N O T I C E

THIS DOCUMENT HAS BEEN REPRODUCED FROM MICROFICHE. ALTHOUGH IT IS RECOGNIZED THAT CERTAIN PORTIONS ARE ILLEGIBLE, IT IS BEING RELEASED IN THE INTEREST OF MAKING AVAILABLE AS MUCH INFORMATION AS POSSIBLE

JPL PUBLICATION 77-37, Revision 1

VICAR Image Processing System
Guide to System Use

J. B. Seidman
A. Y. Smith

```
(NASA-CR-162693)  VICAR IMAGE PROCESSING               N80-16766
SYSTEM GUIDE TO SYSTEM USE (Jet Propulsion
Lab.)  77 p HC A05/MF A01             CSCL 09B
                                                       Unclas
                                             G3/61     46964
```

December 1, 1979

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
FEB 1980
RECEIVED
NASA STI FACILITY
ACCESS DEPT.

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

JPL PUBLICATION 77-37, Revision 1

VICAR Image Processing System
Guide to System Use

J. B. Seidman
A. Y. Smith

December 1, 1979

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

PREFACE

The work described in this document was performed by the Observational Systems Division of the Jet Propulsion Laboratory, California Institute of Technology. This document is a revision of the original issued on October 1, 1968, and revised in May 1977 (JPL Document 77-37).

PRECEDING PAGE BLANK NOT FILMED

ABSTRACT

This document is designed both to instruct the new user in the use of the VICAR (Video Image Communication and Retrieval) System and to serve as a reference manual for the more experienced user. The document has been divided into nine sections which describe the functional characteristics and operating requirements of the VICAR System.

Section 1 presents a general statement regarding the history of the VICAR System.

Section 2 lists other documents which the user will find pertinent to the System.

Section 3 provides a general overview of the operation of the VICAR System.

Section 4 defines some of the major concepts used by the System.

Section 5 presents the rules for general VICAR job setup.

Section 6 details the use and function of VICAR control statements. This section constitutes the main body of this document.

Section 7 describes the use of the EVIL2LIB Procedure Library.

Section 8 presents an introduction to TTM usage.

Section 9 lists error meassages which the user is liable to encounter and appropriate actions to be taken in response to these messages.

CONTENTS

APPENDIXES

SECTION 1

INTRODUCTION

The VICAR system is a set of computer programs designed to facilitate the acquisition, processing, and recording of digital image data. VICAR was developed at the Jet Propulsion Laboratory in the late 1960's to process picture data produced by the planetary exploration program. Its application has since been expanded to all areas in which picture processing may be desired, including astronomy, Earth resources, land use, biomedicine, and forensics. System objectives are ease of use by personnel who may not be expert programmers, and simplified programming for future expansions.

VICAR was originally designed for operation with the IBM 360/44 Programming System (44PS). It was subsequently modified to run with the IBM OS/360 operating system. It will run with all versions of OS including virtual storage versions. At the present time there are known to be about one dozen installations in the world using VICAR.

Image processing with VICAR is performed with the aid of the VICAR language. The VICAR language consists of a series of statements with which the user describes the source of the original data, the desired sequence of processing steps, and the destination of the processed data. The user in effect writes a "program" in the VICAR language to perform his job. Only two fixed Job Control Language (JCL) statements precede the VICAR statements, virtually eliminating the need for knowledge of JCL. The VICAR language is simpler and easier to use than JCL.

Data processing under VICAR is performed by "application programs" residing in a program library that forms a part of the VICAR system. In order to accomplish a prescribed operation on an image, the user must identify an application program, or a sequence of programs, which perform the job. The VICAR system includes in its program library an extensive assortment of general application programs which can be used to perform a wide variety of functions. These functions are written in standard programming languages, primarily Fortran and Assembly. The program generally can be manipulated for specific processing through the use of control parameters that can be specified by the user in VICAR language statements.

New technology and changing applications require the addition of new programs to the library. The VICAR sytem provides a standard set of subroutines for performing input/output and access to user-supplied control parameters. These subroutines comprise the interface between the application programs and the VICAR system. They are designed to reduce some of the routine labor involved in writing an application program.

VICAR is primarily a batch-oriented system, which requires that the entire sequence of operations be described before processing can begin. Once the sequence has begun, the user cannot influence the course of the processing. There exists a related interactive system called LIBEXEC, which allows the selection of each application program dynamically after previous processing has occurred. LIBEXEC makes use of the same program library as VICAR. As of the date of this publication, LIBEXEC has not been distributed outside JPL.

SECTION 2

PREREQUISITE PUBLICATIONS

VICAR is designed so that the user needs very little knowledge of the operating system. However, the following publications may prove to be useful.

(1) "IBM System/360 Operating System: Introduction," IBM Publication GC28-6534.[1]

(2) "IBM System/360 Operating System: Job Control Language Reference," IBM Publication GC28-6704.[1]

(3) "IBM System/360 Operating System: Messages and Codes," IBM Publication GC28-6631.[1]

(4) Stephen Caine and E. Kent Gordon, TTM: A Macro Language for Batch Processing, Programming Report No. 8, Booth Computing Center, California Institute of Technology, Pasadena, Calif. (ongoing)

In addition, the user will require user guides which describe the function and detailed usage requirements for the various application programs available in the VICAR program library.

[1]If OS/360 is not the operating system utilized, equivalent applicable publications should be referred to.

SECTION 3

OVERVIEW OF VICAR

This section describes generally how the VICAR System programs function in relation to each other. It is not essential to use of the system, but a general understanding of the system operation should prove helpful.

A VICAR image processing job actually consists of two computer jobs. The first computer job, sometimes called the VTRAN job, is initiated by the user. It processes the VICAR language statements and, based on their specific content, generates a second job which is submitted to the operating system to be processed. The second job, sometimes called the "X-job," performs the actual image-processing functions. In most installations, the second job is submitted automatically without intervention by the user. The user thus receives two job listings for each job he submits. Normally the first listing can be discarded unless it indicates an error. The second listing contains all pertinent information about the processing which has been performed. It should be noted that the second job will not be generated if an error, such as failing to adhere to VICAR format conventions, has been detected.

VICAR processing is initiated by an OS JCL statement, which invokes the VICAR catalogued procedure. This procedure, in turn, executes three OS job steps. The first step invokes the program TTMSA, which functions as a preprocessor, under control of the TTM library string EVIL2. The preprocessor scans the control statements for preprocessor statements, each of which is translated to zero or more non-preprocessor statements. Output from the preprocessor is passed to the second step which invokes VTRAN, the main VICAR language translator program. VTRAN generates JCL statements and "task queues" for the second job. The last step of the VICAR procedure passes the JCL and task queue to the operating system internal reader.

The second job consists of one or more job steps, as determined by VICAR control statements. Each step is processed as follows. The operating system processes the JCL statements to allocate the necessary permanent and temporary disk data sets, tapes, and special devices needed for the processing. Control is then passed to the VICAR System program VMAST. VMAST contains the I/O and parameter service routines used by all application programs. Immediately after receiving control, VMAST passes control to VICAR System program VMJC. VMJC reads the task queue to determine what processing is to be done. For each task, VMJC sets up the VICAR control blocks in VMAST according to the particular data sets to be used, converts the program parameters to internal format, and performs standard processing of the VICAR image labels. Finally VMJC passes control to the application program specified for the task. When the application program terminates, control returns to VMJC, which repeats the process until the task queue is exhausted.

SECTION 4

THE VICAR ENVIRONMENT

This section describes the characteristics of the data elements which are manipulated by VICAR. We will distinguish the following entities which form a part of the VICAR environment: image, label, data set, tape, and program parameters. A fairly clear comprehension of these concepts is a prerequisite for understanding this manual.

4.1 IMAGE

An image is a set of data to be processed by an application program. The data normally represent a visual "picture," although this is a matter of interpretation, since an image may take various forms. For example, a VICAR "image" may represent the topography of a section of the Earth's surface, or it may represent the population of each of a collection of census districts.

VICAR presently has no standard way of handling multispectral data. However, programs which process multispectral data adopt one of two common conventions. In one, each spectral band is stored as a separate image. Programs processing data in this form require one input image for each spectral band to be processed. This convention is similar to what is often called "band sequential." The second convention is called MSS format and stores all the spectral data from one scanner line on one image line, one band at a time. This convention is equivalent to what is usually called "line sequential."

The format of image data is very simple. An image can contain between 1 and 2^{31}-1 "lines," each of the same length. In the "picture" interpretation, each line represents one raster scan line. The sequence of lines represents the sequence of raster scan lines beginning with line 1 at the top of the picture and proceeding down. Within each line is a sequence of bytes whose length is fixed for a given image. The length of a line may be from 1 to 32767 bytes. In the picture interpretation, the data represents the brightness value of each sample or "pixel," on the raster scan line. Every pixel value is represented by 1, 2, 4, or 8 bytes. Most commonly one byte corresponds to one pixel, the unsigned binary value representing the brightness. The possible representations for pixel value are given in Table 4-1. It should be emphasized once more that these interpretations are conventional but not mandatory.

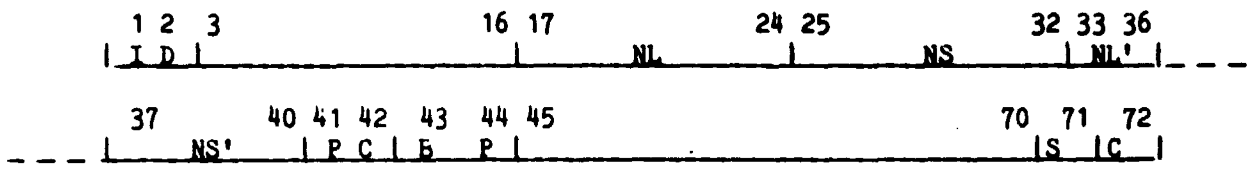
Table 4-1. VICAR Pixel Representations

Name	Pixel Code	Bytes/Pixel	Format
Byte	L	1	Unsigned binary integer (0 to 255)
Halfword	I	2	Signed, two's-complement binary integer (-32,768 to +32,767)
Fullword	I	4	Signed, two's-complement binary integer
Real	R	4	IBM floating point
Complex	C	8	A pair of "real" values

Images residing on disk must have a standard VICAR label in order to be processed by the VICAR system and by most application programs, unless the program's name begins with "V." Images residing on tape need not have a standard VICAR label, but in order to be processed, they must be moved to disk and labeled. VICAR provides a way to add a label and copy the simple tape formats listed in Appendix A. More complex formats necessitate a special purpose program, called a "logging program." The logging program, whose name starts with "V", reads the data and converts it to standard VICAR format, adding the standard VICAR system label and often several lines of text extracted from the original data. The following section describes the VICAR label in detail.

4.2 THE VICAR LABEL

An image may or may not have a "standard" VICAR label preceding it. When it does, the image is said to be in standard VICAR format. A label consists of one or more label lines preceding the image data lines. These label lines are always 72 bytes in length, and usually consist of textual data. Label lines are grouped into label records 360 bytes long consisting of 5 label lines each. The last label record may have fewer than 5 label lines. The first label line is called the VICAR system label and must always be present. It contains the format of the image and its size (number of lines and number of bytes per line). The current format in use for the VICAR system label is shown in Figure 4-1.



All fields are EBCDIC characters. Undefined fields should be blanks.

ID - "77". This label identifier distinguishes this new label format from any previously in use.

NL - number of video lines (or logical data records, excluding the entire label) in the image (data set), right justified, in decimal.

NS - number of bytes in each video line (or logical record), right justified, in decimal.

NL' - same as NL if less than 10000; otherwise all blanks.

NS' - same as NS if less than 10000; otherwise all blanks.

PC - pixel code; legal values are:

"b̸L"	unsigned binary integer
"b̸I"	two's-complement signed binary integer
"b̸R"	single precision 360 floating point
"b̸C"	two single-precision floating point values (complex data)
"b̸b̸"	(two blanks) - no implication about pixel format.

BP - number of bytes per sample, right justified, in decimal; or blanks.

S - "S"; identifies label as a "system label."

C - continuation character, "C" if more labels follow, "L" in the last label.

Figure 4-1. VICAR Standard System Label

The label lines following the system label conform to the following specifications. Character 72 is used to indicate continuation of the label: a "C" (EBCDIC) indicates another label line follows, and anything else (normally "L") indicates the last label. Character 71 indicates a general category into which the label line falls, according to the scheme in Table 4-2.

Table 4-2. VICAR Label Line Categories

Character 71	Type	Notes
blank	Any	1, 5
S	System	2, 6
G	History 1	2, 3
H	History 2	1
P	Parameters	2
U	User annotation (printed)	1
A	User annotation (not printed)	2
E	Program generated (character)	1
D	Program generated (non-character)	4

Notes:

1. The label line is normally printed when the label is printed, unless it is the first label line.

2. The label line is normally not printed, but may be printed by special program or program option.

3. History 2 contains a subset of the data in History 1. History 1 may not be present.

4. Label line should not be printed directly because it may not be in character format.

5. Any label line could have a blank in character 71, but the correct non-blank character is preferred.

6. The System label must be the first label line; the first label line is assumed to be the System label, regardless of character 71.

4.3 DATA SET

The term "data set" is ambiguous in common usage associated with VICAR; it has two distinct meanings. The first meaning is a "set of data." This meaning includes the concept of image as previously defined, as VICAR images are often referred to as VICAR data sets. The second meaning refers to a storage place on a disk in which an image can be stored. This section will further define the second of these two concepts. In common usage the term data set is often used for both meanings. The reader must infer from the context which meaning is intended.

A data set is an area on a disk storage device allocated to hold a single set of data. The data set is referenced by a name which the user may specify. Names for data sets may be up to 44 characters in length and must conform to rules specified in Ref. 2 in Section 2 as well as to any local installation rules.

Data sets are characterized by a block size and a record length. The record length in bytes is selected by the VICAR user, who must ensure that it is not less than the image line length for each image to be stored. Since images on disk must nave a VICAR label, the record length must not be less than 360 bytes, the length of a VICAR label record. The VICAR System will ensure that this requirement is met. The data set block size must be an integral multiple (greater than zero) of the record length. The user may allow the VICAR System to select a block size, or he may specify one of his own. The block size must be no greater than the track length for the specific type of disk being used.

Data sets are allocated with a specific number of records selected by the user. The number of records selected should equal or exceed the number of label records plus the number of image lines for each image to be stored in the data set.

4.4 TAPE

A tape (magnetic tape) is a physical storage medium which can store a number of separate images. Up to 999 images can be stored on a tape for access by the VICAR System. Unlike separate images stored on a disk, tape images are not referenced by unique names. Instead, the entire tape is given a symbolic name and an individual image is referenced by a unique file number indicating its sequential position on the tape.[1] A special record called an end-of-file (EOF) mark, or tape mark, separates sequential images on the tape. The last image on the tape is followed by two such EOF marks.

[1]Some syntactical forms in the VICAR language allow only 2-digit file sequence numbers. However, there is always a way to process 3-digit file sequence numbers.

Under VICAR a tape image can be processed in two ways: either by first copying it to a disk data set, or by causing the application program to read the image directly from the tape during processing. Because of the sequential nature of tape images, only one tape image at a time can be read from a given tape.

4.5 PROGRAM PARAMETERS

Program parameters are, in a sense, another form of data on which an application program operates. They are distinguished from image data in two ways: by their function and by the way in which they are usually passed to the application program under the VICAR system. Functionally, program parameters are used to control the specific way that an application program processes image data. Their usage can be as varied as the application programmer wishes, but there are two common uses. One is to select one of several "modes" in which the program can operate. The other is to provide auxiliary numerical data to be used in processing image data.

The second distinction of program parameters is that they are usually written in the syntax of the VICAR language. They are included by the user in his VICAR language "program" as a description of the processing to be performed. This is in contrast to the image data which is usually written by an application program or an outside source; the user only describes to the VICAR system where to find image data.

It should be noted that it is also possible for program parameters to be written by an application program directly onto a data set. The parameter data may even exhibit some attributes of a VICAR image, including a fixed "line" length and a VICAR label. Only a few application programs have been written to accept parameters in this form. Specific programs with this capability can be found through the program user guides. (A special program, PAR, in effect provides this capability to any other program.)

SECTION 5

VICAR JOB SETUP

Syntax notation used in subsequent pages of this document is as follows:

> Braces { } indicate that a choice must be made from among the optional parameters indicated.
>
> Brackets [ ] indicate that the field or subfield is optional.
>
> Characters presented in upper case are required and must be coded exactly as shown.
>
> An ellipsis . . . indicates that additional parameters (subfields) may be coded.

A VICAR job is normally initiated by submitting to the computer a card deck containing the VICAR control statements and the job control statements which cause the operating system to give control to VICAR. The VICAR job has the following form.

```
//jobname      JOB . . .
//             EXEC VICAR[,DISP=SHR]
.
.
.
VICAR control statements
.
.
.
```

Additional information on the job card is installation dependent. An optional job termination card with // in columns 1-2 and blanks in columns 3-72 may be used at the end. The optional parameter DISP=SHR, if used, allows permanently allocated data sets to be processed by two or more VICAR jobs concurrently. Normally, only reading of those data sets should be done in such jobs. Example jobs may be found in Appendix D.

The JCL statements are processed by the operating system job control processor and invoke cataloged procedure VICAR. Section 3 gives a thorough discussion of the subsequent steps that occur in a VICAR job.

SECTION 6

VICAR CONTROL STATEMENTS

This section constitutes the core of this document. It describes in detail the syntax and usage of each type of VICAR control statement.

Due primarily to the implementation of the VICAR system, there are two broad classes of VICAR control statements: standard statements and preprocessor statements. These statements may be freely intermixed as described subsequently. However, it is convenient to describe them separately. Therefore, sections 6.1 to 6.3 describe the standard statements, while sections 6.4 and 6.5 describe the preprocessor statements.

6.1 STANDARD CONTROL STATEMENT FORMAT

VICAR control statements are designed for an 80-column punched card format. Statements may start in column 1 and cannot extend past column 71. Except for a few statement types, statements may be continued to another card by placing any non-blank character in column 72, or by making the last non-blank character before column 72 a percent sign (%). Up to 15 continuation cards may be used for each statement.

Each statement contains from one to seven fields. Fields are separated by commas. A field consists of one to ten subfields. Subfields are also separated by commas. If a field includes more than one subfield, the field must be enclosed in parentheses. Parentheses are optional if the field consists of only a single subfield. Except where explicitly specified, subfields are limited to eight characters. In general, blank characters are ignored except where otherwise indicated.

Certain statements permit a field or fields to be defaulted (not coded). If there are additional fields to follow, the defaulted field must be indicated by coding a comma.

6.2 STANDARD CONTROL STATEMENT FUNCTIONS

A list of the control statements and a brief description of their function follows.

(1)	RESERVE,BLOCK,A,B	Reserve temporary direct access storage (data sets).
(2)	SAVE,CATLG	Reserve permanent data sets for use in a subsequent job.
(3)	FIND,RELEASE	Access data sets created in a previous job.

(4)	READ,PREAD,WRITE PWRITE,TAPE,PTAPE	Specify device and data formats for tapes.
(5)	EXEC,E	Specify task, input and output data sets and required parameters.
(6)	PARAMS,P	Define a symbolic name for a set of parameters.
(7)	LABEL,L,RELABEL,R	Specify label lines to be added to an output data set.
(8)	NOTE	Print a message on the output listing, and control certain VICAR functions.
(9)	END	Indicate the last control statement for a job step.
(10)	TIME	Set a limit on job CPU time.
(11)	REGION	Set a limit on job main storage utilization.
(12)	NOLIST	Suppress listing input statements on second (X) job.

6.3 CONTROL STATEMENT SPECIFICATIONS

The following specifications define control statement field and subfield requirements. Refer to the examples in Appendix D to clarify the following explanations.

6.3.1 RESERVE, BLOCK, A, B

The RESERVE control statement is used to allocate <u>temporary</u> disk storage space (data sets). A temporary data set is automatically deleted at the end of the job step.

Operation	No. of data sets	Length	No. of records	Volume ID	Data set names
Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
{ RESERVE BLOCK A B }	,n	,(rl [,bl])	,nr	,[volid]	[,(name1, name2, . . .)]

Field	Content
1	RESERVE or BLOCK or A or B. All are equivalent.
2	n is an integer from 1 to 9 specifying the number of data sets to be allocated.
3	rl is the record length (bytes per line). If rl is less than 360, a value of 360 is used to accommodate labels. No image to be stored in the data sets allocated may have a line length exceeding rl. bl is the block length. bl may be omitted, in which case VICAR will use either the largest multiple of rl not exceeding 6447, or rl if it is already greater.
4	nr is the number of records to be allocated in each data set. Sufficient records for data plus labels should be included.
5	volid is the volume serial number of a disk pack, for example, IPLSYS. Alternatively, an asterisk or null field may be coded, in which case the operating system will assign a volume from the public SYSDA device pool.
6	name1,name2,. . . are names to be assigned to the allocated data sets. The names may be from one to eight characters in length. The first character must be alphabetic. Names may be omitted for any or all data sets. Unnamed data sets will be used by VICAR when undefined names appear in subsequent EXEC statements.

A data set is thought of as a place where an image can be stored. When first allocated, a data set is "empty." An image is stored in the data set by an application program as the result of an EXEC statement. When an image is stored in a data set which already contains an image, the new image replaces the old image and the old image is lost. A temporary data set is automatically deleted at the end of the job step.

The user's main concern when allocating a data set is its size (number of records and number of bytes/record). The user must know the sizes of the images with which he is working. Any image which is stored in a data set should not exceed the size of the data set. The required number of records is the number of data lines plus the label records. If the user tries to store an image in a data set which is too small, the job will usually be aborted. If only the number of records is too small, the operating system attempts to allocate additional disk space for the data set equal to 20% of the original number of records. If the attempt is successful, the job continues, but may be using disk space wastefully.

Normally the user does not specify the data set block size, but allows VICAR to calculate it. However, he should be aware that, usually, the larger the block size, the more main storage is required by the programs that process the data set. At the same time, the larger the block size, the lower the overall execution time for the processing programs.

6.3.1.1 Examples

(1) RESERVE,5,1024,500,SPOOL1

This statement allocates five data sets on disk volume SPOOL1. Each data set will consist of 500 records of 1024 bytes each. Optional names have not been specified and the system will therefore assume that these data sets are scratch and available for assignment by VICAR if required.

(2) A,3,(600,7200),700,*,(A,B,C)

This statement will reserve three data sets on disk volumes assigned by the operating system. Each data set will consist of 700 records of 600 bytes each. The records will be grouped into blocks of 7200 bytes each. The data sets will be named A, B and C. The explicit assignment of optional names will cause the system to treat these data sets differently than the scratch data sets in the previous example. To modify a named data set, the programmer must specify the data set name in the output data set field (field 4) of an EXEC control statement.

(3) B,2,1024,1024,IPL304,(X04,Y07)

This statement will reserve two data sets on disk volume IPL304. The data sets will be named X04 and Y07. The names consist of the tape names X and Y with a two-digit file sequence number appended thereto. Again, the system will treat these data sets differently than scratch data sets. If a tape-file name is specified in the input data set field (field 3) of an EXEC control statement, the system will generate a utility task to copy the specified tape-file into the reserved data set, where it will remain for the remainder of the job.

6.3.2 SAVE and CATLG

The SAVE and CATLG statements are used to allocate permanent disk storage space (data sets). They differ only in how data set cataloging is handled.

Operation	No. of data sets	Length	No. of records	Volume ID
Field 1	Field 2	Field 3	Field 4	Field 5
{ SAVE / CATLG }	,n	,(rl [,bl])	,nr	,[volid]

Data set names	Temporary file names
Field 6	Field 7
,[(dsname1, dsname2,. . .)]	,[(fname1, fname2,. . .)]

<u>Field</u>	<u>Content</u>
1	SAVE or CATLG. They are not equivalent.
2	<u>n</u> is an integer from 1 to 9 specifying the number of data sets to be allocated.
3	<u>rl</u> is the record length (bytes per line).
4	<u>nr</u> is the number of records to be allocated in each data set. Sufficient records for labels must be included.
5	<u>volid</u> is the volume serial number of a disk pack, for example, IPLSYS. Alternatively, the field may be an asterisk or empty, in which case the operating system will assign a volume from the storage SYSDA pool, and the data set will be entered in the operating system catalog.

6 __daname1, daname2,__ . . . are names to be assigned to the allocated data sets. A name may be from one to forty-four characters in length if a file name is given and one to eight characters if it is not. Permanent data set names should conform to installation and operating system rules.

7 __fname1, fname2__, . . . are temporary file names associated with the data sets named in field 6. The file name is optional, unless the corresponding data set name exceeds eight characters. The file name must begin with an alphabetic character and must not exceed eight characters.

A permanent data set is used similarly to a temporary data set, except that when the job step terminates it is not deleted. It may be accessed in a subsequent job or job step by using the FIND or RELEASE statement. A permanent data set will remain available to VICAR jobs until processed by the RELEASE statement or deleted by non-VICAR methods.

The concept of the file name allows processing of data sets whose names exceed eight characters. The file name is a short, temporary name assigned to a data set for the duration of the job step. The name by which a data set is referenced in an EXEC statement cannot exceed eight characters, but in a large multi-user environment, longer names are often needed. Therefore, the file name when defined is used for reference in the EXEC statement. (This use of the term "file" should not be confused with the concept of a tape file.)

SAVE and CATLG are very similar; they differ only in the way use is made of the system catalog. They are identical if a volume serial number is omitted from field 5. The operating system selects a volume, and it is entered in the catalog with the data set name. If the volume serial number is specified, the SAVE statement does __not__ cause it to be cataloged; references in subsequent jobs or job steps then require respecification of the volume serial number. CATLG always causes the data set to be cataloged regardless of whether the volume serial number is specified. Use of the system catalog relieves the user of the burden of remembering and specifying the volume serial numbers of his permanent data sets.

6.3.2.1 __Examples__

(1) SAVE,3,600,700,IPL302,(A,B,C)

The above statement will reserve three data sets, named A, B, and C, on disk volume IPL302. All three data sets will be preserved for use in a subsequent job.

(2) SAVE,1,(1000,7000),800,*,JMS.G1

The above statement will allocate a permanent data set named JMS.G1 on a storage disk in the SYSDA pool. The data set will be cataloged in the operating system catalog. The data set will have 800 records of 1000 bytes each. There will be seven records per physical block.

(3) CATLG,1,512,512,,JQP51.TSAVE.DATA3,A

The above statement allocates a permanent data set named JQP51.TSAVE.DATA3 on a storage disk in the SYSDA pool. The data set will be cataloged in the operating system catalog. The data set will have 512 records of 512 bytes each. The data set must be referred to by the name A in EXEC statements.

(4) CATLG,3,1000,1000,JPL008,(JQP.M1,JQP.M2,JQP.M3),(A,B,C)

The above statement allocates three permanent data sets on the disk named JPL008. They will be cataloged in the system catalog with names JQP.M1, JQP.M2, and JQP.M3. Within the current VICAR job step they will be referred to as A, B, and C, respectively, in EXEC statements. Even though the names in field 6 do not exceed eight characters, file names may still be defined in field 7. In this case, only the file names may be used in EXEC statements even though the data set names do not exceed eight characters.

6.3.3 FIND and RELEASE

The FIND and RELEASE control statements are used to access a disk data set created in a previous job or job step. They are required in order to refer to the data set in an EXEC statement.

Operation	Data set names and volume IDs	Temporary file names
Field 1	Field 2	Field 3
{FIND RELEASE}	,(dsname1, vol1, dsname2, vol2,. . .)	[,(fname1,fname2,. . .)]

Field	Content
1	FIND or RELEASE. They are not equivalent.
2	dsname1, vol1, dsname2, vol2, . . . are up to five pairs of data set names and volume serial numbers. The name is identical to the name specified when the data set was allocated. The volume serial number is that of the disk containing the data set. Alternatively, the volume serial number may be entered as an asterisk or omitted if the data set is entered in the operating system catalog.
3	fname1,fname2,. . . are up to five temporary file names associated with the data sets named in field 2. The file name is optional, unless the corresponding data set name exceeds eight characters. The file name must begin with an alphabetic character and must not exceed eight characters.

As with the SAVE and CATLG statement, the file name is a short, temporary name assigned to a data set for the duration of the job step. The file name, required for a data set whose name exceeds eight characters, is used to refer to the data set in EXEC statements. Note that a file name may be required even if it is never referenced, as when deleting data sets whose names exceed eight characters.

The effects of FIND and RELEASE are almost identical. RELEASE causes the data set to be deleted at the end of the job step; FIND causes it to be retained. Use of RELEASE to do processing and deletion in the same job step is obviously dangerous since the processing may fail and the data set deletion would prevent the job from being rerun.

When using FIND or RELEASE on a cataloged data set, the catalog will be ignored if the volume is specified in the statement. With RELEASE, the catalog entry will not be deleted if the volume is specified. It is, therefore, recommended that the volume not be specified with RELEASE when the data set is cataloged.

6.3.3.1 Examples

(1) FIND,(A,IPLSYS,JMS.G1,*)

Data set A on disk volume IPLSYS and cataloged data set JMS.G1 are assumed to have been created in a previous job. The above statement will cause the system to access these data sets and will enable the user to reference the data sets in EXEC statements by their actual names.

(2) RELEASE,(A,IPLSYS,C,IPLSYS)

The above statement will cause the system to access data sets A and C on disk volume IPLSYS and will enable the user to reference the data sets in EXEC statements by their actual names. The data sets will be deleted at the end of the job.

(3) FIND,(JBS.M1,*,JBS.M2,*,JBS.M3,*),(X1,X2,X3)

This statement will make available the three data sets JBS51.M1, JBS51.M2, and JBS51.M3. All the data sets are cataloged in the system catalog so they can be located automatically. They will be referred to by the names X1, X2, and X3 in EXEC statements.

(4) RELEASE, JBS51.TSAVE.DATA3,X

This statement makes available the data set JBS51.TSAVE.DATA3 which is cataloged in the system catalog. This data set and its catalog entry will be deleted at the end of the current job step. The data set must be referred to by the name X in any EXEC statements [illegible]ch appear in the step. Even if it is not referenced in an EXEC [illegible]tement, the name in field 3 is still required because the data set na[illegible] in field 2 exceeds eight characters.

6.3.4 READ, PREAD, WRITE, PWRITE, TAPE, and PTAPE

One of these statements must be included for each tape drive to be used in the job step, whether for reading, writing, or temporary storage. The statement specifies the attributes of tapes to be used on the drive, and assigns a symbolic name for reference in EXEC statements.

Operation	Symbolic device name	Tape ID and data set names	Symbolic tape name	Format code	Blocking data
Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
{ READ PREAD WRITE PWRITE TAPE PTAPE }	,dev	,(volser,name1, name2,. . .)	,tname	,fmt	[,(rl[,bl])]

Field	Content
1	READ, PREAD, WRITE, PWRITE, TAPE, or PTAPE. They are not all equivalent.
2	dev should normally be an asterisk or omitted, causing a unique tape drive to be assigned to the tape. By entering the same symbolic name of up to eight characters for two or more such statements, the same physical tape drive can be assigned to several tapes.
3	volser is the volume serial number of the actual magnetic tape reel desired. It can be up to six alphanumeric characters. This name will appear in a message to the computer operator instructing him to mount the tape.
	name1, name2, . . . are optional names of data sets that are specified in field 4 of EXEC control statements, and are copied to the output tape following execution of the program named in the EXEC statement. This feature does not apply to READ and PREAD.
4	tname is a symbolic name assigned to the input tape. It may be from one to six characters in length and the first character must be alphabetic. This name is used in EXEC statements to refer to the tape.
5	fmt is a one- or two-character format code which specifies the data format of the tape. The valid codes are listed in Appendix A. If field 5 is omitted, the system will assign a default format code.
6	This field is used for blocked tapes and for unblocked tapes whose records exceed 7200 bytes. rl is the record length and bl is the block length, specified in bytes. If bl is omitted, it is assumed to be equal to rl. If both are omitted, a default value of 7200 is assumed for both. If the assumed or specified values of rl and bl are equal, they will be treated as upper limits for the actual size. (Some application programs may be written to ignore the values in this field.)

The total of tape specification statements (READ, etc.) may not exceed an installation-defined limit.

All of the statements -- READ, PREAD, WRITE, PWRITE, TAPE, and PTAPE -- are very similar and can usually be used interchangeably. The differences are described below.

WRITE and TAPE are always equivalent; PWRITE and PTAPE are always equivalent. The "P" prefix (PREAD, PWRITE, PTAPE) causes the last tape used on the designated unit to be made available to the next job step. The tape will not be unloaded from the tape drive when the current job step ends. However, an appropriate tape definition statement is still required in the next job step. Without the "P" prefix (READ, WRITE, and TAPE), the tape is unloaded at the end of the current job step, making the tape drive available to other jobs. For jobs consisting of a single job step, PREAD and READ are equivalent, PWRITE and WRITE are equivalent, and PTAPE and TAPE are equivalent.

The remaining differences are concerned with the VICAR feature called "indirect tape input and output." The VICAR system will "automatically" copy images from tape to disk and from disk to tape if the user wishes. The user can take advantage of this feature by appropriate specification on the tape description statement, and on the EXEC statement. (The use of this feature at the Jet Propulsion Laboratory is very low.)

To cause a tape image to be copied automatically to disk the user must: (1) allocate a temporary disk data set which is either unnamed, or has a name of the form "tnamexx" where "tname" is a symbolic tape name appearing on a READ or PREAD statement and xx is a two digit file sequence number; and, (2) specify an input data set name on an EXEC statement in the form "tnamexx" or "tname/xyz" where "xyz" is a file sequence number descriptor. (See the section on the EXEC statement for more complete information on its syntax.)

To cause a disk image to be automatically copied to tape, the user must enter the name of the data set containing the image into field 3 of a WRITE, PWRITE, TAPE, or PTAPE statement. Each time the data set name appears as an output data set on an EXEC statement, the image written on the data set as a result of the EXEC statement is automatically written onto the tape. The file sequence number depends on the previous position of the tape, and must be inferred from the previous sequence of processing steps. The format of the image is converted to that specified on the WRITE (or equivalent) statement.

If field 2 is defaulted, a unique tape drive is automatically assigned for the tape specified in field 3. For reasons of economy, it is often desirable to "share" the use of a tape drive among several tapes. One method is to specify a unique symbolic name in field 2 of two or more tape-specification statements. This allows all tapes with the same name in field 2 to use the same physical tape drive. Tapes are then unloaded and mounted automatically as called for in the processing tasks. This feature is particularly useful in job steps which require use of a single tape drive for two or more tapes having the same number of tracks, but different formats otherwise. By specifying a different format code in field 5 and assigning the same symbolic name in field 2, the user can assign one tape drive for all tapes. (However, the user should be careful never to assign 7-track and 9-track tapes to the same drive!)

The above method, while useful in certain situations, is not economical in terms of time. A tape is not unloaded until a new tape is called for in an EXEC statement, so processing must necessarily halt while the new tape is mounted. To efficiently conserve elapsed time, the VICAR program VMOUNT should be used for sharing tape drives. A VMOUNT task can be specified immediately after the last use of a tape, allowing the operator to mount the next tape while other processing steps are being executed. This provides for the next tape to be ready when it is called for in an EXEC statement. However, it should be noted that VMOUNT does not allow for any tape format changes.

Tape images may be blocked or unblocked. "Unblocked" means that each data record is written on the tape as one physical block. "Blocked" means that two or more records are combined and written together on the tape as one physical block. The block is the smallest unit of data the tape drive can read or write. Blocking tape data reduces the amount of tape needed for storage, reduces the number of tape operations needed to process an image, and can speed up processing. Under VICAR, the user must describe the blocking information in field 6 of the tape specification statement; otherwise VICAR assumes the tape is unblocked.

6.3.4.1 Examples

(1)
```
READ,*,SCR001,X,8F
```

The above statement will cause VICAR to access tape SCR001 and assign the symbolic name X to this tape. In addition, VICAR will record the format of this tape as 8F.

(2)
```
READ,71,SAVE1,Y,8

READ,71,SAVE2,Z,5A,(510,10400)
```

The first statement above will cause VICAR to access tape SAVE1 and assign the symbolic name Y to this tape. VICAR will record the format of this tape as 8. The second statement will cause VICAR to access tape SAVE2 and assign the symbolic name Z to this tape. In addition, VICAR will record the format of this tape as 5A and will record the record length of 510 bytes and a block size of 10400 bytes. Since the same symbolic device name was used in field 2 of both statements, the same physical tape drive will be used for both tapes. This, of course, implies both tapes are not needed simultaneously for the same task.

(3) WRITE,*,(IPSZ81,C,D,E,F),Z,8F

This statement will cause VICAR to access tape IPSZ81 and to assign the symbolic name Z to this tape. The system will record the format of this tape as 8F. In addition, the data set names C, D, E and F will cause VICAR to create a utility task to copy each of these data sets to the output tape, whenever they appear in the output data set field of an EXEC control statement.

(4) WRITE,*,(JBS005,AA,DD),Z,8F

EXEC,FILTER,(X/1-10),AA,,P1

The above EXEC control statement will create 10 tasks, each of which will filter one frame. The output of each task will be written in disk data set AA. Specification of the data set name AA in the WRITE control statement will cause this data set to be written on the output tape, JBS005, following each execution of the program FILTER.

6.3.5 EXEC, E

The EXEC control statement is used to specify a task or a sequence of tasks. It is also used to specify the input and output data sets as well as the parameters required for the specified tasks.

Operation	Program name	Input data sets
Field 1	Field 2	Field 3
[*] |EXEC| | E |	,pname	{ ,* ,name1 ,(name1,name2, . . .) ,(tname/filesd,name2,name3, . . .) ,(*tname/filesd,name2,name3, . . .)

Output data Sets	Output size parameters
Field 4	Field 5
{ , ,* ,name1 ,(name1,name2, . . .) }	, [(sl, ss, nl, ns) (SL=sl, SS=ss, NL=nl, NS=ns)]

Optional parameters
Field 6
--- , [(param1, param2, . . .)]

Field	Content
1	EXEC or E or *EXEC or *E. The optional asterisk (*) in this field indicates that the task specified is one of a sequence of tasks in a "DO group," and is not the last task in the sequence. See below for a discussion of VICAR DO groups.
2	pname is the name of the program to execute. The name is one to eight alphanumeric characters, beginning with a letter. The program must exist in the VICAR program library. If the first character in the program name is the character "V," automatic label processing is suppressed and VICAR will not write system or user labels on any of the output data sets. In this case, the program itself must write any required system or user labels. Such programs are normally used to process data sets or tapes which are not in any of the standard formats.
3	As shown, there are several optional combinations which may be used to define the input data set field. The number and function of the input data sets is program-dependent. A maximum of 10 data sets may be specified. The field may be omitted entirely, in which case the primary output data set from the last EXEC statement will be used as the input data set. A single asterisk is used to indicate that the task has no input data sets. name1,name2, . . . are names of data sets or tapes or tape files to be used as input images for the task. The format for each is dsname or fname or tname or tnamexx or *tnamexx, where dsname is the actual name of a disk data set, fname is the file name of a disk data set if defined, tname is the symbolic name of a tape, and xx is a

two-digit file sequence number. If the format is tname, VICAR will use the next sequential file following the last one processed on that tape, or 1 if the tape has not been used. The application program will read the image directly from the tape. In that case, the tape format must be logically equivalent to the standard VICAR format (5,6,8, or 9). If the format is tnamexx, VICAR will generate a utility task to copy the tape image to disk and convert the format as necessary. The user must allocate a temporary disk data set with no name, or with the name tnamexx; otherwise the job will fail. If the format is *tnamexx, the tape will be positioned to file xx; the application program will read the image data directly from tape. Again, the tape format must include standard VICAR labels.

tname/filesd is a tape name followed by a slash and a file sequence descriptor. The file sequence descriptor consists of one or more file sequences separated by commas, of the form xxx or xxx-yyy where xxx and yyy are positive decimal numbers of one to three digits each. A single number specifies a single file; two numbers separated by a hyphen specify a continuous sequence of files. Several individual files and/or ranges may be specified separated by commas. When more than one file is specified, a "DO group" is implied (see the explanation below). As with the format for tnamexx, a preceding asterisk implies the tape is read directly by the application task; absence of an asterisk implies indirect input, requiring an unnamed data set to be allocated.

4 A single asterisk is used to indicate that the task has no output data sets.

name1,name2, . . . are up to four disk data set names, file names, or tape names to be used for program output. The number and function of the data sets is program dependent. Tape file numbers may not be specified for output tapes.

This field may be defaulted completely, in which case VICAR assigns a single output data set from among the unnamed data sets allocated. If there are none available, the job will fail.

5 (sl,ss,nl,ns) The output size parameters are optional. If this option is used, all four parameters must be coded as shown. Each parameter is a positive integer from 1 to 99999.

(SL=sl,SS=ss,NL=nl,NS=ns) The size parameters may also be specified in keyword format, as shown. When keywords are used, any or all four parameters may be coded, and the order is of no significance.

nl and ns are, respectively, the number of data records (image lines) and the number of bytes/record to be processed from the input data set. sl and ss are, respectively, the starting data record and starting byte in the input picture at which processing is to begin. These parameters allow the user to specify that a rectangular subsegment of the input image is to be processed. In general, it is advised that the user specify ns if ss has been specified and nl if sl has been specified. The exact interpretation of these parameters is determined by the application program, and may differ from that given here.

When size field parameters are defaulted, values for sl and ss of 1 will be assumed, and values for nl and ns will be assigned equal to the values of these parameters in the primary input data set system label. In this way, the output data set will be equal in size to the input data set.

6 param1,param2, . . . are names of parameter sets as defined in PARAMS control statements. Optionally, they may be any of the types of parameters permitted on parameter statements, providing that each individual parameter is equal to or less than eight characters. Parentheses may be omitted if a single parameter or parameter set name is used. Since the maximum number of subfields within a field is limited, if the desired number of parameters exceeds 10, one or more parameter sets delimited by a PARAMS control statement must be used.

Each program has unique requirements for the parameters which may or must be specified, as well as the number of input and output data sets and their content. Programs may have restrictions on the size of pictures which can be processed. This information must be obtained from the user guide for the individual program.

6.3.5.1 Examples

(1) EXEC,GEN,*,AA,(NL=300,NS=400)

This statement will generate a single task using the program GEN. The asterisk in the input data set field (field 3) specifies that the task has no input data sets. The output data set is named AA and the size is specified as 400 samples by 300 lines. (The default sample size for GEN, as for most VICAR programs, is one byte.)

(2) EXEC,LIST,AA,*,,PLIST

This statement again will generate a single task using the program LIST. The asterisk in the output data set field (field 4) specifies that the task has no tape or disk output data sets.

(3) EXEC,PICAVE,(A,B,C),AA

This statement will also generate a single task using the program PICAVE. The task will have three parallel input data sets, A, B, and C. The output data set is AA. Data set A (the first data set in the input data set field), is the primary input data set. The system will copy the labels from the primary input data set A to the output data set AA, while adding a "history label" containing the program name, PICAVE.

(4) EXEC,STRETCH,A,B,,(LINEAR,60,225)

The above statement shows how the user may include parameters in field 6 of the EXEC control statement. In this case, the restriction is that the field may contain up to 10 subfields separated by commas, and each subfield may contain a maximum of eight characters.

6.3.5.2 DO GROUP. The VICAR DO group provides a way of repetitively executing a fixed sequence of VICAR tasks, using a different input tape file each time the sequence is repeated. This capability is useful when the input data to be processed exists as multiple files on a single tape, and when all processing steps are identical, including data set names and program parameters. A more advanced iterative capability described later employs the preprocessor procedure capability and the library procedure, DO.

The first statement of a VICAR DO group must be an EXEC statement whose first input data set name is of the form tname/filesd or *tname/filesd. The file sequence descriptor, filesd, lists the file numbers from the input tape to be processed. Each time the statements of the DO group are executed, the next file in sequence is used as the first input data set for the first task. The file sequence descriptor is a single file number, or a contiguous set of numbers from xxx to yyy, written as "xxx-yyy," or any combination of these forms separated by commas. (Non-monotonically-increasing sequences cause inefficient motion of the input tape). The complexity of the file sequence descriptor is limited by the VICAR restriction of 10 subfields to a field, and the fact that each comma and slash within a subfield is treated as a subfield separator. File numbers may have from one to three digits each.

The list of tasks which form part of the VICAR DO group is denoted by preceding each EXEC statement in the group except the last with an asterisk. A DO group may consist of a single statement, in which case the leading asterisk is not used at all.

6.3.5.3 Examples

(1)
```
READ,*,XYZABC,X,8F

A,2,1000,1000,*,A

EXEC,FILTER,(X/1-5,7-10,13),A,,WEIGHTS
```

The above statements will generate ten tasks using the program FILTER. The input data sets for the tasks will be files 1 through 5, files 7 through 10, and file 13, all from tape XYZABC. In general, the above configuration is employed with tape files. As shown, a READ control statement with the tape name X is required. VICAR will also automatically generate a utility task prior to each FILTER task. This utility task will copy the specified tape file to disk, performing the necessary format conversion.

The programmer must reserve disk data sets which the system can use. For the above example, the programmer has reserved one scratch (unnamed) data set which will be reused by the system. Another alternative would be to reserve named data sets with names matching the above files (i.e., X01, X02, X03, etc.). The system will always direct the utility task output to an appropriately named data set, if available.

(2)
```
READ,*,SCR001,X

READ,*,SCR002,Y

B,2,500,1000,*,A

EXEC,ICOR,(*Y/1-20,X03),A,,ICORPAR
```

The above statements will generate 20 tasks using the program ICOR. The asterisk preceding the tapename Y indicates the direct tape input option. VICAR will assign the primary input data set for the ICOR program to tape SCR002. On the other hand, the secondary input data set (tape file X03) is not preceded by an asterisk. The system will, therefore, generate a utility task to copy file 3 from tape SCR001 into a disk data set using one of the data sets allocated in the "B" statement. Once this file is on disk, it will then be available for each of the 20 ICOR tasks.

To use a tape file directly, without copying it to disk, its format must be exactly equivalent to that of a disk file. As shown above, the formats on the READ control statements have been defaulted. Assuming the default format is equivalent to the disk format, it would have been possible to include an asterisk prior to X03 in the input data set field. However, considering the tape rewind time, it might be preferable to have the secondary input file X03 on disk, since it is used in each of the 20 ICOR executions.

6.3.6 PARAMS,P

The PARAMS control statement is used to define a symbolic name for a set of parameters.

Operation	Parameter set name
Field 1	Field 2
|PARAMS| | P |	,psname

<u>Field</u>	<u>Contents</u>
1	PARAMS or P
2	<u>psname</u> is a symbolic parameter set name of from one to eight characters.

The parameters to be included in the parameter set being defined are listed on one or more parameter statements immediately following the associated PARAMS statement. The parameter statements are terminated by the appearance of another VICAR control statement of any type.

Parameters may start in column 1 and cannot extend past column 71. Blanks or commas are used as separators between parameters. Blanks and commas may be used in literal parameters which are enclosed between apostrophes. Parameters may be continued on as many cards as required. <u>No</u> continuation character should be used in column 72. A parameter cannot be split between cards. Parameter may be integer, real, alphanumeric, hexadecimal, or literal, as listed:

<u>Integer Parameters</u> - Positive and negative integer parameters may consist of an optional sign digit and up to seven decimal digits.

<u>Real Parameters</u> - Positive and negative real parameters may be represented in the format

$$\pm n_1\ n_2 \text{ --- } n_m \cdot n_p \text{ --- } n_r\ E{\pm}e_1e_2$$

(e_1 and e_2 are both required)

or

$$\pm n_1\ n_2 \text{ --- } n_m \cdot n_p \text{ --- } n_r$$

where n_1 through n_r, e_1 and e_2 are decimal digits, and $\pm e_1e_2$ represents the power of 10 by which the number is to be multiplied.

Alphanumeric Parameters - A parameter containing one or more alphabetic characters is padded with blanks on the right, or truncated, to exactly eight characters.

Hexadecimal Parameters - They are represented by an X and one to eight hexadecimal digits enclosed within apostrophes. The parameter is stored in a four-byte field padded with zeros on the left if less than eight digits.

Literal Parameters - A character string of any length up to 69, including numbers, blanks, and special characters, may be enclosed within apostrophes. The character string, less apostrophes, will be padded on the right if necessary to a length which is a multiple of four.

One or more symbolic parameter set names may be included in field 6 of the EXEC control statement. In this case, the symbolic name must be specified in a PARAMS control statement followed by a number of parameter records. The system will incorporate the specified parameter sets into the task queue.

6.3.6.1 Example

```
PARAMS,P2
TRANS,555 666 777 SCALE 989.05
PARAMS,P4
'THIS IS A SPECIAL TITLE FOR A SPECIAL TASK', 42
EXEC,PROGA,A,B,,(P2,P4)
```

As shown, two parameter sets, P2 and P4 are specified in the above EXEC control statement. The parameters will be presented to the program in the order specified, P2 followed by P4.

6.3.7 LABEL,L,RELABEL,R

The LABEL control statement is used to add a label line of arbitrary text up to 68 characters to the output data sets of an EXEC statement. The RELABEL statement has the same effect except that all previous labels are deleted (except the system label). Either statement follows the EXEC statement to which it applies. In the case of a DO group, only the first EXEC statement can be followed by LABEL or RELABEL statements.

Operation	Data set name	Text
Field 1	Field 2	Field 3
{ LABEL / L / RELABEL / R }	,[{ name / * }]	, text

Field	Contents
1	LABEL or L, RELABEL, or R. L is equivalent to LABEL; R is equivalent to RELABEL.
2	_name_ is a one to eight character symbolic name which is used to add more than one label line to a specific data set. A single asterisk is used to indicate that the label line is to be added to the output data sets for each iteration in a DO group.
3	_text_ may consist of from one to 68 characters. Text may not extend beyond column 71. LABEL and RELABEL control statements _cannot_ be continued, but may be repeated.

In the case of a DO group, only the first EXEC statement of the group may be followed by LABEL or RELABEL statements. If an asterisk appears in field 2, the labels are applied to the output images for each iteration of the DO group. If no asterisk appears, the first LABEL or RELABEL following the EXEC is used on the first iteration, the second LABEL or RELABEL on the second iteration, and so forth until all have been used or the number of iterations specified is completed.

6.3.7.1 _Examples_

(1)
```
E,SAR,A,B
L,1, THIS IS LINE 1 OF MY TITLE
L,1, THIS IS LINE 2 OF MY TITLE
L,1, THIS IS LINE 3
```

In this example, 3 lines of text are added to the history label of data set A as it is written out on data set B. If the user had wished to center the title he could precede each line with blanks, as in the next example.

```
(2)    E,SAR,A,B
       R,1, THIS IS LINE 1 OF MY TITLE WHICH
       R,1, I WISH TO CONTINUE ON TO LINE 2.
```

In this case, the previous history labels of data set A will be replaced by the lines of text specificed in the RELABEL statements.

6.3.8 NOTE

The NOTE control statement is used to print a message on the output listing. It is also used to control certain VICAR functions.

Operation	Text
Field 1	Field 2
NOTE	,text

Field	Contents
1	NOTE
2	*text* may consist of one to 64 characters.

Certain text strings are significant to the VICAR system, as follows:

NOTE, CONTINUE causes the system to continue executing tasks, even if a subsequent task terminates abnormally.

NOTE, ABORT causes the job to be terminated if a subsequent task terminates abnormally.

NOTE, ABORT and NOTE, CONTINUE may be inserted anywhere and repeated as necessary in the sequence of EXEC statements to control VICAR response to abnormal task terminations. At the beginning of the job, ABORT is in effect.

NOTE, WTO causes a message to be displayed to the computer operator at the beginning of each task. The message includes the program and job names. This feature allows a user to follow the progress of a VICAR job in execution if he has access to a system console.

NOTE, PLAB allows program parameters and the "G" history label to be inserted into the VICAR label. NOTE, NOBLAB is the default and inhibits the parameters and "G" label from being recorded in the history label. Either statement may be alternately specified at any point(s) in the sequence of EXEC statements in a VICAR job.

6.3.9 END

The END control statement is used as a delimitier to terminate a job step. If the job consists of a single step, END may be omitted.

Operation
Field 1
END

Field	Contents
1	END

6.3.10 TIME

The TIME control statement is used to set the maximum CPU time allowed for the second of the pair of VICAR jobs. This statement has meaning only for operating systems with job step timing. If no TIME statement appears, an installation-defined default is assumed (see Appendix C).

Operation	Maximum Time
Field 1	Field 2
TIME	,time

Field	Contents
1	TIME
2	TIME is an integer giving the number of minutes set as the time limit for the second VICAR job. In systems with job step timing, the job will be aborted (ABEND code S322) after this interval.

6.3.11 REGION

The REGION control statement is used to set the size of the region of main storage to be used by the second of the pair of VICAR jobs. The effect of exceeding the specified region depends on the version of the operating system in use. In MVT systems, the job will be aborted. If no REGION statement appears, an installation-defined default is assumed.

Operation	Maximum Time
Field 1	Field 2
REGION	,size

Field	Contents
1	REGION
2	size is the requested region size in units of 1024 bytes, specified as an integer followed by the letter K. If the requested size is an odd number, the next higher even number is used.

6.3.11.1 Example

REGION,250K

6.3.12 NOLIST

This statement suppresses the listing of the VICAR input statements on the second job ("X" job).

Operation
Field 1
NOLIST

Field	Contents
1	NOLIST

6.4 Preprocessor Statements (EVIL)

Preprocessor statements are those which are detected and processed by the preprocessor step of the VICAR procedure. There is little significance to the separate processing except that implementation is simplified. However, the general syntax rules of the preprocessor statements are not always the same as for standard statements so they are described separately. Preprocessor statements use all 80 columns of the card or card image. Blanks generally are significant except for trailing blanks.

For historical reasons, the preprocessor is sometimes called EVIL, and preprocessor statements are called EVIL statements.

The preprocessor statements constitute a general macro capability within VICAR. This capability has found a great deal of use in repetitive processing of large amounts of data using similar or identical algorithms.

6.4.1 DEFINE,D

The DEFINE control statement introduces a body of VICAR control statements which are to constitute a procedure or macro. The statement names the procedure, and defines zero or more substitutable character-string arguments. The body of the procedure consists of any number of VICAR statements following the DEFINE statement. The procedure definition is terminated by an END statement or another DEFINE statement. Therefore, END or DEFINE statements may not appear in the procedure body. Except for the search for another DEFINE or END statement, the procedure body is not scanned for validity at the time it is defined. Thus, text which would not constitute a valid VICAR statement may appear in a procedure.

Operation	Procedure name	Argument	Argument
Field 1	Field 2	Field 3	Field 4
DEFINE D	,procname	[,argument1]	[,argument2]

Field	Contents
1	DEFINE or D
2	procname is the name of the procedure being defined. Any character string is valid.
3,4. . .	argument1,argument2, . . . are any number of character string arguments. Each argument should appear in the body of the procedure, and will be replaced by a character string parameter in a subsequent CALL statement.

6.4.2 END

The END control statement is used as a delimiter to terminate a procedure definition.

Operation
Field 1
END

Field	Contents
1	END

This statement is syntactically identical to the END statement which terminates a VICAR job step definition. The effect of an END statement depends on whether a procedure is being defined.

6.4.3 CALL,C

The CALL statement is used to invoke a procedure which has been previously defined using the DEFINE statement. All statements from the procedure body will be copied into the job stream, and substitutions of character-string parameters will be made. CALL statements may be nested and recursive.

Operation	Procedure name	Parameter	Parameter
Field 1	Field 2	Field 3	Field 4
|CALL| | C |	,procname	[,parameter1]	[,parameter2]

Field	Contents
1	CALL or C
2	procname is the name of a procedure which must be previously defined. However, the effect of invoking an undefined procedure is specifically defined to be null. No error messages are given in this event.
3,4. . .	parameter1, parameter 2,. . . are character string parameters which are to be substituted for corresponding arguments in the DEFINE statement. That is, parameter1 is substituted for argument1, parameter2 is substituted for argument2, etc. If the number of parameters does not match the number of arguments, excess parameters will be ignored, while missing parameters will be assumed to have null value. Virtually any character string not containing a comma is a valid parameter. Parameter strings are delimited by the surrounding commas, and may therefore contain leading, trailing or embedded blanks, except that the last one may not have trailing blanks. Null strings are valid parameters.

6.4.4 GET,G

The GET control statement is used to access one or more sequences of statements stored on the VICAR procedure library. These stored sequences normally, but need not, consist of one or more procedures headed by DEFINE statements. It is important to understand that the GET statement does not invoke a procedure; it merely retrieves text from the library. A subsequent CALL statement must be used to invoke a procedure which is retrieved by a GET statement. It is also important to understand that the name under which the text is stored in the library is normally, but need not be, the same as the name of the procedure which the text defines. See Section 7 for a description of how to store procedures in the library.

Operation	Text name	Text name
Field 1	Field 2	Field 3
GET G	,textname1	[,textname2]

Field	Contents
1	GET or G
2,3,. . .	textname1,textname2,. . . are names of text files in the VICAR procedure library. Normally the text name is the same as the name of the procedure defined in the text. Each textname must have been previously stored in the library.

6.5 USE OF PREPROCESSOR STATEMENTS

The following subsections provide examples of the use of the preprocessor statements DEFINE, CALL and GET. Use of the DO library procedure is introduced and examples are provided.

6.5.1 The DEFINE Control Statement

The user may include DEFINE control statements to introduce and name a procedure, or macro, consisting of any VICAR statements except DEFINE and END. The DEFINE statement may also list character string parameters which are to be replaced when the procedure is invoked by a subsequent CALL statement. Procedure definitions must precede procedural invocation. Once defined, the procedure definition is effective until redefinition of the same procedure name in another DEFINE statement, or until the end of the job.

6.5.1.1 Example

```
D,STRCLIP,FILE,BITS
E,SAR,(*X/FILE),A
E,STRETCH,A,B,,(CLIP,BITS)
D,STRLIN,LOWDN,HIDN,NLX,NSX
E,STRETCH,A,B,,(LINEAR,LOWDN,HIDN)
L,,LINEAR(LOWDN-HIDN)
E,BOXFLT,B,C,,(NLW,NLX,NSW,NSX)
L,,LOW PASS FILTER (NLX BY NSX)
END
```

In this example, two procedures are defined; one called STRCLIP and the other called STRLIN. The first is terminated by the DEFINE statement introducing the second. STRCLIP has two arguments, FILE and BITS, and STRLIN has four arguments, LOWDN, HIDN, NLX, and NSX. STRCLIP consists of two EXEC statements which will have the effect of executing the two application programs SAR and STRETCH. The syntax of the "E,SAR. . ." is not valid in that the characters following the "/" should be a one-to-three digit number. Also, the STRETCH user guide would show that the parameter following the keyword "CLIP" should be a number. These apparent errors are acceptable provided that when the procedure STRCLIP is invoked by a CALL statement, the arguments FILE and BITS are replaced by numbers.

6.5.2 The CALL Control Statement

The user may include CALL statements that invoke a procedure and specify its parameters. The procedure must have been previously defined, either by its appearance in the sequence of VICAR control statements, or by being obtained from the procedure library using the GET statement. The effect of the CALL statement is similar to invoking a macro in assembly language. The parameters provided in the CALL statement are substituted for the corresponding (positional) arguments in the procedure definition. The expanded procedure text is then substituted for the CALL statement in the sequence of VICAR statements. Of course the original procedure definition is unchanged and is available for repeated CALLing with different parameters.

If the name of the procedure in the CALL statement has not been previously defined, the effect is to ignore the statement. Even though there is a procedure library, there is no automatic searching of the library. The situation is not considered an error and no diagnostic message is produced. For this reason spelling errors in procedure names can have startling effects which may be hard to diagnose. The user must keep this feature of procedure usage in mind and use appropriate care.

Building on the example shown in Section 6.5.1, suppose that that sequence has appeared in a VICAR job. The following statements could then appear:

```
CALL,STRCLIP,1,3
C,STRLIN,50,150,3,3
```

The effect of these statements is the same as if the following sequence had appeared in their place:

```
E,SAR,(*X/1),A
E,STRETCH,A,B,,(CLIP,3)
E,STRETCH,A,B,,(LINEAR,50,150)
L,,LINEAR(50-150)
E,BOXFLT,B,C,,(NLW,3,NSW,3)
L,,LOW PASS FILTER (3 BY 3)
```

6.5.3 The GET Control Statement

The user may include GET control statements to retrieve sets of VICAR statements which have previously been saved on the VICAR (EVIL) procedure library. (The process of storing text on the procedure library is not done by VICAR control statements, but by a TTM procedure. This process is discussed in detail in Section 7. Because the CALL statement does not cause automatic searching of the library, each procedure CALLed, but not defined in the job, must be explicity retrieved from the procedure library by means of the GET statement.

Although the procedure library stores arbitrary sets of statements, common practice is that each set of text constitutes one procedure, beginning with a DEFINE statement and ending with an END statement. Each set of statements stored in the EVIL library is stored under a unique name by which it is retrieved, the name specified in the GET statement. Common practice is to have the name by which a set of statements is stored equal to the name of the procedure that is defined by that set of statements. Thus, to retrieve the procedures STRCLIP and STRLIN, which have been previously stored on the library, the following statement is used:

```
GET,STRCLIP,STRLIN
```

It is not unreasonable nor uncommon to store under a single name the definitions of two or more procedures which are commonly used together. This simplifies usage by requiring the user to GET only one set of text from the library, even though a number of procedures are to be used.

6.5.4 The DO Library Procedure

The DO procedure provides an iterative capability which is much more powerful than the simple DO group described in Section 6.3.5. The DO procedure allows the user to repeatedly invoke any other defined procedure, while the first parameter to the other procedure takes on, in sequence, each of a set of values specified. The DO procedure is on the procedure library and may be invoked after it has been retrieved with a GET statement.

The general form of the DO procedure usage is as follows:

```
CALL,DO,procname,arglist,arg2,arg3, . . .
```

Procname is the name of a previously defined procedure. *arg*2, *arg*3, . . . are arbitrary character strings not containing commas. *Arglist* is a specification of a list of parameters of the form:

```
argl/argl'/argl"/. . .
```

If the argument list consists of positive integers in ascending numerical sequence, then

```
arg1/arg1+1/arg1+2/argl+3/. . . /argl'
```

may also be written

```
argl-argl'
```

The alternative forms may be mixed using a slash (/) as a separator to produce a list of the following or similar form:

```
argl/argl'-argl" . . .
```

The effect of the DO procedure is to invoke the procedure with name *procname* a number of times equal to the number of items specified by *arglist*.

```
CALL,procname,argl,arg2,arg3,. . .
CALL,procname,argl',arg2,arg3,. . .
CALL,procname,argl",arg2,arg3,. . .
                 .
                 .
                 .
```

or

```
CALL,procname,argl,arg2,arg3,. . .
CALL,procname,argl+1,arg2,arg3,. . .
CALL,procname,argl+2,arg2,arg3,. . .
                 .
                 .
                 .
```

The second and subsequent arguments to the specified procedure are always the same, and are *arg2*, *arg3*, etc. The first time the specified procedure is CALLed, its first argument is the first item in the list specified by *arglist*. The second time it is CALLed, its first argument is the second item in the list, and so on until the last item in the list has been used.

6.5.4.1 *Examples*

Suppose the procedure STRCLIP has been defined as in Section 6.5.1, and the following statements then appear:

```
(1)  G,DO
     CALL,DO,STRCLIP,3-5/7,3
```

The effect of these statements is as if the following had been coded:

```
E,SAR,(*X/3),A
E,STRETCH,A,B,,(CLIP,3)
E,SAR,(*X/4),A
E,STRETCH,A,B,,(CLIP,3)
E,SAR,(*X/5),A
E,STRETCH,A,B,,(CLIP,3)
E,SAR,(*X/7),A
E,STRETCH,A,B,,(CLIP,3)
```

In this case, the same result may also be obtained using the DO group.

```
*E,SAR,(*X/3-5,7),A
E,STRETCH,A,B,,(CLIP,3)
```

In the preceding example only the repetitive feature of a DO procedure is illustrated. A more powerful feature of the DO procedure is the ability to combine repetitive steps with variation of processing parameters. Consider the following example:

```
(2)   G,DO
      D,STRF,LO
      E,STRETCH,A,B,,(LINEAR,LO,150)
      E,FOTO,B,*
      END
      CALL,DO,STRF,90/100/110
```

In this example, the data set A is contrast enhanced using three different variations of the low-DN parameter in program STRETCH through the use of the STRF procedure.

These statements are equivalent to the following:

```
E,STRETCH,A,B,,(LINEAR,90,150)
E,FOTO,B,*
E,STRETCH,A,B,,(LINEAR,100,150)
E,FOTO,B,*
E,STRETCH,A,B,,(LINEAR,110,150)
E,FOTO,B,*
```

SECTION 7

PROCEDURE LIBRARY OPERATIONS

User-defined EVIL procedures which perform repetitive processing tasks can be stored in the VICAR procedure library. This feature allows the user to access procedures in subsequent job steps without redefining the procedure each time.

A separate facility exists for manipulating the user-defined statement sequences (text files). This facility is not part of VICAR, but is an adjunct to it. There are four statements which are used to manipulate user-defined VICAR statement sequences (text files). These are SAVE, DELETE, LIST and NAMES. Each statement is defined in subsequent sections of this text.

The computer job to perform library operations has the following general form:

```
//jobname JOB . . .
// EXEC TTM
#<EVIL2LIB>
    .
    .
    .
library manipulation statements
    .
    .
    .
```

7.1 SAVE

The SAVE statement is used to save a user-defined text file in the VICAR procedure library under a specified text name.

Operation	Text name	Terminator
Field 1	Field 2	Field 3
SAVE	,textname	,ENDIN=delimiter

Field	Contents
1	SAVE
2	textname is the name under which the text file is to be stored in the VICAR procedure library. The maximum number of characters for textname is 20.

Field	Contents
3	ENDIN=delimiter. The delimiter is a user-specified character or characters which are used to terminate the text file contents. A typical delimiter might be $$$$. The delimiter must fit onto one computer card; however delimiters should be kept to a reasonable size.

7.2 DELETE

The DELETE statement is used to delete a text file and its associated text name from the VICAR procedure library.

Operation	Text name	Text name
Field 1	Field 2	Field 3
DELETE	,textname1	,textname2

Field	Contents
1	DELETE
2,3 . . .	textname1, textname2, . . . are the names of text files in the VICAR procedure library. The maximum number of characters for each name is 20.

7.3 LIST

The LIST preprocessor statement is used to list the contents of the text file with the specified textname in the VICAR procedure library.

Operation	Text name	Text name
Field 1	Field 2	Field 3
DELETE	,textname1	,textname2

Field	Contents
1	LIST
2	textname1, textname2, . . . are the names of text files in the VICAR procedure library. The maximum number of characters for each name is 20.

7.4 NAMES

The NAMES statement is used to list the text names of all the text files currently in the VICAR procedure library. The NAMES statement should always be used to check for duplicate text files before SAVE or DELETE is specified.

Operation
Field 1
NAMES

Field	Contents
1	NAMES

7.5 EXAMPLES OF PROCEDURE LIBRARY OPERATIONS

7.5.1 Example 1

```
//NAMES JOB (DPM701,23)
// EXEC TTM
#<EVIL2LIB>
NAMES
//
```

This job lists the text names of all the text files currently in the VICAR procedure library. A job such as this should always be run prior to saving a new text file to ensure that text names will not be duplicated. If a job using NAMES is not run, an error message will appear indicating that duplicate text names are being specified.

7.5.2 Example 2

```
//SAVE TEXT JOB (DPM701,23)
// EXEC TTM
#<EVIL2LIB>
SAVE,PROCS3,ENDIN=$$$$
B,2,1000,1000,*,(A,B)
D,P1,&T,&F
E,INSERT,(*&T/&F),A
E,ASTRTCH2,A,B
E,MASK76,B,VFC,,COMP
END
D,P2,&T,&F
E,INSERT,(*&T/&F),A
E,FILTER,A,B
E,MASK76,B,VFC,,COMP
END
```

```
D,P3,&T,&F
E,INSERT,(*&T/&F),A
E,ROTATE,A,B
E,MASK76,B,VFC,,COMP
END
$$$$
//
```

Under the text name 'PROCS3', a BLOCK data set statement and three procedures are stored as a text file in the VICAR procedure library. In subsequent jobs, a 'G, PROCS3' VICAR control statement will retrieve the text file, making all three procedures available to the 'CALL' VICAR control statement. The 'B' VICAR statement will always be processed.

7.5.3 Example 3

```
//USETEXT JOB (DPM701,23)
// EXEC VICAR
TAPE,*,INTAPE,S,6
TAPE,*,SCRVFC,VFC,6
G,PROCS3
C,P1,S,1
C,P1,S,2
C,P1,S,6
//
```

In this example, only procedure P1 is used out of the available procedures P1, P2, and P3. The tape INTAPE with symbolic name S is used in the procedure call for the parameter &T.

SECTION 8

TTM

The preprocessor function #<EVIL2LIB> used in the previous section is invoked through the use of TTM, a recursive language designed primarily for character string manipulation. Users may employ TTM directly to obtain greater flexibility and economy in the manner in which their VICAR procedures are created and executed. For example, TTM provides a means for the conditional inclusion of VICAR statements within a procedure when the procedure is called and executed. The following discussion and examples are intended to present a brief introduction to the TTM function calls and their usage. (See Reference 4 in Section 2 for a more thorough description of TTM.)

The basic form of a TTM function call is:

```
#<S1;S2;S3; . . .;Sn>
```

where S1,S2,. . .,Sn are character strings and $n \geq 1$. The first string, S1, is taken to be the name of the function and the remaining strings are its parameters. TTM functions may be user-defined or TTM standard functions. Among the operations provided by the TTM standard functions are basic arithmetic as well as logical and numeric comparisons.

In general, a TTM function call other than #<EVIL2LIB> may be placed anywhere in the VICAR text but should not begin in column one. In addition, the characters # ; < and > have special meaning in the TTM language. The user should consult the TTM language reference mentioned above for correct usage. It should be noted that a TTM function call that is placed inside a VICAR procedure is not invoked unless the procedure is called via a CALL control statement. When invoked, a TTM function call returns a character string which then replaces the call itself in the VICAR text. That is, #<S1;S2;...;Sn> is replaced by the character string that is generated by the function operation.

8.1 EXAMPLES

(1) Arithmetic Operations

Suppose a user wishes to process the first 25 even-numbered files on a tape with symbolic name S using VICAR procedure P.

```
D,P,&F
E,INSERT,(*S/&F),A
    .
    .
    .
END
```

To perform the necessary processing, the procedure P would have to be called 25 times:

```
C,DO,P,2/4/6/8/. . ./50
```

Using the TTM standard library function MU (multiply), the above process may be simplified by writing the procedure P as follows:

```
D,P,&F
E,INSERT,(*S/#<MU;&F;2>),A
    .
    .
    .
END
```

When the procedure P is called, the effect of the function MU is to take the parameter substituted for &F, multiply it by two, and leave the result in place of the text of the function call. For example,

```
C,P,7
```

results in the statement sequence

```
E,INSERT,(*S/14),A
    .
    .
    .
```

The first 25 even-numbered files may thus be processed by the statement:

```
C,DO,P,1-25
```

TTM function calls may have as their arguments other TTM function calls. To illustrate this, suppose that the user wishes to process the first 25 odd-numbered files on a tape. To accomplish this, the procedure P may be rewritten as:

```
D,P,&F
E,INSERT,(*S/#<SU;#<MU;&F;2>;1>),A
    .
    .
    .
END
```

The calls to P would again be

```
C,DO,P,1-25
```

Here, the SU (subtract) and MU (multiply) TTM standard library functions are combined to produce the file number given by (&F) X 2 - 1.

(2) Logical Comparisons

Consider Example 2 of Section 6.6.5. The only difference in the three procedures P1, P2, and P3 is the second statement in each case. Using the TTM standard library function EQ? (logical comparison: equality), the three procedures may be combined into a single procedure. The form of the EQ? function is given by

```
#<EQ?;S1;S2;S3;S4>
```

where the result of the function is S3 if S1 is equal to S2. Otherwise, the result of the function is S4. As before, S1, S2, S3, and S4 are character strings. A suitable procedure might be:

```
D,P,&T,&F,&P
E,INSERT,(*&T/&F),A
  #<EQ?;&P;A;E,ASTRTCH2,A,B;>
  #<EQ?;&P;F;E,FILTER,A,B;>
  #<EQ?;&P;R;E,ROTATE,A,B;>
E,MASK76,B,VFC,,COMP
END
```

By adding a third parameter &P to the procedure definition, the appropriate program to be executed at the second step may be specified through the use of the EQ? function. For example, the procedure call

```
C,P,S,5,A
```

would result in

```
E,INSERT,(*S/5),A
E,ASTRTCH2,A,B
E,MASK76,B,VFC,,COMP
```

Note that S4 is a null string in this particular usage of the EQ? function. In the above procedure call then, the logical comparison with F and R yield null strings.

SECTION 9

ERRORS

In a system as complex to use as VICAR, error situations occur frequently. This section gives some guidance on what to do if a problem occurs.

Errors may be categorized as occurring during the first job or the second job. Errors which occur during the first job are almost invariably due to incorrect syntax in one or more VICAR statements. These are caught by the program VTRAN during its execution, and a helpful message is printed. Normally the second job is suppressed. Errors which occur during the second job usually result in abnormal termination of the job, accompanied by a user or system "completion code." A user code means the error was detected by the VICAR system, while a system code means the error was detected by the operating system.

9.1 VTRAN ERRORS (FIRST JOB)

The most common kinds of errors occurring during the first VICAR job are given below, along with some suggested actions to take.

PARENTHESIS ERROR
Parentheses are unbalanced.

TOO MANY FIELDS
The maximum number of fields in a VICAR statement is seven. Look for an extra comma or a missing set of parentheses.

TOO MANY SUBFIELDS
The maximum number of subfields within any one field is ten. Subfield delimiters are comma and slash. Look for an extra comma.

ILLEGAL VTRAN CARD
There is an unclassifiable syntax error.

PARAMETER CARD PUNCHED IN COLUMN 72
Column 72 of a parameter card must be blank.

LABEL AT END OF DO GROUP
LABEL IN MIDDLE OF DO GOUP
Only the first task of a DO group may be labeled. Use of preprocessor features may allow the desired processing without using a DO group.

ILLEGAL SIZE FIELD
Look for an extra or missing comma if the size field looks correct.

NUMBER FIELD ON RESERVE CARD NOT BETWEEN 1 AND 9
The field referred to is the second field on a RESERVE, A, B, BLOCK, SAVE, or CATLG statement.

MORE THAN 8 TAPE DATA SETS REQUESTED
The total number of READ, WRITE and TAPE statements may not exceed eight. (Implementation restriction)

TOO MANY PARAMETER SETS SPECIFIED
The total number of PARAMS and P statements may not exceed 200. (Implementation restriction)

NO DATA SET AVAILABLE FOR OUTPUT
An asterisk may have been unintentionally omitted from a tape name in an EXEC statement. The use of the "tape-file no." form of a data set name without an asterisk requires the definition of a suitable data set with a RESERVE-type statement.

LABEL CARD ENCOUNTERED UNEXPECTEDLY
The data set name field may have been omitted from a LABEL or RELABEL statement.

9.2 EXECUTION ERRORS (SECOND JOB)

Errors resulting in abnormal termination during the second job may be either user errors or program errors. Program errors usually must be solved by the application programmer responsible for the program which terminated. User errors can be corrected by changes to VICAR statements in the user's job. Distinguishing between user errors and program errors can be quite difficult, and may ultimately depend on the intentions of the programmer as to how his program should work.

The most common user completion codes associated with VICAR jobs are given below, along with possible user errors. (Since this is not a programmer's guide, the associated possible program errors are not given.)

USER 69
There was insufficient main storage available for buffers. Try a larger region size.

USER 71
An attempt was made to write a record larger than the allocated record size. Be sure the data set record size specified is large enough to accomodate the picture being processed.

USER 72
Insufficient disk space allocated and no more space is available. Increase allocation and be sure to allow for labels.

USER 73
Write error under unusual conditions. Not normally a user error.

USER 240
This code is produced when the ABEND is intercepted by the Fortran run-time routine. The actual ABEND code is found in another printed message.

USER 324
The application program intentionally terminated abnormally, and "NOTE, ABORT" was in effect. There should be an associated explanatory message produced by the program. The predominant reason is an error in parameters specified for the program. Be sure the spelling of names of parameter sets on the EXEC statement matches the spelling on the PARAMS statement.

USER 999
The requested program was not in the program library or was not executable. Check correct spelling.

USER 1111
The op code in PS44 SVC simulation routine is illegal. This is not a user error. (This code has not been observed with this version of VICAR.)

USER 1112
The VICAR system index for a data set is invalid. Be sure all the required data sets have been specified for the program.

USER 1200
The SYSOUT system file cannot be opened. This is not a user error. (This code has not been observed with this version of VICAR.)

USER 1492
The VICAR data set reference number is illegal. This is not normally a user error.

USER 1970
There is an error in the task list passed to the second VICAR job. This would occur if an error occurred in the first job but the second job was not suppressed. Look for a syntax or usage error in the VICAR control statements.

USER 1980
The VICAR system encountered an I/O error in processing parameters or data set labels. Be sure the correct data sets and tapes are specified as input files, that the correct tape format is specified and that the tape does not have a permanent I/O error in the label data.

USER 2400
The maximum number of tape units requested exceeds the installation limit. This error is normally caught in the first job.

APPENDIX A

TAPE FORMAT CODES

Format Code	Description	Comments
9 8 5 6	These formates are logically identical, formats 9 and 6 representing nine-track tape at 800 and 1600 bpi respectively, and formats 8 and 5 representing seven-track tape at 800 bpi and 556 bpi respectively.	Standard VICAR formats
8A 9A 6A	These formats are for 7 track, 800 bpi, 9 track, 800 bpi, and 9 track, 1600 bpi, 8 bit data, unlabelled[a] tapes respectively.	Used for film recorder 8-bit data tapes.
8F 5F 2F	These formats are all logically equivalent, with 8F, 5F, and 2F representing seven track 800 bpi, 556 bpi and 200 bpi respectively. The data format is 6-bit, unlabelled.[a]	Used for film recorder 6-bit data tapes.
8L 5L 2L	These formats are all logically equivalent, 8L, 5L and 2L representing seven-track 800 bpi, 556 bpi and 200 bpi, respectively. The data format is 6-bit with 80 character labels.	Used for obsolete Surveyor 6-bit data tapes.

[a] Throughout this document, "label" means VICAR label. It should not be confused with operating system standard labels, which are not supported by the VICAR system.

Note: These format codes are used on tape declaration statements described in Section 6.3.4.

APPENDIX B

JCL INSERTIONS IN SECOND VICAR JOB

It is occasionally necessary to modify JCL statements generated by the VICAR system. JCL procedure statements may be overridden using standard operating system methods (see Ref. 2, Section 2). These methods cannot be applied to the second VICAR job because the JCL is generated by a program, VTRAN. However, JCL statements can be inserted in the second job in either of two locations, just before and just after the ". . . EXEC PGM=VMAST" statement. This is accomplished by supplying the statements to be inserted as either of two data sets processed by the first job.

To insert statements _before_ the EXEC statement, the following sequence is used.

```
//VTR.FT08F001 DD DATA
              .
              .
              .
JCL statements to be inserted
              .
              .
              .
/*
```

To insert statements _after_ the EXEC statement, the following sequence is used.

```
//VTR.FT10F001 DD DATA
              .
              .
              .
JCL statements to be inserted
              .
              .
              .
/*
```

Either of the above sequences is placed in the job deck following the VICAR control statements. If both sequences are used, the "FT08" sequence precedes the "FT10" sequence.

As an example, JCL insertions may be used to obtain a core dump after a job ABEND. The following sequence inserted after the VICAR control statements provides a dump.

```
//VTR.FT10F001 DD DATA
//SYSUDUMP     DD SYSOUT =A
/*
```

Similarly, a private program library will be searched ahead of the standard library if the following sequence is included.

```
//VTR.FT10F001 DD DATA
//STEPLIB      DD DSN=privatelibrary,DISP=SHR
//             DD DSN=IPL1.SDSRUN,DISP=SHR
/*
```

Both can be combined to perform a private library search and dump if the following is specified.

```
//VTR.FT10F001 DD DATA
//STEPLIB      DD DSN=privatelibrary,DISP=SHR
//             DD DSN=IPL1.SDSRUN,DISP=SHR
//SYSUDUMP     DD SYSOUT=A
/*
```

APPENDIX C

JPL IMAGE PROCESSING LABORATORY

CONVENTIONS AND DEFAULTS

This appendix provides information about the use of VICAR that may vary from one installation to another. The information in this appendix applies to the JPL Image Processing Laboratory as of the date of publication.

C.1 JOB CARD

The job card, which is the first JCL statement of a batch job, is written as described in Reference 2 of Section 2. Generally, it appear as follows:

```
//jobname JOB acct,'programmer name', . . .
```

where

<u>jobname</u> is one to eight alphanumeric characters beginning with an alphabetic character,

<u>acct</u> is the accounting field described below,

'<u>programmer name</u>' may be anything enclosed in single quotes, and <u>. . .</u> are other optional parameters as described in the reference.

No embedded blanks are allowed except before and after JOB.

The accounting field has the general form

```
(uidprj,boxno,time,lines,cards,forms,copies,log,linect)
```

where

<u>uid</u> is a three-character user identification assigned by the facility (usually the user's initials),

<u>prj</u> is a two- or three-digit project code assigned by the facility,

<u>boxno</u> is the number of the user's output box assigned by the facility,

<u>time</u> is the estimated elapsed execution time in minutes for the job (default:30),

<u>lines</u> is the estimated number of lines to be printed in units of a thousand (default:3),

<u>cards</u> is the estimated number of cards to be punched (default:100),

forms is the name of a special form to be used for printing the job (default:STD.),

copies is the number of copies of the job to be printed (default:1),

log is N to omit the HASP log from the job listing, and

linect is the number of lines to be printed between page ejects, or 0 to omit page ejects (default:60).

All fields are optional except uidpri and boxno. Commas must be entered to indicate defaulted fields preceding a nondefaulted field. Trailing commas within the parentheses may be omitted.

C.2 DATA SET NAMES

The names of permanent data sets should begin with the user's three-character user identification, which is assigned by the facility (usually the user's initials).

C.3 TAPE FORMAT CODE

The default tape format code is 8 (7-track, 800 bpi, 8-bit data, VICAR labels present).

C.4 TAPE SPECIFICATION STATEMENTS

The maximum number of tape specification statements (READ, PREAD, WRITE, PWRITE, TAPE, PTAPE) is eight.

C.5 TAPE DRIVES

The maximum number of tape drives which can be used in a VICAR job step is eight.

C.6 REGION SIZE

The default region size is 150K.

C.7 CPU TIME

The maximum CPU time a job step can use is 1439 minutes (one day less one minute).

APPENDIX D

EXAMPLE JOBS

Example 1

```
            //EXAMPLE JOB (AYS900,45,10),'AYS BOX-45',REGION=160K
            // EXEC VICAR

11  <------ REGION ,160K
 8  <------ NOTE ,WTO
            TAPE,TI,EHT602,S,6
 4  <------ TAPE ,TI,SCRGRE,GRE,9
 3  <------ FIND ,(JQP.940.DATA,ERA001),WB
 1  <------ B ,2,1000,1060,*,(A,B)
            E,INSERT,*SO3,A,(100,425,1040,1000)
 7  <------ L ,1,HUMBOLT COUNTY AREA
            L,1,WATERSHED BOUNDARIES
            L,1,LANDSAT DATA
 5  <------ E ,F2,(A,WB),B,,PF
 6  <------ P ,PF
            FUNCTION,'IN1+IN2'
            E,SAR,B,S
            E,ASTRTCH2,B,A,,(GAUSS,GSIG,2.5)
            E,MASK76,A,GRE,,COMP
            //
```

This is an example of a simple VICAR job using several of the control statements defined on subsequent pages. It has been provided to familiarize the new user with the general format of a VICAR job. Control statements have been boxed and connected with a number at the left which corresponds to the general definition of function provided in Section 6.2. Using the FIND statement, the job accesses a data set which was saved on a disk in a previous job and assigns this data set the file name "WB." An image file is also read from tape and this image is then summed with WB using the application program F2. A Gaussian contrast enhancement is applied to the composite image output from the program F2 and written onto tape for playback on a film-recording device.

Example 2

Refer to this example for:

Use of a private library for accessing a special version of a VICAR program.

```
//PD05 JOB (DPM701,23),'DPMADURA'
/*MESSAGE 15 MIN
/*MESSAGE 9 TRK=ERTS-2029 ERTS-2172 I4
// EXEC VICAR
NOTE,WTO
TAPE,*,ER2029,S,6
B,3,1500,1500,*,(B4,B7,R)
E,INSERT,(*S/1),B4,(800,1000,1000,1000)
E,INSERT,(*S/4),B7,(800,1000,1000,1000)
D,F,(B7,B4),R
E,LIST,R,*
E,VMOUNT,S,*,,ER2172
E,INSERT,(*S/1),B4,(800,1300,900,700)
E,INSERT,(*S/4),B7,(800,1300,900,700)
E,F,(B7,B4),R
E,LIST,R,*
//VTR.FT10F001 DD DATA
//STEPLIB DD DSN=REA708.LIBRARY,DISP=SHR
// DD DSN=IPL1.SDSRUN,DISP=SHR
/*
//
```

Example 3

Refer to this example for:

Use of preprocessing statements GET, DEFINE and CALL.

Use of a single tape drive for 2 tapes of different densities.

```
//LBU18A JOB (DPM701,23),'D P MADURA'
/*MESSAGE 30 MIN
/*MESSAGE 9TRK=TFC210 N4 SCRSAV(2,
// EXEC VICAR
G,DO
NOTE,WTO
TAPE,T1,TFC210,S,6
TAPE,T1,SAVDM2,GRE,9
B,3.900,4600,*,(V1,V2,V3)
B,3,900,4600,*,(S1,S2,S3)
E,INSERT,(*S/1),V1
E,INSERT,(*S/2),V2
E,INSERT,(*S/3),V3
D,P,&N
E,MASK76,S&N,V&N
END
E,ASTRTCH2,V1,S1,,(GAUSS,GSIGMA,2.0)
E,ASTRTCH2,V2,S2,,(HPER,20.0,LPER,1.0)
E,ASTRTCH2,V3,S3,,(PERCENT,12.0)
C,DO,P,1-3
E,FORGRE,(V3,V2,V1),GRE,,2MIL
E,VMOUNT,GRE,*,,SAVDM3
E,FORGRE,(V1,V2,V3),GRE,,2MIL
//
```

Example 4

Refer to this example for:

Use of FIND to access a data set saved in a previous job.

Use of TTM to increment parameters.

Use of preprocessor statements GET, DEFINE and CALL.

```
//PIXGR2 JOB (AYS900,45,180),'AYS BOX-45',REGION=150K
// EXEC VICAR
REGION,150K
G,DO
READ,*,ER2501,S,6
FIND,(AYS.CORPNT,*),CP
B,5,2000,2400,*,(A,B,C,D,E)
D,P,&B
E,STRETCH,A,B,,P&B
P,P&B
TABLE 0,0.0,#<EQ;&B;1;;#<SU;&B;1>,0.0>,&B,1.0,#<AD;&B;1>,0.0,255,0.0
E,F2,(B,C),D,,PY
P,PY
FUNCTION 'IN1*IN2'
E,SAR,D,S
E,LIST,D,*
END
E,INSERT,*S01,C
E,VMOUNT,S,*,,ER2223
E,INSERT,*S04,A,(400,600,1000,700)
E,SAR,CP,S
E,SAR,CP,A,(400,600,1000,700)
E,VMOUNT,S,*,,ER2502
C,DO,P,43-64
//
```

Example 5

Refer to this example for:

Use of FIND to access a data set saved in a previous job.

Use of preprocessor statements GET, DEFINE and CALL.

Use of a private library.

```
//RERSLOC JOB (GWG314,31)
// EXEC VICAR,DISP=SHR
G,DO
TAPE,,Q14340,EDR,6
B,1,800,805,,A
FIND,(GMY.RESL,*),RESL
B,1,4000,100,,RES
D,PROC,FILE
E,VGRLOG,EDR,A,,(PIC,FILE)
E,RESLOC,(A,RESL),RES
END
C,DO,PROC,1620503/1620515/1620519
E,VMOUNT,EDR,*,,Q14350
C,PROC,1620535
E,VMOUNT,EDR,*,,Q14370
C,DO,PROC,1620950/1621006/1621010
//VTR.FT10F001 DD DATA
//STEPLIB DD DSN=VGR.SDSRUN,DISP=SHR
// DD DSN=IPL1.SDSRUN,DISP=SHR
/*
```

Example 6

Refer to this example for:

Use of TTM

Use of EVIL2LIB to save a procedure

Use of EVIL2LIB to access a procedure saved in a previous job.

```
//LOGRAW JOB (GWG314,31)
// EXEC TTM
#<EVIL2LIB>
DELETE,VGRLOGRAW
SAVE,VGRLOGRAW,ENDIN=???
G,DO
D,CATRAW,&FDS,&SC
CATLG,1,800,802,VGR004,VGR&SC.F&FDS.RAW,F&FDS
E,VGRLOG,EDR,F&FDS,,(PIC,&FDS,PRINT)
END
D,FNDRAW,&FDS,&SC
FIND,(VGR&SC.F&FDS.RAW,*),F&FDS
E,VGRLOG,EDR,F&FDS,,(PIC,&FDS,PRINT)
END
???
//

//LOG15 JOB (PLJ314,32)
// EXEC VICAR,DISP=SHR
TAPE,,011340,EDR,6
G,VGRLOGRAW
C,DO,FNDRAW,1385048/1385050/1385052,1
//
```

Example 7

Refer to this example for:

Use of TTM

Use of EVIL2LIB

Testing of VICAR jobs

The user has included a test of the procedure FARPHOTMAP at the end of his job. Since data sets C and D have deliberately not been allocated, his job will not execute, but he will be able to examine the procedure FARPHOTMAP for errors.

```
//EVIL JOB (JAM317,34),FARPHOTMAP
// EXEC TTM
#<EVIL2LIB>
DELETE,FARPHOTMAP
SAVE,FARPHOTMAP,ENDIN=$$$
NOLIST
FIND,(IPLLSF,*),LSF
FIND,(GMY,RESL,*),RESL
B,2,2000,1010,,(A,B)
B,1,1000,4000,,IDS
B,1,3600,10,,PDS
READ,,#<TAPE>,X,6,(1600,6400)
B,2,3600,10,,(R,G)
D,P,FNO,SLONX,SUNLONX
E,INSERT,(*X/FNO),B
E,RESLOC,(B,RESL),(R,G)
E,RESSAR77,(B,R),A,,HALF
E,FARENC,(A,G),B,(NL=#<NLF>,NS=#<NSF>),FAR
L,RDR ON #<TAPE>/FNO     GEOMA(JUST TO FOOL SEARCV)
E,PHOTFUNC,B,A,,(PHOT,SOLAR,#<SUNLAT>,SUNLONX,SLON,SLONX)
L,,SUBSOLAR LAT #<SUNALT>LONG SUNLONX *LAMBERT PHOTOMETRIC FUNCTION
E,F2,A,B,,PF
E,MAP4,(B,IDS,PDS),A,(NL=#<NLM>,NS=#<NSM>),(MAP,LONG,SLONX,SLON,SLONX)
L,,SUB S/C LAT #<SLAT> LONG SLONX RANGE #<RMAG>
E,CCOPY,(A,LSF),TA
END
D,COPY
E,VREWIND,TA,*
E,VCOPY,TA,*,,(LIST,9X)
END
P,PHOT
HALF JUPI VGR INCR 5 LAXIS #<DV;#<NLF>;2>. SAXIS#<DV;#<NSF>;4>.
MINN 1. FARE NOSE
NORA #<NORA> SLAT #<SLAT> RMAG #<RMAG>
SSCPT #<DV;#<NLF>;2>.   #<DV;#<NSF>;4>.
P,FAR
SUNA #<SUNA> FAR #<RMAG>   LATI #<SLAT>
```

Example 7 (contd)

```
HALF AUTO GEOM 3 ANGL #<NORA> REQU 71400. RPOL 66773.
P,PF
FUNCTION 'IN1/10.' HALF OUTBYTE
P,MAP
NORA #<NORA> SLAT #<SLAT> RMAG #<RMAG>
NOPR SSCPT #<DV;#<NLF>;2>. #<DV;#<NSF>;4>.
LATI 75. FOCL 1502.38 REQ 71400. RPOLE 66773. FARE
LORA 0. PSCA 84.821 MERC SCAL #<SCALE> LINE 1. SAMP #<SAMP>
$$$
NAMES
// EXEC VICAR,DISP=SHR
 #<DS;SUNA;50.>
 #<DS;SCALE;10.>
 #<DS;TAPE;123>
 #<DS;SAMP;88.>
 #<DS;NORA;1.>
 #<DS;RMAG;2.>
 #<DS;NLF;800>
 #<DS;NSF;300>
 #<DS;SUNLAT;4.>
 #<DS;NLM;444>
 #<DS;NSM;500>
 #<DS;SLAT;80.>
G,FARPHOTMAP
C,P,10,22.,33.
C,Q,1,2.,3.,4,5,6,7
E,SAR,C,D
//
```

Example 8

Refer to this example for:

Use of TTM

Use of private libraries

Use of option to dump program in case of ABEND

```
//T1297A JOB (JAM317,34,500),'FARPHOTMAP REV 1',REGION=150K
/*MESSAGE EST 2 HOURS 9 TRK JS2608 9 TRK JS2614
// EXEC VICAR,DISP=SHR
WRITE,,JS2614,TA,6,(360,3600)
 #<DS;SUNA;257.>
 #<DS;NORA;165.>
 #<DS;RMAG;57963900.>
 #<DS;SLAT;3.25>
 #<DS;SUNLAT;.77>
 #<DS;TAPE;JS2608>
 #<DS;NLF;400>
 #<DS;NSF;800>
 #<DS;NLM;700>
 #<DS;NSM;300>
 #<DS;SCALE;400.>
 #<DS;SAMP;116.>
G,FARPHOTMAP
C,P,1,69.31,46.32
E,FOTO,A,*,,(ASTR,SIZE)
C,P,3,141.37,118.37
C,P,4,143.31,120.31
C,P,5,213.43,190.43
C,P,6,215.37,192.36
C,P,13,-74.50,-97.51
C,P,8,-72.56,-95.57
C,P,9,-2.44,-25.45
C,P,10,-.50,-23.52
C,P,11,69.61,46.59
C,P,12,71.55,48.53
C,COPY
//VTR.FT10F001 DD DATA
//STEPLIB DD DSN=JAM825.LIBRARY,DISP=SHR
//         DD DSN=VGR.SDSRUN,DISP=SHR
//         DD DSN=IPL1.SDSRUN,DISP=SHR
//SYSUDUMP DD SYSOUT=A
/*
//
```