.

If you specified the WORD option, the contents of the segment is displayed in WORD format with 8 words to a line. The ASCII representation <*a...a*> is not displayed in the WORD format.

If you did not specify the BYTE or the WORD option when you invoked the Analyzer, the contents display does not appear.

If you specified both the BYTE and WORD option when you invoked the Analyzer, the contents field appears in both formats.

TASK REPORT

The Analyzer lists information about tasks in two different ways. Figure 4-13 shows the format for a non-interrupt task and Figure 4-14 shows the format for an interrupt task.

```
%
%-------------------------------------------------------------------------
%
%  Task report, token = <xxxx>
%
%-------------------------------------------------------------------------
%

   <deletion pending message>

 Static pri       <xx>          Dynamic pri       <xx>    Task state      <xxxx>
 Suspend depth    <xx>          Delay req         <xxxx>  Last exchange   <xxxx>
 Except handler   <xxxx:xxxx>   Except mode       <xx>    Task flags      <xx>
 Containing job   <xxxx>        Interrupt task     no

*
*  Task descriptor
*

   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
       .
       .
       .
   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>

*
*  Task stack segment
*

   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>...
```

Figure 4-13. Non-Interrupt Task Report

```
%
%-------------------------------------------------------------------------
%
%  Task report, token = <xxxx>
%
%-------------------------------------------------------------------------

   <deletion pending message>

 Static pri      <xx>          Dynamic pri      <xx>    Task state      <xxxx>
 Suspend depth   <xx>          Delay req        <xxxx>  Last exchange   <xxxx>
 Except handler  <xxxx:xxxx>   Except mode      <xx>    Task flags      <xx>
 Containing job  <xxxx>        Interrupt task    yes    Int level       <xx>
 Pending int     <xx>          Max interrupts   <xx>    Master mask     <xx>
 Slave mask      <xx>          Slave number     <xx>

*
*  Task descriptor
*

   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
       .
       .
       .
   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>


*
*  Task stack segment
*

   Tasks  SS:SP   xxxx:xxxx

   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>...
```

Figure 4-14. Interrupt Task Report

The fields in Figure 4-13 and 4-14 are as follows:

<deletion pending message>	This message is present only if there is some type of deletion pending against the object. The messages are either, "DELETION PENDING" or "FORCED DELETION PENDING."
Static pri	The current priority of the task. This value was set when the job was created with the system call RQ$CREATE$TASK.

Dynamic pri

A temporary priority that the Nucleus sometimes assigns to the task (temporarily) in order to improve system performance.

Task state

The state of the task. There are five possible states:

State	Description
ready	ready for execution
asleep	task is asleep
suspended	task is suspended
asleep/susp	task is both asleep and suspended
deleted	task is being deleted

If this field can't be interpreted, the Analyzer displays the actual hexadecimal value followed by a space and two question marks.

Suspend depth

The current number of outstanding RQ$SUSPEND$TASK system calls applied to this task without corresponding RQ$RESUME$TASK system calls.

Delay req

The number of sleep units the task requested. See the iRMX 86 NUCLEUS REFERENCE MANUAL for more information on sleep units.

Last exchange

The token for the mailbox, region, or semaphore at which the task is currently waiting.

Except handler

The start address of the task's exception handler. This value was set when the task was created with RQ$CREATE$TASK, RQ$CREATE$JOB, or RQ$CREATE$IO$JOB, or when RQ$SET$EXCEPTION$HANDLER was used.

Except mode

The value used to indicate when control is to be passed to the task's exception handler. It is encoded as follows:

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

This value was set when the task was created with RQ$CREATE$TASK, RQ$CREATE$JOB, or RQ$CREATE$IO$JOB, or when RQ$SET$EXCEPTION$HANDLER was used.

Task flags

The task flags parameter used when the task was created with the system call RQ$CREATE$TASK. The bits (where 15 is the high-order bit) have the following meanings:

Bit	Meaning
15-1	Reserved bits which should be set to zero.
0	If one, the task contains floating-point instructions. These instructions require the 8087 component for execution. If zero, the task does not contain floating-point instructions.

Containing job

The token for the job which contains this task.

Interrupt task

"No" signifies that the task is not an interrupt task. In this case, there are no more fields in the display (see Figure 4-14).

"Yes" signifies that the task is an interrupt task. In this case, there are six more fields in the display (see Figure 4-15).

Int level

The interrupt level that the interrupt task services. This level was set when the system call RQSETINTERRUPT was used.

Pending int

The number of RQ$SIGNAL$INTERRUPT calls that are pending.

Max interrupts

The maximum number of RQ$SIGNAL$INTERRUPT calls that can be pending.

Master mask

The hexadecimal value associated with the interrupt mask for the master interrupt controller. This value comes from the bits that correspond to the different master interrupt levels. Remember that bit numbers corresponds to interrupt level numbers. For example, bit 0 corresponds to interrupt level 0 and bit 7 corresponds to interrupt level 7. If the bit is set, the corresponding interrupt is disabled. For more information see the iRMX 86 NUCLEUS REFERENCE MANUAL.

Slave mask	The hexadecimal value associated with the interrupt mask for a slave interrupt controller. This value comes from the bits that correspond to the different slave interrupt levels. Remember that bit numbers correspond to interrupt level numbers. For example, bit 0 corresponds to interrupt level 0 and bit 7 corresponds to interrupt level 7. If the bit is set, the corresponding interrupt is disabled. For more information see the iRMX 86 NUCLEUS REFERENCE MANUAL.
Slave number	The programmable interrupt controller number of the slave that is referred to by the slave mask. For more information see the iRMX 86 NUCLEUS REFERENCE MANUAL.

The task descriptor has information which is not useful to application and system programmers. You should ignore this information.

The task stack segment displays the address of the stack segment:stack pointer (SS:SP) along with a hexadecimal display of the contents of the task's stack segment beginning at SS:SP. The task's stack segment contains part of the data in your task beginning at SS:SP.

MAILBOX REPORT

The Analyzer lists information about mailboxes in three different ways. The first listing (Figure 4-15) appears when nothing is queued at the mailbox, the second listing (Figure 4-16) appears when objects are queued at the mailbox, and the third listing (Figure 4-17) appears when tasks are queued at the mailbox.

```
%
%-------------------------------------------------------------------------
%
%  Mailbox report, token = <xxxx>
%
%-------------------------------------------------------------------------
%

   <deletion pending message>

  Containing job      <xxxx>              Queue discipline     <xxxx>
  Task queue head     <xxxx>              Object queue head    <xxxx>
  Object cache depth  <xx>

*
*  Mailbox descriptor
*

   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
```

Figure 4-15. Mailbox Report (Mailbox With No Queue)

```
%
%-------------------------------------------------------------------
%
%  Mailbox report, token = <xxxx>
%
%-------------------------------------------------------------------
%

   <deletion pending message>

  Containing job      <xxxx>                Queue discipline     <xxxx>
  Task queue head     <xxxx>                Object queue head    <xxxx>
  Object cache depth  <xxxx>

  Object queue  <xxxx>J/<yyyy>t     <xxxx>J/<yyyy>t     <xxxx>J/<yyyy>t...

*
*  Mailbox descriptor
*

   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
```

Figure 4-16. Mailbox Report (Mailbox With Object Queue)

```
%
%-------------------------------------------------------------------
%
%  Mailbox report, token = <xxxx>
%
%-------------------------------------------------------------------
%

   <deletion pending message>

  Containing job      <xxxx>                Queue discipline     <xxxx>
  Task queue head     <xxxx>                Object queue head    <xxxx>
  Object cache depth  <xxxx>

  Task queue       <xxxx>J/<yyyy>T   <xxxx>J/<yyyy>T    <xxxx>J/<yyyy>T...

*
*  Mailbox descriptor
*

   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
```

Figure 4-17. Mailbox Report (Mailbox With Task Queue)

The fields in Figures 4-15, 4-16, and 4-17 are as follows:

<deletion pending message> — This message is present only if there is some type of deletion pending against the object. The messages are either, "DELETION PENDING" or "FORCED DELETION PENDING."

Containing job — The token for the job that contains this mailbox.

Queue discipline — The order in which you specified the tasks (making requests from the mailbox) be queued. This order is set up with RQ$CREATE$MAILBOX. The tasks can be order in a "first-in/first-out" (FIFO) method or in a priority-based method (PRI).

Task queue head — The token for the task at the head of the queue.

Object queue head — The token for the object at the head of the queue.

Object cache depth — The maximum number of entries allowed in the high-performance queue associated with the mailbox. The size of this cache was set up when the mailbox was created with RQ$CREATE$MAILBOX.

When the list of tokens in the object queue is greater than the object cache depth, you have temporarily overflowed your high-performance queue. Succeeding objects are stored in a low-performance queue associated with the mailbox.

Object queue — A list of tokens for the objects queued at the mailbox and their containing jobs where:

<xxxx>J — The token for the job that contains the object.

<yyyy>t — The token for the object where "t" is one of the following characters that identify iRMX 86 object types:

Character	Object Type
C	composite
G	segment
J	job
M	mailbox
R	region
S	semaphore
T	task
X	extension

This list appears in the display only if there are objects queued at the mailbox.

Task queue	A list of tokens for the tasks queued at the mailbox and their containing jobs where: <xxxx>J The token for the job that contains the task. <yyyy>T The token for the task. This list appears in the display only if there are tasks queued at the mailbox.

The mailbox descriptor contains information which is not useful to system and application engineers. You should ignore this information.

SEMAPHORE REPORT

The Analyzer lists information about semaphores in two ways. The first listing (Figure 4-18) appears when no tasks are queued at the semaphore, and the second listing (Figure 4-19) appears when tasks are queued at the semaphore.

```
%
%-------------------------------------------------------------------
%
%  Semaphore report, token = <xxxx>
%
%-------------------------------------------------------------------
%

   <deletion pending message>

  Containing job     <xxxx>              Queue discipline   <xxxx>
  Task queue head    <xxxx>              Maximum value      <xxxx>
  Current value      <xxxx>

*
*  Semaphore descriptor
*

   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
```

Figure 4-18. Semaphore Report (Semaphore With No Queue)

```
%
%-------------------------------------------------------------------
%
%  Semaphore report, token = <xxxx>
%
%-------------------------------------------------------------------
%

   <deletion pending message>

  Containing job     <xxxx>                Queue discipline   <xxxx>
  Task queue head    <xxxx>                Maximum value      <xxxx>
  Current value      <xxxx>

  Task queue       <xxxx>J/<yyyy>T   <xxxx>J/<yyyy>T   <xxxx>J/<yyyy>T...

*
*  Semaphore descriptor
*

   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
```

Figure 4-19. Semaphore Report (Semaphore With Task Queue)

The fields in Figures 4-18 and 4-19 are as follows:

<deletion pending message>	This message is present only if there is some type of deletion pending against the object. The messages are either, "DELETION PENDING" or "FORCED DELETION PENDING."
Containing job	The token for the job which contains the semaphore.
Queue discipline	The way the tasks are ordered in the queue. The tasks can be ordered in a "first-in/first-out" (FIFO) method or a priority based method (PRI) when the semaphore is created with RQ$CREATE$SEMAPHORE.
Task queue head	The token for the task at the head of the queue.
Maximum value	The maximum number of units the semaphore can have. This number was set when the semaphore was created with RQ$CREATE$SEMAPHORE.
Current value	The number of units currently contained in the semaphore.

Task queue	A list of tokens for the tasks queued at the semaphore and their containing jobs where: <xxxx>J The token for the job that contains the task. <yyyy>T The token for the task. This list appears in the display only if there are tasks queued at the semaphore.

The semaphore descriptor has information which is not useful to application and system programmers. You should ignore this information.

REGION REPORT

The Analyzer lists information about regions in two ways. The first listing (Figure 4-20) appears when no tasks are queued at the region, and the second listing (Figure 4-21) appears when tasks are queued at the region.

```
%
%-------------------------------------------------------------------
%
%  Region report, token = <xxxx>
%
%-------------------------------------------------------------------
%

   <deletion pending message>

   Containing Job    <xxxx>      Queue discipline    <xxxx>
   Entered task      <xxxx>

*
*  Region descriptor
*

   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
```

Figure 4-20. Region Report (Region With No Queue)

```
%
%-------------------------------------------------------------------
%
%  Region report, token = <xxxx>
%
%-------------------------------------------------------------------
%

   <deletion pending message>

   Containing Job    <xxxx>      Queue discipline    <xxxx>
   Entered task      <xxxx>

   Task queue       <xxxx>J/<yyyy>T  <xxxx>J/<yyyy>T   <xxxx>J/<yyyy>T...

*
*  Region descriptor
*

   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
   BBBB:0000 <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx> <xxxx>
```

Figure 4-21. Region Report (Region With Task Queue)

The fields in Figures 4-20 and 4-21 are as follows:

<deletion pending message>	This message is present only if there is some type of deletion pending against the object. The messages are either, "DELETION PENDING" or "FORCED DELETION PENDING."
Containing job	The token for the job that contains the region.
Queue discipline	The way you ordered the tasks in the queue. The tasks can be ordered in a "first-in/first-out" (FIFO) method or in a priority-based method (PRI) when the region is created with RQ$CREATE$REGION.
Entered task	The token for the task that is currently accessing information guarded by the region.
Task queue	A list of tokens for the tasks queued at the region and their containing jobs where: <xxxx>J The token for the job that contains the task. <yyyy>T The token for the task. This list appears in the display only if there are tasks queued at the region.

The region descriptor contains information which is not useful to application and system Engineers. You should ignore this information.

EXTENSION OBJECTS IN JOB

This section of the listing displays the tokens for all of the extension objects contained by the job as shown in Figure 4-22. It then displays information about each extension along with its descriptor.

```
%
%-------------------------------------------------------------------------
%
%  Extension objects in job <xxxx>
%
%-------------------------------------------------------------------------

   Token  Extension  Deletion
          type       mailbox

   <aaaa>  <bbbb>    <cccc>      descriptor:       BBBB:0000 <xxxx> <xxxx>...
   <deletion pending message>                      BBBB:0000<xxxx>xxxx...
                                 composite list:   <xxxx>J/<yyyy>X
                                                   <xxxx>J/<yyyy>X...
     .
     .
     .
   <aaaa>  <bbbb>    <cccc>      descriptor:       BBBB:0000<xxxx>xxxx...
                                                   BBBB:0000<xxxx>xxxx...
                                 composite list:   <xxxx>J/<yyyy>X
                                                   <xxxx>J/<yyyy>X...
```

Figure 4-22. Extension List

The fields in Figure 4-22 are as follows:

<aaaa>	The token for the extension object.
<bbbb>	The extension type code for the extension. This code was specified when the extension was created with RQ$CREATE$EXTENSION. This extension object represents the license to create composite objects of this type./
<cccc>	The token for the mailbox to which this extension goes when it is to be deleted. This mailbox was specified when the extension was created with RQ$CREATE$EXTENSION.
<deletion pending message>	This message is present only if there is some type of deletion pending against the object. The messages are either, "DELETION PENDING" or "FORCED DELETION PENDING."

The extension descriptor contains information which is not useful to application and system Engineers. You should ignore this information.

The composite list consists of a list of composite tokens and the jobs that contain the tokens for the objects of this extension type, where:

<xxxx>J	The token for the job that contains the object.
<yyyy>X	The token for the object where "X" identifies the token as an extension.

COMPOSITE LIST REPORT

If the job contains any composite objects, the Analyzer displays a Composite List Report. The Composite List Report consists of the following sections:

- Composite List Report Header
- Extension Sub-Header
- Composite Object Report

The Composite List Report contains a composite list report header followed by one extension sub-header (Figure 4-23) for each extension type with composite objects in the job. Each extension sub-header consists of information about the extension type, the extension object, the extension's containing job, and the deletion mailbox for the extension object.

Each extension sub-header is followed by a list of Composite Object Reports. The Analyzer displays either a general composite object report or one of six special reports for Basic I/O System (BIOS) composites. The types of reports for BIOS composites are as follows:

- Physical File Driver Connection Report
- Steam File Driver Connection Report
- Named File Driver Connection Report
- Dynamic Device Information Report
- Logical Device Object Report
- I/O Job Object Report

Each of the special reports contain information from the general composite object report along with information special to the specific composite. Because some fields shown in the figures in this section are repeated, this manual avoids unnecessary repetition by explaining only those fields introduced in the figure.

COMPOSITE LIST REPORT HEADER AND EXTENSION SUB-HEADER

Figure 4-23 shows the composite list report header followed by an extension sub-header. Each sub-header contains general information concerning the extension object.

NOTE

Remember that the extension object can be contained in a different job than the one that contains the composite object. You should refer to the Extension Report in the extension's containing job for more detailed information on the extension object.

```
%
%-------------------------------------------------------------------
%
%  Composites in job <xxxx>
%
%-------------------------------------------------------------------
%

*
*  Extension type                <xxxx>
*  Extension object              <xxxx>
*  Extensions containing job     <xxxx>
*  Deletion mailbox              <xxxx>
*
```

Figure 4-23. Composite List Report Header And Extension Sub-Header

The fields in the extension's subheader (Figure 4-23) are as follows:

Extension type	The extension type code for the composite. This code was specified when the composite was created with RQ$CREATE$COMPOSITE.
Extension object	The token for the extension object that represents the license to create this type of composite.
Extensions containing job	The token for the job that contains the composite.
Deletion mailbox	The token for the mailbox to which this composite goes when it is to be deleted. This mailbox was specified when the extension was created with RQ$CREATE$EXTENSION.

GENERAL COMPOSITE OBJECT REPORT

Figure 4-24 shows the composite object report for all composites except special composites. Special composites include Physical File Driver Connection reports, Stream File Driver Connection reports, Named File Driver Connection reports, Dynamic Device Information, Logical Device Information, and I/O Job Object reports. These special composites displays appear in place of the general composite object report.

```
*
*  Composite object, token = <xxxx>
*

    <deletion pending message>
    Extension type      <xxxx>    descriptor:  BBBB:0000 <xxxx> <xxxx>...
                                               BBBB:0000 <xxxx> <xxxx>...
                                               BBBB:0000 <xxxx> <xxxx>...
    Number of slots     <xxxx>
    Object size         <xxxx>
    Component List      <xxxx>J/<yyyy>t   <xxxx>J/<yyyy>t   <xxxx>J/<yyyy>t...
```

Figure 4-24. General Composite Object Report

The fields in Figure 4-24 are as follows:

<deletion pending message>	This message is present only if there is some type of deletion pending against the object. The messages are either, "DELETION PENDING" or "FORCED DELETION PENDING."
Extension type	The extension type code for the composite. This code was specified when the composite was created with RQ$CREATE$COMPOSITE.
Number of slots	The number of positions available in the composite for tokens of component objects. This value was set when the composite was created with RQ$CREATE$COMPOSITE.
Object size	The size of the object in paragraphs.

The descriptor contains information which is not useful to application and system Engineers. You should ignore this information.

The component list consists of a list of tokens and their containing jobs for the objects that currently make up the composite, where:

<xxxx>J — The token for the job that contains the object.

<yyyy>t — The token for the object where "t" is one of the following characters that identify iRMX 86 object types:

Character	Object Type
C	composite
G	segment
J	job
M	mailbox
R	region
S	semaphore
T	task
X	extension

PHYSICAL FILE DRIVER CONNECTION REPORT

Figure 4-25 shows the listing for a connection to a physical file.

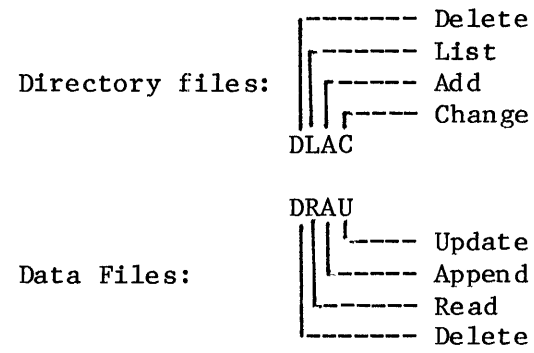
```
*
*  Physical file driver connection, token = <xxxx>
*

   <deletion pending message>
   Extension type  <xxxx>   descriptor:  BBBB:0000 <xxxx> <xxxx> <xxxx>...
                                         BBBB:0000 <xxxx> <xxxx> <xxxx>...
                                         BBBB:0000 <xxxx> <xxxx> <xxxx>...
   Containing job  <xxxx>   Conn flags     <xx>       Access          <xxxx>
   Open mode       <xxxx>   Open share     <xxxx>     File pointer    <xxxx:xxxx>
   File node       <xxxx>   Device desc    <xxxx>     DUIB pointer    <xxxx:xxxx>
   Num of conn     <xxxx>   Num of readers <xxxx>     Num of writers  <xxxx>
   File type       <xxxx>   File share     <xxxx>     Device conn     <xxxx>
```

Figure 4-25. Physical File Driver Connection Report

The fields introduced in Figure 4-25 are as follows:

Conn flags	The flags for the connection. The connection is active if bit 1 is set to one; the connection is a device connection if bit 2 is set to one.
Access	The access rights for this connection. The access rights are displayed in the same format as the display access rights for the DIR command in the Human Interface. This display uses a single character to represent a particular access right. If the file has the access right, the character appears. However, if the file does not have the access right, a dash (-) appears in the character position. The access rights along with the characters that represent them are as follows:



Open mode

The mode established when this connection was opened. The possible values are:

Open Mode	Description
Closed	Connection is closed
Read	Connection is open for reading
Write	Connection is open for writing
R/W	Connection is open for reading and writing

If this field can't be interpreted, the Analyzer displays the actual hexadecimal value followed by a space and two question marks. This value is set during a RQSOPEN or RQAOPEN system call. See the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL or the iRMX 86 EXTENDED I/O SYSTEM REFERENCE MANUAL for more information.

Open share

The sharing status established when this connection was opened. The possible values are:

Share Mode	Description
Private	Private use only
Readers	File can be shared with readers
Writers	File can be shared with writers
ALL	File can be shared with all users

If this field can't be interpreted, the Analyzer displays the actual hexadecimal value followed by a space and two question marks. This value is set during an RQSOPEN or an RQAOPEN system call. See the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL or the iRMX 86 EXTENDED I/O SYSTEM REFERENCE MANUAL for more information.

File pointer

The current contents of the file pointer for this connection.

File node

A token for a segment that the Operating System uses to maintain information about the connection. The information in this segment appears in the next two fields.

Device desc

A token for the segment that contains the device descriptor. The device descriptor is used by the Operating System to maintain information about the connections to the device.

DUIB pointer

The address of the Device Unit Information Block (DUIB). See the GUIDE TO WRITING DEVICE DRIVERS FOR THE iRMX 86 AND iRMX 88 I/O OPERATING SYSTEMS for more information on the DUIB.

Num of conn — The number of connections to the file.

Num of readers — The number of connections currently open for reading.

Num of writers — The number of connections currently open for writing.

File type — The type of file. This field is for Named files only so it does not apply (N/A) to this display.

File share — The share mode of the file. This parameter defines how the file can be opened. The possible values are:

Share Mode	Description
Private	Private use only
Readers	File can be shared with readers
Writers	File can be shared with writers
ALL	File can be shared with all users

If this field can't be interpreted, the Analyzer displays the actual hexadecimal value followed by a space and two question marks. This value is set during RQSOPEN or RQAOPEN system calls. See the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL or the iRMX 86 EXTENDED I/O SYSTEM REFERENCE MANUAL for more information.

Device conn — The number of connections to the device.

STREAM FILE DRIVER CONNECTION REPORT

Figure 4-26 shows the listing for a stream connection.

```
*
*  Stream file driver connection, token = <xxxx>
*

  <deletion pending message>
  Extension type  <xxxx>  descriptor:  BBBB:0000 <xxxx> <xxxx> <xxxx>...
                                       BBBB:0000 <xxxx> <xxxx> <xxxx>...
  Containing job  <xxxx>  Conn flags     <xx>       Access         <xxxx>
  Open mode       <xxxx>  Open share     <xxxx>     File pointer   <xxxxxxxx>
  File node       <xxxx>  Device desc    <xxxx>     DUIB pointer   <xxxx:xxxx>
  Num of conn     <xxxx>  Num of readers <xxxx>     Num of writers <xxxx>
  File type       <xxxx>  File share     <xxxx>     Device conn    <xxxx>
  Req queued      <xxxx>  Queued conn    <xxxx>     Open conn      <xxxx>
```

Figure 4-26. Stream File Driver Connection Report

The fields introduced in Figure 4-26 are as follows:

Req queued	The number of requests that are currently queued at the stream file.
Queued conn	The number of connections that are currently queued at the stream file.
Open conn	The number of connections that are currently open on the stream file.

See the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for more information about the previous fields.

NAMED FILE DRIVER CONNECTION REPORT

Figure 4-27 shows the listing for a named file connection.

```
*
*  Named file driver connection, token = <xxxx>
*

 <deletion pending message>
 Extension type <xxxx>     descriptor:  BBBB:0000 <xxxx> <xxxx> <xxxx>...
                                        BBBB:0000 <xxxx> <xxxx> <xxxx>...
 Containing job <xxxx>     Conn flags     <xx>         Access          <xxxx>
 Open mode      <xxxx>     Open share     <xxxx>       File pointer    xxxxxxxx
 File node      <xxxx>     Device desc    <xxxx>       DUIB pointer    <xxxx:xxxx>
 Num of conn    <xxxx>     Num of readers <xxxx>       Num of writers  <xxxx>
 File type      <xxxx>     File share     <xxxx>       Device conn     <xxxx>
 Fnode flags    <xxxx>     Owner          <xxxx>       File ID         <xxxx>
 File gran      <xxxx>     Fnode ptr(s)   <xxxx:xxxx>  Total blocks    <xxxxxxxx>
 Alloc size     <xxxxxxxx> File size      <xxxxxxxx>   Volume name     <xxxxxx>
 Volume gran    <xxxx>     Volume size    <xxxxxxxx>
```

Figure 4-27. Named File Driver Connection Report

The fields introduced in Figure 4-27 are as follows:

File type — The type of file. The possible values are:

File Type	Description
DIR	Directory file
DATA	Data file

Fnode flags — A word containing flag bits. Each bit has a corresponding description. If that bit is one, then the corresponding description is true; if the bit is zero, then the corresponding description is false.

Bit	Description
0	This fnode is allocated
1	The file is a long file
2	Primary fnode
3-4	Reserved
5	This file has been modified
6	This file is marked for deletion
7-15	Reserved

Owner	The ID of the owner of the file. If this field has a value of FFFF, then the field is interpreted as "WORLD." See the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.
File ID	The number of the file's fnode. The fnode is a Basic I/O System data structure containing file attribute and status data.
File gran	The granularity of the file (in volume granularity units).
Fnode ptr(s)	The values of the fnode pointers. See the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.
Total blocks	The total number of volume blocks currently used for the file; this includes indirect blocks. See the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.
Alloc size	The total size (in bytes) allocated to the file. See the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.
File size	The size (in bytes) of the file. See the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information.
Volume name	The name of the volume.
Volume gran	The granularity (in bytes) of the volume.
Volume size	The size (in bytes) of the volume.

DYNAMIC DEVICE INFORMATION REPORT

Figure 4-28 shows the information the Analyzer displays when a file has a dynamically created Device Unit Information Block (DUIB).

```
*
*  Dynamic device information for connection  <xxxx>
*

 File drivers    <xxxx>    Device gran   <xxxx>    Device size   <xxxx>
 Device functs   <xxxx>    Device name   <xxxx>
```

Figure 4-28. Dynamic Device Information Report

The fields introduced in Figure 4-28 are as follows:

File drivers — The validity of the file driver. The bits are associated with the file drivers as follows:

Bit	File Driver
0	physical
1	stream
3	named

Device gran — The value of the the volume granularity specified when the volume was formatted.

Device size — The number of bytes of information that the device-unit can store.

Device functs — The I/O function validity for this device-unit. The bits associated with the functions as follows:

Bit	Function
0	READ
1	WRITE
2	SEEK
3	SPECIAL
4	ATTACH DEVICE
5	DETACH DEVICE
6	OPEN
7	CLOSE

Device name — The name of the DUIB.

See the GUIDE TO WRITING DEVICE DRIVERS FOR THE iRMX 86 I/O SYSTEM for more information concerning the previous fields.

LOGICAL DEVICE OBJECT REPORT

The listing in Figure 4-29 shows the device names and system logical names of the logical device composite object.

```
*
*  Logical device object, token = <xxxx>
*
   <deletion pending message>
  Extension type  <xxxx>    descriptor  BBBB:0000  xxxx  xxxx  xxxx...
                                        BBBB:0000  xxxx  xxxx  xxxx...
  Containing job  <xxxx>    Physical conn  <xxxx>    File driver    xx
  Owner ID        <xxxx>

                        Name          Length    Hex representation
  Device name           <aaaaaaa>     <bb>      <xx xx xx xx xx xx...>
  Sys logical name(s)   <aaaaaaa>     <bb>      <xx xx xx xx xx xx...>
```

Figure 4-29. Logical Device Object Report

The fields introduced in Figure 4-29 are as follows:

Physical conn	The token for the physical connection.
Device Name	The 1-to 14-character name under which the logical device object is cataloged. This name was specified when RQ$LOGICAL$ATTACH$DEVICE was called.
Sys logical name(s)	The 1-to 14-character name under which the the system logical name is cataloged. This name was specified when RQ$LOGICAL$ATTACH$DEVICE was called.
<bb>	The length of the device name or the system logical name. This name was specified in the DUIB during Basic I/O System configuration.
<xx>	The hexadecimal representation of each letter in the device name or the system logical name.

I/O JOB OBJECT REPORT

The section of the listing in Figure 4-30 displays information about exit messages in I/O job objects.

```
*
*  I/O job object, token = <xxxx>
*

   Extension type       <xxxx>     descriptor:  BBBB:0000  <xxxx>  <xxxx>...
                                                BBBB:0000  <xxxx>  <xxxx>...
   Exit message token   <xxxx>
   Exit message mbx     <xxxx>
```

Figure 4-30. I/O Job Object Report

The fields introduced in Figure 4-30 are as follows:

Exit message token	The token for the segment containing the exit message.
Exit message mbx	The mailbox that contains the exit message segment.

See the iRMX 86 EXTENDED I/O SYSTEM REFERENCE MANUAL for more information about the previous fields.

SUMMARY OF ERRORS

Figure 4-31 shows the header for the summary of errors. This summary lists all error messages that the Analyzer encountered during the analysis. The summary also includes the page number of the listing on which the error occurred. For more information on error messages and their meanings, see the section in this chapter entitled, "Error Messages."

```
%
%-------------------------------------------------------------------
%
%  Summary of errors detected by analysis
%
%-------------------------------------------------------------------
%
   error message                          Page xx
        .                                   .
        .                                   .
        .                                   .
   error message                          Page xx
```

Figure 4-31. Summary Of Errors

The fields in Figure 4-31 are as follows:

error message	The error message(s) that the Analyzer detected during analysis.
page	The page number of the listing on which the error message is printed.

ERROR MESSAGES

The Analyzer can detect two kinds of errors:

- Operational Errors

  Errors that occur when you invoke the Analyzer or errors that occur during file operations. These errors are described in Chapter 3.

- Dump File Errors

  Errors within the dump file.

This section lists the Dump File Errors error messages and their descriptions. Dump file errors are errors that the Analyzer detects within the dump file. The Analyzer prints these errors in the section of the listing in which they occur. It also prints the error messages along with the page numbers on which they occur in the section of the listing entitled "Summary of Errors Detected by Analysis."

Message	Description
Nucleus entry or data segment corrupted, analysis terminated	The Analyzer uses the Nucleus interrupt vector to locate the Nucleus code segment. It then uses the code segment to find the data segment. The Analyzer uses the data segment as the basis for analysis. If any item in this chain is damaged, the Analyzer cannot function correctly.
Internal iRMX 86 <type> field corrupted at <BBBB>:<0000>	The Analyzer discovered an error in an internal operating system structure of <type> BYTE, WORD, or POINTER. The problem is located at base <BBBB> and offset <0000>.
Internal iRMX 86 <type> field corrupted at <BBBB>:<0000> Object token:<cccc>	The Analyzer discovered an error in an internal operating system structure of <type> BYTE, WORD, or POINTER. The problem is located at base <BBBB> and offset <0000>. The error is in the object whose token is <cccc>.
Stack overflow	This message appears when a task overflows its stack segment.
Registers not available	The processor's registers were not available to the Dumper. This message appears only under the "Current Processor State" portion of the listing.

Task stack segment not distinct: above display may contain other data in addition to stack	The task's stack is in a segment that was not allocated when the task was created. This message appears following the task segment listing because the Analyzer cannot distinguish stack data from other data in the segment. Therefore, this particular message indicates a lack of information necessary to the Crash Analyzer rather than a problem.
Task SS:SP not known: stack segment not displayed	The Analyzer could not find a valid stack segment:stack pointer (SS:SP). This message appears following a task segment display. This particular message indicates a lack of information necessary to the Crash Analyzer rather than a problem.
Unable to locate complete <block name>. Missing area is <address> for <size> bytes	The Analyzer could not find the entire <block name>. The block name is one of the following: connection object job directory task stack NPX save area composite list mailbox cache logical device object segment contents The <address> is the base and offset address of the missing information. The <size> is the size of the missing information in hexadecimal.
Unable to located job pool information	This message appears in the pool report for a job when the Analyzer cannot find the information describing a job's pool.
Unable to locate list of <object type> in job <job token>	The Analyzer cannot find the list of a particular object type for a job. This message appears in place of the list of an object type in the section entitled, "Objects Contained by Job" The <object type> is one of the following: child jobs tasks mailboxes semaphores regions segments extensions composites

	The <job token> is the token for the job in which the Analyzer cannot find the object type.
Unable to locate file node for <connection token>	This message appears when the Analyzer is unable to read the contents of the fnode because the pointer to the fnode has been destroyed.
Unable to locate device descriptor for connection <connection token>	This message appears when the Analyzer cannot find the device descriptor for a connection. This may happen because one of your tasks wrote over and internal data structure.
Unable to locate object queued on mailbox <token>. Token of missing object is <object token>	This message appears in the "Mailbox Report" when the Analyzer cannot find an object queued at the mailbox.
End of stack segment not known; stack segment not displayed	This message appears in the "Task Report" when the Analyzer cannot find the end of the stack segment.

LINK ERROR

The iRMX 86 Operating System maintains tokens in doubly-linked lists. So, whenever a listing contains a token, the Analyzer automatically checks the validity of that token by looking at the token's forward links.

A forward link error means that the iRMX 86 data structures have been damaged or destroyed. The most common reason for this problem is overwriting. You or one of your tasks may have accidentally written over part of the Operating System's data structures and/or code. Another possible reason for the problem (if you are using a non-maskable interrupt) could be that you interrupted the Nucleus while it was setting up the links.

If a token's forward link is bad, the Analyzer generates a forward link error message along with the information that the particular listing usually displays. The forward link error message is as follows:

```
Forward link ERROR:  <aaaa> --> <bbbb>  ?<cccc> <-- <bbbb>
```

The arrows represent links. A right pointing arrow represents a forward link. The object with the token <aaaa> is linked forward to the object with token <bbbb>. The object with the token <bbbb> should be linked back to the object with the token <aaaa> rather than <cccc>. Therefore, the Crash Analyzer assumes the link from <aaaa> to <bbbb> is incorrect and terminates the analysis of the objects in the portion of the listing in which the error appears.

***

INDEX

Primary references are underscored.